



University College of North Denmark

AP Degree Program - Computer Science

DMAJ0914 – Group 1

Project Report

Group Members:

Andrej Ivankov

Ieva Jonikaite

Miroslav Pakanec

Teodor Dimitriev

Karolis Ramanauskas

Supervisors: Nadeem Iftikhar and Simon Kongshøj

Submission date: December 14th 2015



AP Computer Science / DMAJ0914 – Group 1

Abstract:

This project consists of preparing, designing and implementing the C# language software and database for the company, with aspects from technology, system development and programming points of view.

The software is developed with an open layered architecture, where the client is used to control every operation of the system.

Preface:

We are third semester students of the Computer Science course. The group consists of five members, and the age of teammates varies between 19 and 31. All of us are Europeans from different countries – Slovakia, Bulgaria, Croatia, and two members from Lithuania.

The target in our report is the project task where we are creating a completely new software system which handles communication between users who would like to rent an accommodation (students), and those who would like to rent out an accommodation (landlords). The system will be able to handle all relevant information regarding users, flats, applications and confirmations.

Problem statement:

Looking for an accommodation is always difficult, and especially if person is moving to another city or even country. If we focus on students who are trying to find an accommodation near their study location, we can detect some issues that exist in Aalborg itself - there is no centralized system in English which provides adequate information about accommodation for students. This brings issues like: lack of sufficient information about system of renting, it is impossible to check the apartment itself before person moves in¹.

How can we design software that finds an accommodation for foreign students in Denmark?

¹ Used in the System Development report as well

Table of contents:

1	INTRODUCTION.....	2
2	PRELIMINARY PLANNING	3
2.1	DETAILED PROBLEM STATEMENT	3
2.2	RELATED WORK	4
2.3	REQUIREMENTS.....	4
3	SYSTEM ARCHITECTURE.....	6
3.1	DOMAIN MODEL	6
3.2	ARCHITECTURE	7
4	TECHNOLOGY	8
4.1	COMMUNICATION	8
4.1.1	<i>Public service</i>	8
4.1.2	<i>Private service</i>	8
4.2	CONCURRENCY	9
4.3	SECURITY.....	9
4.3.1	<i>Sensitive information</i>	9
4.3.2	<i>User authentication</i>	10
5	DESIGN AND IMPLEMENTATION	11
5.1	RELATIONAL MODEL & SCHEMA.....	11
5.2	DATA LAYER	12
5.2.1	<i>Secured passwords</i>	12
5.2.2	<i>Stored procedures</i>	12
5.2.3	<i>Performance</i>	13
5.3	DATA ACCESS LAYER (DAL)	13
5.3.1	<i>Connection</i>	13
5.3.2	<i>CRUD operations</i>	13
5.3.3	<i>Transaction and concurrency management</i>	14
5.4	BUSINESS LAYER.....	15
5.4.1	<i>WCF</i>	15
5.4.2	<i>Server functionality</i>	16
5.4.3	<i>Host</i>	18
5.5	CLIENT PRESENTATION LAYER	18
5.6	CLIENT LAYER	18
5.7	PERFORMANCE TESTING.....	19
6	CONCLUSION.....	20
7	REFERENCES.....	21
8	APPENDIX	23

1 Introduction

The report is focused on the creation and implementation of a software system for connecting landlords and students. The report has been separated into 4 parts, which are:

1. Preliminary planning
2. System architecture
3. Technology
4. Design and implementation

The hardest aspect of starting a project after identifying a problem is to gather all the necessary information and make a research before even starting the process of creating a solution. We started with a preliminary planning in which we decided to make a small market research, and find out all of the requirements that system has to fill.

After gathering all the information in the preliminary planning part, we started to build architecture of our system. In this part we have focused on how the appearance of the system and which aspects of the system are going to be the most important.

Third part is focused on technology aspect of the system – which requirements are the most important for the system and what are we going to do to achieve them.

The final and the biggest part of our report is design and implementation part. In this part we focused mostly the appearance and reasons and process of implementation of previous parts, and the reasons why we didn't implement certain points in those parts. Also, in this part we tried to see whether we managed to solve our problem statement or not.

2 Preliminary planning

The preliminary investigation of the project consisted of several different areas of interest – related work and requirements.

In the first area, the related work, we researched the market. Based on the research we put stress on what to focus on and what to avoid.

Second area, the requirements, is derived from the first area. Here the main focus is put on all the requirements that system has to satisfy. The requirements are split into two sections: non-functional and functional requirements of the system.

2.1 Detailed problem statement

Looking for an accommodation is not always a simple task –people have to find a place that would match their lifestyle, have a convenient placement in the urban structure of the city, have a reasonable living cost, etc. This process becomes significantly harder if people decide to move to another city or a country. The language barrier alone causes a lot of struggling and in most cases people cannot physically visit the place if they are still in their home country. Sometimes people and accommodation companies are their only source of information.

The city of Aalborg is no exception. Every year there is more and more young people from foreign countries coming to study, the development of this trend has made accommodation search even more difficult. If we take a closer look at the problem, we will see that there is a clear communication problem between landlords and tenants. There are a lot of flats available, but information about them is scattered in many places. Also, in many cases the information provided is in Danish and therefore it is complicated for foreign students to understand the information. Furthermore, there is a demand from students to live near the center of the city or their study location. This causes a long queue for certain flats, while other ones are ready to be rented out. Not only that, but this influx happens largely only two times per year – the beginning of the semesters, which is around September and February.

The final problem we have noticed is the cultural difference. Foreigners do not know about the Danish laws concerning accommodation –the dates when it is possible to move, how many days before moving out is it mandatory to sign the flat out, how is sub renting system working or other practical information. That led us to a question - How can we design software that makes accommodation search for foreign students in Denmark easier?

2.2 Related work

In the beginning of the project, we focused on the research of already existing companies – ePortals for renting. So in this part of the process we have tried to contact the companies to check what are their main issues and/or advantages of their software.

We were lucky and got feedback from two companies - AKU-Aalborg² and BoligPortal.dk which answered 6 different questions³ and gave us a valuable feedback on the following topics:

- Email, Address with zip code and city, and phone number are a must both for people who are renting out and people who want to rent flats;
- Flats need to have as much relevant information as possible; most of them include flat location, rent, deposit, map location and description;
- Flat search option is one of the most important functions of their systems – it provides business value to their companies;
- Archive of all users and contracts is stored in the database;
- English language is important, as more and more of their users are non-Danish citizens.

After getting this side of the story, we also made a research by contacting students who were looking for a flat. Their main feedback was that the system is supposed to be user-friendly, secure and it would be simple to confirm contracts between tenants and landlords through it.

2.3 Requirements

Analyzing the feedback we have determined the requirements for the system. We split the requirements into non-Functional and Functional requirements.

Functional requirements include:

- English language support
- Registration of users
- Option to edit users profile
- Adding flats
- Option to edit flats
- Search - finding the accommodation if only one attribute is given (f.e. city)
- Adding flats to wish list
- Confirming contracts
- Making archive of information

² <http://aku-aalborg.dk/> - Anvisning af Kollegie- og Ungdomsboliger / Company for providing accommodation for students only

³ Questions and answers can be seen in Appendix

Non-Functional requirements include:

Usability*	<ul style="list-style-type: none"> • The user does not have to possess more than basic computer skills in order to operate the system • The interface must be easy to navigate and user friendly, allowing swift procedures, requiring a maximum of 4 clicks in order to complete
Performance*	<ul style="list-style-type: none"> • System must support several hundreds of users browsing simultaneously. • Communication with the database should take less than 10 seconds. • The application shouldn't take more than 2 seconds to load and crucial processes must be quick and reliable.
Security*	<ul style="list-style-type: none"> • All user passwords must be encrypted upon storing to the database. • Confidential user information is only accessible by authorized users. • Registered users are the only ones able to access the system and use its functions.
Availability	<ul style="list-style-type: none"> • The application should be available 99% of the time. However, server outages must be taken into consideration. • The system should not be taken offline for upgrades without notice to all users. • The application must be able to be accessed from any web browser.
Scalability	<ul style="list-style-type: none"> • The program is to be used by students for searching flats and landlords for submitting the flats. Therefore it must allow further expansion for an increasing number of users. • It must allow functional scalability for easy implementation of new features
Maintainability	<ul style="list-style-type: none"> • System must allow easy code changes in order to meet new requirements
Backup	<ul style="list-style-type: none"> • A backup database must be created to ensure that none of the information gets lost. • The application must update the backup database as often as possible every day.

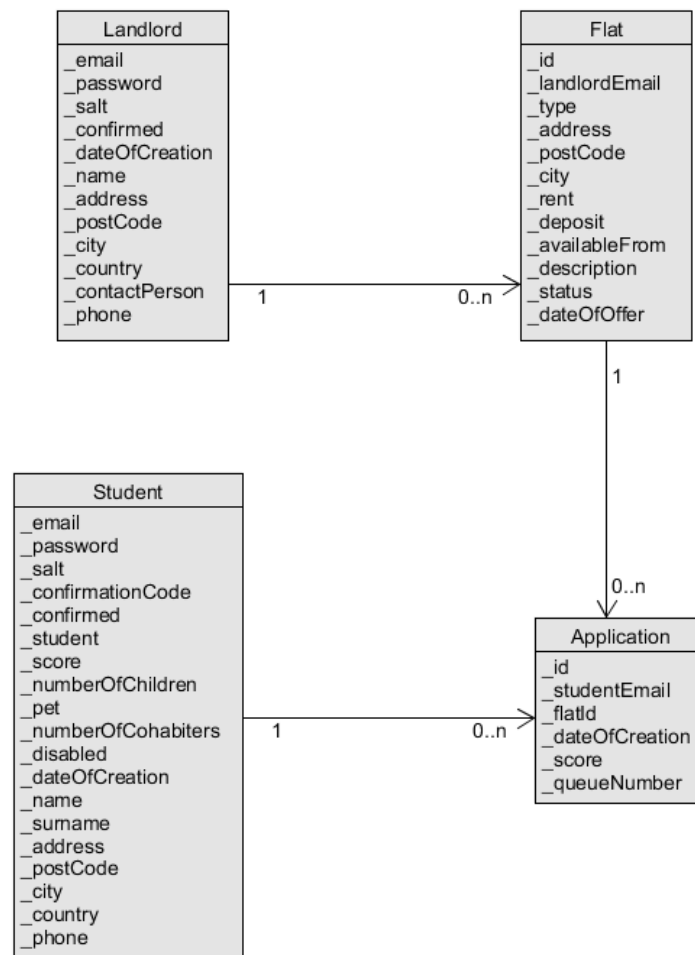
Neither functional nor non-functional requirements are set in an order of priority, they are both set in a way that system has to have all of them. The main difference is that non-functional requirements are actually split into two categories – must have (the ones marked with *) and need to have. The difference is that the must have requirements are mandatory to be satisfied by our project, while the need to have requirements are rather a great addition to the system that we will try to implement.

3 System architecture

During the whole process of creating the project we discussed what the system should look like and why is it better to implement certain aspects in a particular way. In this section, we focused on the domain model and the final architecture of the system.

3.1 Domain model

Landlords and students can access the system through the web application. In our domain model Landlord and Student are represented in separate classes since they can access different parts of the system, perform different operations and have some different attributes. Putting them both in the same class would add a bunch of unnecessary complexity and confusion. Landlords can create, read and update flats, while students can apply for them. The Application class takes information from both the student and the flats. It also displays queue number, which is based on the profile score of the student and the date the user applied for the flat.



Picture 1. Domain model

3.2 Architecture

Architecture is one of the most important parts of the system that is going to be developed, but also enable that the system is adaptive for all of the changes in the future. For our architecture we have discussed Classic Client/Server Architecture, Classic Web Architecture and N-tier (multi-tier) Architecture.

We decided that our distributed application is developed using the 3-tier architecture. That means that our system consists of three main components, which communicate with each other by passing messages. Even though these components are running on the same physical machine, they are designed to be able to run individual processes, therefore can be considered distributed. Each tier has a unique role in the system:

1. Presentation Tier - user accesses this tier directly, using UI or web page.
2. Business Tier - handles majority of application logic (Queue algorithm, etc.)
3. Data Tier - external data source (Database)

Individual tiers have been broken further into layers. While Tier (usually) represents a physical machine, a layer is, however, a logical software component grouped by functionality.

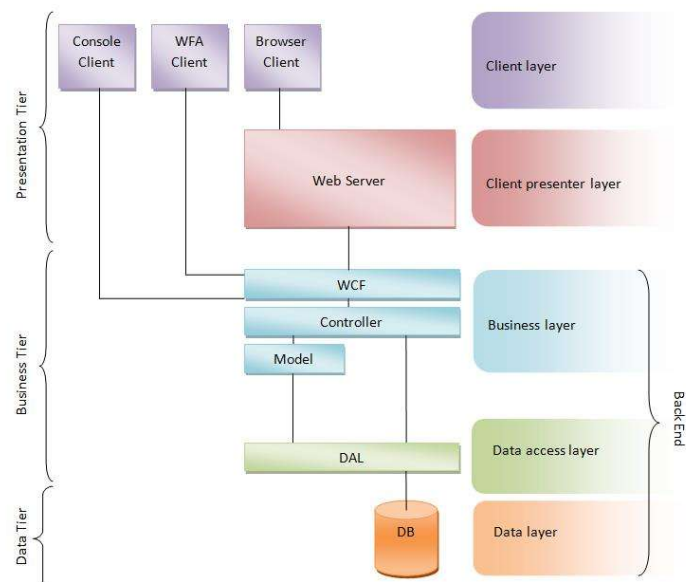
- Client layer - is involved with user directly. Ours contains 3 implementations of client - browser client, windows form application and console client.
- Client presenter layer - holds presentation logic - how is the data going to be presented. In our system ASP.NET represents this layer.
- Business layer - handles application logic.
- DAL (Data access layer) - handles connection to the database, reading and writing into the database.
- Data layer - External data source (Database). In our system also handles simple logic using procedures.

Advantages of using this architecture:

- Adaptable security
- Scalability
- Maintenance
- Reusability

Disadvantages of using this architecture:

- Performance
- Cost



Picture 2. Architecture design of our system

4 Technology

Having architecture is only the first step to create a functional system. Another part is different technologies used to complete the step. In this chapter we focused on technologies used in our system and how we have dealt with certain parts of them.

Our main focus in this part will be the communication, concurrency and security. However we would like to emphasise that we are going to refer more in detail into these in Design and implementation section as well.

4.1 Communication

Windows Communication Foundation (WCF) separates how the software for an application is written from how it communicates with other software. Binding specifies how to communicate with the endpoint. This includes:

- The transport protocol to use (for example, TCP or HTTP).
- The encoding to use for the messages (for example, text or binary).
- The necessary security requirements (for example, SSL or SOAP message security).

In our project we have separated communication for public and private services.

4.1.1 Public service

Since this service should be accessible over the internet the most appropriate would be the usage of binding using http protocol. Some of the data sent from client to server might be confidential so we had to make sure that the communication is secure. We have decided to use message level security over transport level security.

One of the drawbacks of transport level security is that if the message must travel through an intermediary before reaching its destination, there is no way to ensure that transport security has been enabled for the step after the intermediary. On the other hand, message level security is ensuring the integrity and privacy of individual messages, without regard for the network. Through mechanisms such as encryption and signing via public and private keys, the message will be protected even if sent over an unprotected transport (such as plain HTTP).

When choosing appropriate binding we have taken into account the requirements listed above. We have decided to use wsHttpBinding which is a secure, reliable, interoperable binding suitable for non-duplex service contracts. The binding implements the following specifications: WS-Reliable Messaging for reliability, and WS-Security for message security and authentication. The transport is HTTP, and message encoding is Text/XML encoding.

4.1.2 Private service

Our private service is designed for the administrator. Administrator is to be located on the intranet therefore we have decided to choose net.tcp binding.

By default, netTcpBinding offers improved performance over an HTTP binding and is the ideal choice for cross-machine communication between WCF clients and a WCF service, in an intranet environment. Transport security is the default security mode for netTcpBinding.

4.2 Concurrency

Since multiple clients can access resources at the same time, and we have allowed concurrency calls on WCF, we had to find another way how to deal with its issues.

At the beginning we were very pessimistic. We only considered locks as a way how to deal with concurrency. Its solution is simple - there is no concurrency. Individual threads are simply waiting for each other. This could easily bring down scalability of the application, when threads could be stack in the queue for a large period of time.

Second implementation was using transactions. Transactions are ACID (atomic, consistent, isolated and durable). A transaction consists of a single command or a group of commands that execute as a package. Transactions allow combining multiple operations into a single unit of work. If a failure occurs at one point in the transaction, all of the updates can be rolled back to their pre-transaction state.

1. Atomic: If all parts of the transaction individually succeed then data will be committed and the database will be changed. If any single part of a transaction fails then whole transaction will fail and the database will remain unchanged. Part of the transaction might fail for various reasons like: business rule violation, power failure, system crash, hardware failure, etc.
2. Consistent: Transaction will change the database from one valid state to another valid state following various database rules like various data integrity constraints (primary/unique key, check/not null constraint, referential integrity with valid reference, cascading rules) etc.
3. Isolation: One transaction will be hidden from another transaction. In another words, one transaction will not affect another transaction if both work concurrently.
4. Durability: After a transaction is successfully completed (committed to the database), changed data will not be lost in any situation like system failure, database crash, hardware failure, power failure, etc.

4.3 Security

One of the most important non-functional requirements for our system is most definitely security. To protect users for not losing any of their information, we have focused on 2 main fields of security – sensitive information and user authentication.

4.3.1 Sensitive information

Most of the data is sensitive, as we are dealing with confidential information. To determine which kind of information should be classified sensitive, we have decided to divide it by threat types. We have determined three main types of threats:

- Identity theft
- Credit card fraud
- Corporate espionage

4.3.2 User authentication

User authentication is crucial for our system. Users which did not pass the authentication are not able to use the system. In our case the most reasonable way to go would be the "specific knowledge" type of authentication. First we need to deal with a very common and difficult problem - how are we going to store passwords and maintain them secured?

It is unacceptable to store passwords in exact form as they were given. People with access to the database would automatically have access to all of the passwords and if somebody unauthorized got into it, we save them a lot of effort. Encryption would be another way to go. It would make the passwords hidden, but anybody who has access to the encryption key would also have the access to passwords.

We have decided to choose password Hashing - using a cryptographic hash function, which example you can see below.

```
hash("hello") = 2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824  
hash("hblllo") = 58756879c05c68dfac9866712fad6a93f8146f337a69afe7dd238f3364946366  
hash("waltz") = c0e81794384491161f1777c232bc6bd9ec38f616560b120fda8e90f383853542
```

However if we only hash the passwords, they are still vulnerable to several types of attacks and cracking methods such as Lookup tables, Reverse Lookup tables or Rainbow table. The solution to these problems would be adding salt to the hash algorithm. Lookup tables and rainbow tables only work because each password is hashed the exact same way. If two users have the same password, they'll have the same password hashes. We can prevent these attacks by randomizing each hash, so that when the same password is hashed twice, the hashes are not the same. To achieve this, we add a random string, called salt, to the password before hashing. We need the salt, in order to check whether the password is correct or not. Salt will be either stored along with the hash or as a part of the hash string itself.

```
hash("hello") = 2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824
```

```
hash("hello" + "QxLUF1bgIAdeQX") =  
9e209040c863f84a31e719795b2577523954739fe5ed3b58a75cff2127075ed1
```

```
hash("hello" + "bv5PehSMfV11Cd") =  
d1d3ec2e6f20fd420d50e2642992841d8338a314b8ea157c9e18477aaef226ab
```

```
hash("hello" + "YYLmfY6lehjZMQ") =  
a49670c3c18b9e079b9cfaf51634f563dc8ae3070db2c4a8544305df1b60f007
```

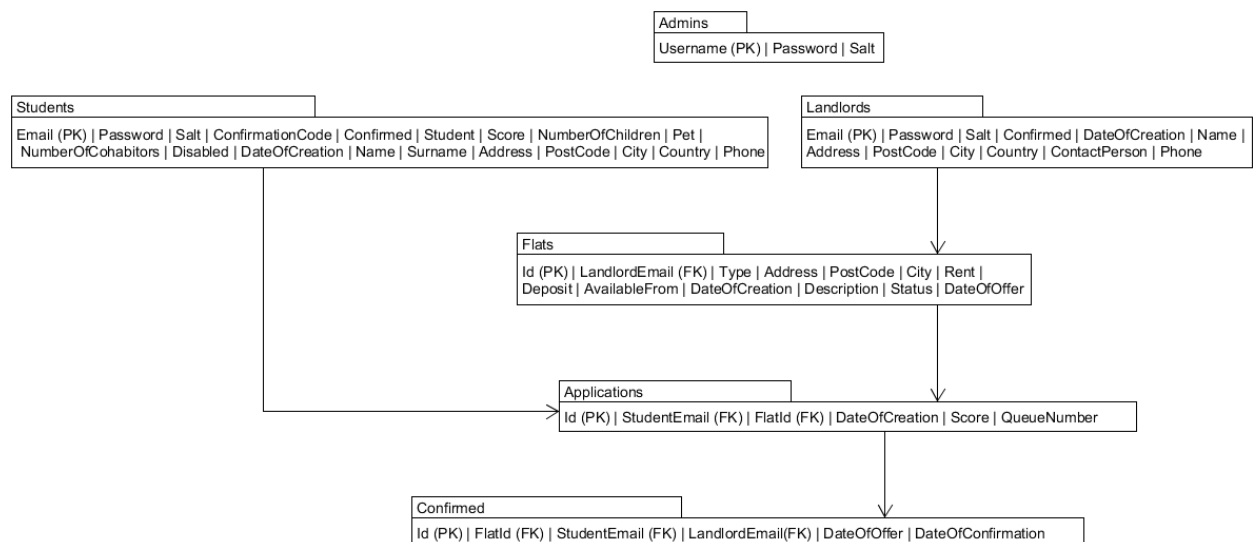
5 Design and implementation

The final two steps in our project were to design and implement all the requirements mentioned above. As these two steps are really closely connected, we have developed this title in 8 parts, mostly matching our idea about the architecture layers, starting with relational model, data layer, DAL, Business layer, Client presentation layer, Client layer and finish with describing more in depth the concurrency (isolation) and testing.

All of the parts are intertwined between design and implementation, starting with a little knowledge about the theory, discussion and then the final implementation.

5.1 Relational model & schema

Relational schema represents our database components and how they are linked. All of the tables have their own unique primary key, which, in most cases is the Id. Only the Student, Landlord and Admin have differently named primary keys (Username and Email). This was done because two people cannot share the same email, and admins must create differing usernames.



Picture 3. Relational model of our system

5.2 Data layer

Data layer is our data source. It holds Information about Students, Landlords, Flats, Applications and Contracts. In many cases this information is highly sensitive such as user passwords, user personal information, and information when flats are empty, etc.

5.2.1 Secured passwords

In order to make sure that the passwords are safely stored, we avoided manipulation with them as much as possible. In our system we are using hashing algorithm SHA256. One of the drawbacks of this hashing algorithm is the fact that we cannot choose number of iterations - how many times should be the password hashed.

5.2.2 Stored procedures

A stored procedure is a set of SQL statements with an assigned name that is stored in the database in compiled form so that it can be shared by a number of programs. The biggest advantages of using stored procedures are:

1. Reuse of the execution plan - Execution plan is the best possible way for the query to retrieve the data. Stored procedures are capable of caching and reusing the execution plan instead of generating a new one every time they are called, which increases performance.
2. Reducing the network traffic - Stored procedures can be incredibly long. In order to execute such procedure, we only need to transmit the name of the procedure and its parameters. Not using stored procedures and sending all the lines of SQL statements will, on a larger scale, definitely impact network traffic.
3. Code reusability and maintainability - Several applications can reuse stored procedures on SQL Server. If the logic of the stored procedure needs to be changed, it only needs to be changed at one place. If you use plain SQL statements, we would have to change it across the whole application.
4. Security - Prevention of SQL Injection attacks.

Picture 4. An example of a stored procedure

```

Use EFlats_v2
Go
Create procedure spInsertNewFlat
(
    @LandlordEmail char(50),
    @Type char(50),
    @Address char(50),
    @PostCode char(50),
    @City char(50),
    @Rent float,
    @Deposit float,
    @AvailableFrom char(50),
    @Description char(500),
    @MessageOutput char(100) output
)
With Encryption
As
Begin

    declare @UserCount int
    /*check if landlord email exists*/
    select @UserCount = Count(Email) from Landlords where Email = @LandlordEmail
    if @UserCount > 0
    begin
        Insert into dbo.Flats
        ([LandlordEmail], [Type], [Address], [PostCode], [City],
        [Rent], [Deposit], [AvailableFrom], [DateOfCreation], [Description])
        values
        (@LandlordEmail, @Type, @Address, @PostCode, @City,
        @Rent, @Deposit, @AvailableFrom, getDate(), @Description)
        select @MessageOutput = 'Successfully added.'
    end
    else
        select @MessageOutput = 'Unable to add flat due to nonexistent landlord email.'
End
  
```

5.2.3 Performance

In order to increase performance, by reducing the number of calls to the database by server, we put some logic on the data layer. Very straight forward example of this would be adding a new flat into our database. To successfully add the flat to the database, we need to ensure, that landlord is already registered in our system and so exists in the Landlord table. Instead of calling the database twice (first time to read, second time to write) we handle this directly in our stored procedure. The foreign key constraint, would of course, not allow it in the first place, but following these steps we can identify the error and return it as output parameter of the stored procedure to the server.

5.3 Data access layer (DAL)

Data access layer works as a link between the business entities in the application and the data storage layer. This layer encapsulates the code that is used to connect to the database and performs CRUD operations. It is used to create and populate business entities with data from the database and for updating and storing business entities in the database.

5.3.1 Connection

One of the things that are handled in the DAL is connection. For this, we use SqlConnection class. During the implementation we began with using one static object of SqlConnection located in our DbConnection class. This was proven to be ineffective and caused a lot of fails when calls were requested concurrently. Instead of reusing one connection, we have decided to create a connection object every time database access is requested. In order to ensure that these connection objects will be closed successfully we have implemented the using block, which disposes the connection object automatically.

```
public string AddFlat(MdlFlat mdlFlatObj)
{
    using (SqlConnection conn = new SqlConnection(DbConnection.connectionString))
    {
        conn.Open();
        return GetOutput(conn, mdlFlatObj);
    }
}
```

Picture 5. Connection to the database

5.3.2 CRUD operations

For security purposes, every time we are accessing the database, we are using SQL Command of type procedure.

In some cases, when we need to fetch data from the database, we also use DataAdapter object rather than DataReader. We see a huge benefit using an adapter. First of all, it reduces the number of calls to the database (loading data at once and not row by row). Secondly, it has a very good connection management. When calling method Fill, the adapter opens a connection, executes command, reads the data, populates the DataSet and closes the connection. When connection is closed and dataset is populated, we are given the possibility to maintain the table in the offline mode and then update the results.

Very useful implementation of this is, for example, when administrator maintains table Students. Once the table is loaded, connection is closed and he is able to add, update or delete changes in the offline mode. He can either discard the changes by clicking refresh button or save the changes clicking commit button. Data adapter then first handles all deleted rows, then updated rows and then inserted rows returning number of modified rows.

```
//execute delete, update and insert
int deleted = dataAdapter.Update(ds.Tables[0].Select(null, null, DataRowState.Deleted));
int updated = dataAdapter.Update(ds.Tables[0].Select(null, null, DataRowState.ModifiedCurrent));
int added = dataAdapter.Update(ds.Tables[0].Select(null, null, DataRowState.Added));

//if nothing was updated return false
if (deleted == 0 && updated == 0 && added == 0)
    return false;
```

Picture 6. Data adapter example

5.3.3 Transaction and concurrency management

Transaction management is also a part of this layer. Reason why transactions are implemented in this layer is mainly to avoid concurrency issues related to the database. In here we focus mainly on transaction consistency and isolation property.

We are using database transactions because we want to create series of data manipulation statements (insert/update/delete) execute as a whole. When manipulating with database, there is a number of issues we can come across with.

- Dirty Read: One transaction reads changed data of another transaction but that data is still not committed. You may take decision/action based on that data. A problem will arise when data is rolled-back later. If rollback happens then your decision/action will be wrong and it produces a bug in your application.
- Lost update: Two transactions are trying to update same resources, while one overrides the other, and the update is lost.
- Non Repeatable Read: A transaction reads the same data from same table multiple times. A problem will arise when for each read, data is different.
- Phantom Read: Suppose a transaction will read a table first and it finds 100 rows. A problem will arise when the same transaction goes for another read and it finds 101 rows. The extra row is called a phantom row.

In our project we had to determine in what way is the transaction going to manipulate the data and to set the transactions isolation level appropriately. In order to achieve maximum performance we had to set as low isolation level as needed - higher isolation level, slower performance.

1. Serializable: Highest level of isolation. It locks data exclusively when read and write occurs. It acquires range locks so that phantom rows are not created.
2. Repeatable Read: Second highest level of isolation. Same as serializable level of isolation, except it does not acquire range locks so phantom rows may be created.
3. Read Committed: It allow shared locks and read only committed data. That means never read changed data that are in the middle of any transaction.
4. Read Un-Committed: It is the lowest level of Isolation. It allows dirty read.

For example when we try to get users wish list data from the database, we read only once, so Read Committed isolation level is appropriate. On the other hand when we update flats status, we call a procedure consisting of 4 individual update statements which could lead to an issue such as lost update so we decide to use isolation level Serializable.

```
var option = new TransactionOptions();  
option.IsolationLevel = System.Transactions.IsolationLevel.Serializable;
```

Picture 7. Isolation property

5.4 Business layer

Business layer can be divided into three main components:

- WCF - handles WCF deployment
- Server functionality - handles and encapsulates all of business domains and logic
- Service Host

5.4.1 WCF

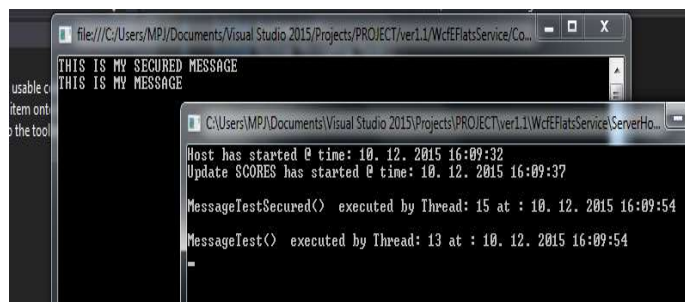
Windows Communication Foundation (WCF) is a framework for building service-oriented applications. Our WCF Service consists of two different Service Contracts. One of them, IWcfEFlatsService, is designed to be accessible by public whereas IWcfEFlatsServiceAdmin is only accessible by administrator on the same Local Area Network as the server. For both services we have configured different endpoints.

We also set different protection levels of our operation contracts. We configured higher protection level when passing data (datasets or collections) from the database and lower protection level when we are writing or updating the database.

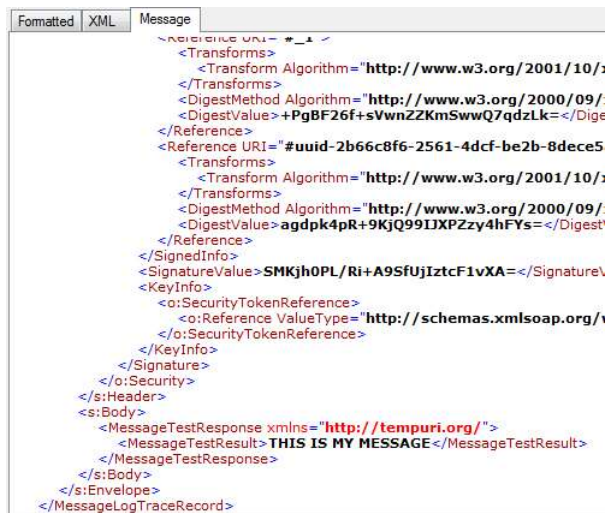
```
[OperationContract(ProtectionLevel = ProtectionLevel.None)]  
bool ConfirmTenants(int flatId, string studentEmail);  
  
[OperationContract(ProtectionLevel = ProtectionLevel.EncryptAndSign)]  
DataSet GetConfirmedFlats(string landlordEmail);
```

Picture 8. Protection levels of operation contracts

In order to see that messages are really encrypted we configured Microsoft Service Trace Viewer.



Picture 9. Message encryption check

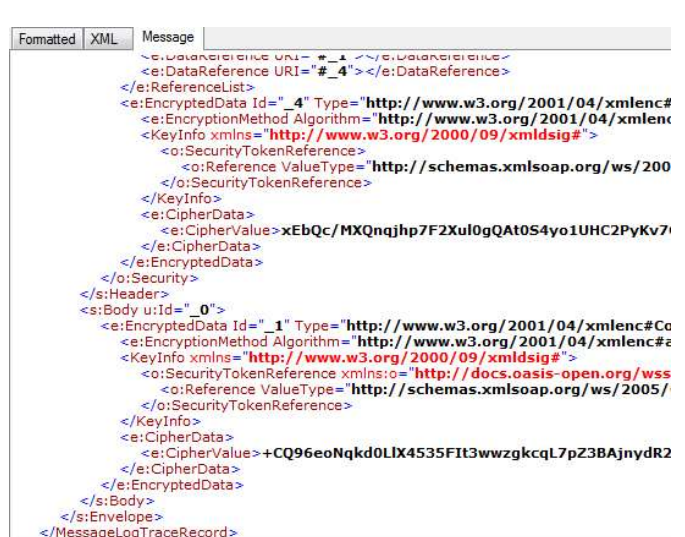


```

<?xml version='1.0' encoding='utf-8'>
<Transforms>
  <Transform Algorithm="http://www.w3.org/2001/10/...
</Transforms>
<DigestMethod Algorithm="http://www.w3.org/2000/09/...
<DigestValue>+PgBF26f+sVwnZZKmSwwQ7qdzLk=</Digest
</Reference>
<Reference URI="#uuid-2b66c8f6-2561-4dcf-be2b-8dece5
</Reference>
<Transform Algorithm="http://www.w3.org/2001/10/...
</Transforms>
<DigestMethod Algorithm="http://www.w3.org/2000/09/...
<DigestValue>agdpk4pR+9KjQ99IJXPZzy4hFYs=</Digest
</Reference>
</SignedInfo>
<SignatureValue>SMKjh0PL/Ri+A9SfUjIztcF1vXA=</SignatureV
<KeyInfo>
  <o:SecurityTokenReference>
    <o:Reference ValueType="http://schemas.xmlsoap.org/1
    </o:SecurityTokenReference>
  </KeyInfo>
</Signature>
</o:Security>
</s:Header>
<s:Body>
  <MessageTestResponse xmlns="http://tempuri.org/">
    <MessageTestResult>THIS IS MY MESSAGE</MessageTestResult>
  </MessageTestResponse>
</s:Body>
</s:Envelope>
</MessageLogTraceRecord>

```

Picture 11. Message not encrypted



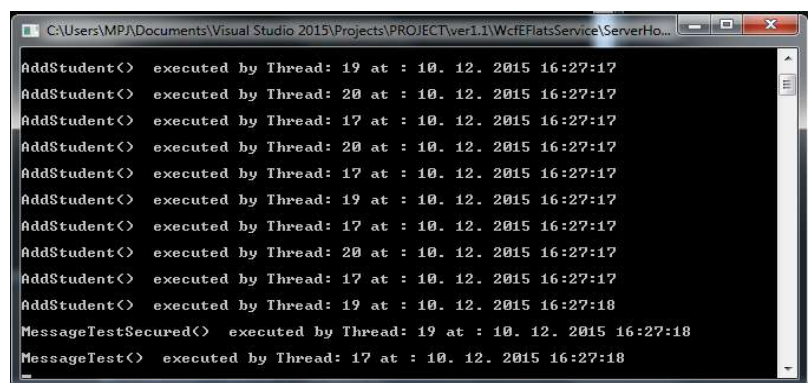
```

<?xml version='1.0' encoding='utf-8'>
<e:DataReference URI="#_4"></e:DataReference>
</e:ReferenceList>
<e:EncryptedData Id="_4" Type="http://www.w3.org/2001/04/xmlenc#
<e:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#
<KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
  <o:SecurityTokenReference>
    <o:Reference ValueType="http://schemas.xmlsoap.org/ws/200
    </o:SecurityTokenReference>
  </KeyInfo>
  <e:CipherData>
    <e:CipherValue>xEbQc/MXQnqjhp7F2Xul0gQAt0S4yo1UHC2PyKv7
  </e:CipherData>
  </e:EncryptedData>
</o:Security>
</s:Header>
<s:Body u:Id="_0">
  <e:EncryptedData Id="_1" Type="http://www.w3.org/2001/04/xmlenc#Cc
  <e:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#e
  <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
    <o:SecurityTokenReference xmlns:o="http://docs.oasis-open.org/wss
    <o:Reference ValueType="http://schemas.xmlsoap.org/ws/2005/
    </o:SecurityTokenReference>
  </KeyInfo>
  <e:CipherData>
    <e:CipherValue>+CQ96eoNqkd0LIX4535Fit3wwzgkcqL7p23BAjnydR2
  </e:CipherData>
  </e:EncryptedData>
</s:Body>
</s:Envelope>
</MessageLogTraceRecord>

```

Picture 10. Encrypted message

WCF Class is implementing both Service interfaces. Service behavior is configured to have Instance Context Mode – Single, which ensures that all Clients will be calling the same instance of WCF, and Concurrency Mode – Multiple, which allows multiple thread accessing the same Operation contract at the same time.



```

C:\Users\MPA\Documents\Visual Studio 2015\Projects\PROJECT\ver1.1\WcfEPlatsService\ServerHo...
AddStudent() executed by Thread: 19 at : 10. 12. 2015 16:27:17
AddStudent() executed by Thread: 20 at : 10. 12. 2015 16:27:17
AddStudent() executed by Thread: 17 at : 10. 12. 2015 16:27:17
AddStudent() executed by Thread: 20 at : 10. 12. 2015 16:27:17
AddStudent() executed by Thread: 17 at : 10. 12. 2015 16:27:17
AddStudent() executed by Thread: 19 at : 10. 12. 2015 16:27:17
AddStudent() executed by Thread: 17 at : 10. 12. 2015 16:27:17
AddStudent() executed by Thread: 20 at : 10. 12. 2015 16:27:17
AddStudent() executed by Thread: 17 at : 10. 12. 2015 16:27:17
AddStudent() executed by Thread: 19 at : 10. 12. 2015 16:27:18
MessageTestSecured() executed by Thread: 19 at : 10. 12. 2015 16:27:18
MessageTest() executed by Thread: 17 at : 10. 12. 2015 16:27:18

```

Picture 12. Message on hosts Console application to give an overview to the administrator of what and when is being called

5.4.2 Server functionality

This layer is used for logical operation, encapsulation and connection between the DAL and Operation Contracts called at WCF. It consists of a controller and model.

5.4.2.1 Model

In this layer we implemented model classes. For example we encapsulate all students' attributes into one student object. This object is then either passed to DAL or returned to WCF. We also specified which attributes we want to serialize, its name and order. Doing this, client is able to manipulate with these model objects. Since client only has to view data received from WCF, it would be reasonable to set those objects into the read-only state.

5.4.2.2 Controller

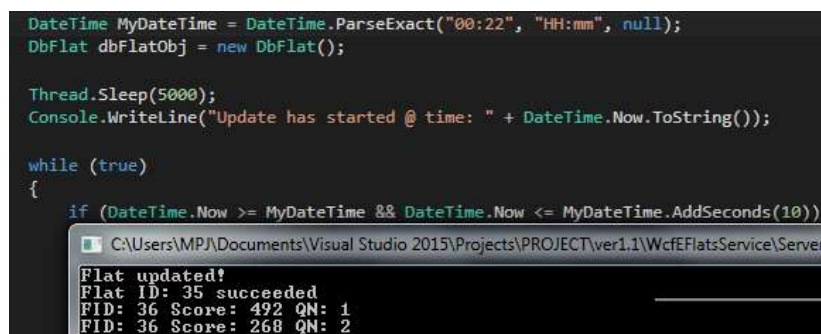
In majority of cases, controller is used to pass data from WCF to DAL. It also handles logic, such as Calculating profile scores, Updating queue numbers, hashing passwords, etc.

5.4.2.3 Performance

On business layer we had to deal with number of performance issues. One of the biggest challenges we were facing was the queue algorithm. The queue algorithm is supposed to calculate student's profile value on the "queue based" attributes he/she has (such as number of children, number of cohabiters, etc.) and store this value as (profile) "score". Then update the value of all his applications. Application value, or application score, is the sum of profile score and number of hours passed since the date student applied. Keeping the date updated was another issue since it increases the value of the application by one score point every hour. Comparing all applicants' scores, we had to assign a queue number to the student, which would indicate how many people are obliged to get the apartment before him.

Such updates could be a very big performance threat. Update, in which one student edits his profile, by changing one of the queue based attributes, could very easily scale into large proportions. For example if this particular student applied for 100 apartments, every apartment having 99 other students in the line, we would end updating queue number of 10 000 students.

We have resolved the issue by breaking this algorithm down into parts. First part happens when student edits his/hers profile - profile score is automatically updated and stored into the database. When the student applies for an apartment, only queue numbers of applicants who already applied for that specific apartment are being recalculated. To connect these two things together we have implemented a scheduled update. We have configured our server to once a day automatically run update of queue numbers of all applicants. This could happen early in the morning, when traffic is less dense. This way the student can progress in the queue by changing his profile and we get rid of "holes" in the queue caused by removed applications.



```

DateTime MyDateTime = DateTime.ParseExact("00:22", "HH:mm", null);
DbFlat dbFlatObj = new DbFlat();

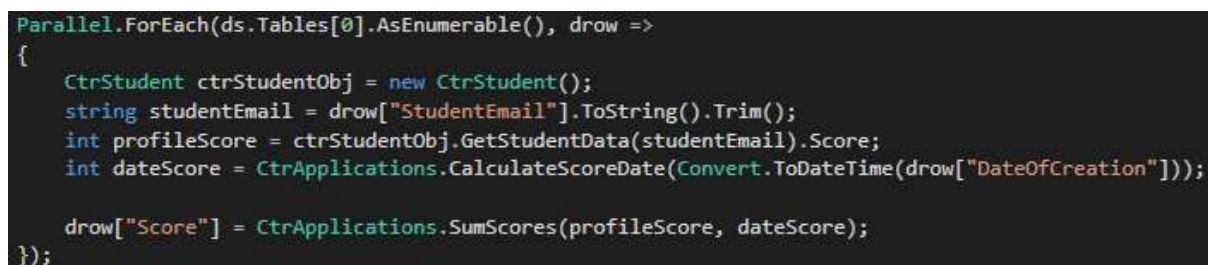
Thread.Sleep(5000);
Console.WriteLine("Update has started @ time: " + DateTime.Now.ToString());

while (true)
{
    if (DateTime.Now >= MyDateTime && DateTime.Now <= MyDateTime.AddSeconds(10))
    {
        Flat updated?
        Flat ID: 35 succeeded
        FID: 36 Score: 492 QN: 1
        FID: 36 Score: 268 QN: 2
    }
}

```

Picture 13. Scheduled update of the system

In order to increase performance we have implemented a parallel foreach loop when recalculating application scores. Since the resources are not shared we haven't come across any concurrency issues. We witnessed significant improvement in performance.



```

Parallel.ForEach(ds.Tables[0].AsEnumerable(), row =>
{
    CtrStudent ctrStudentObj = new CtrStudent();
    string studentEmail = row["StudentEmail"].ToString().Trim();
    int profileScore = ctrStudentObj.GetStudentData(studentEmail).Score;
    int dateScore = CtrApplications.CalculateScoreDate(Convert.ToDateTime(row["DateOfCreation"]));

    row["Score"] = CtrApplications.SumScores(profileScore, dateScore);
});

```

Picture 14. Calculating the score using a parallel foreach loop

We tried to apply parallelism on queue updates as well. We thought that resources would not be shared, if we execute it in parallel for individual flat. Unfortunately we came across deadlock and we were not able to resolve this problem. One of the reasons could be high transactions isolation level.

5.4.3 Host

In our project we are self-hosting WCF Service using a console application. One of the advantages of the self-hosting service is that it is very simple to set up. After configuring the App.config file, all we had to do was to create an instance of service host, specify the type of the service. After that we invoke Open method, which reads the configuration from App.config file and opens communication channel for the clients in order to consume our Service. It also supports all of bindings and transport protocols (while hosting with ISS, only http bindings are supported). This was very useful since we have implemented both wshttp and net.tcp bindings.

Self-hosting is also very easy to debug, which makes it very suitable during the development and demonstration phase. However it is not appropriate for hosting live WCF Services.

```
using (ServiceHost host = new ServiceHost(typeof(WcfEFlatsService.WcfEFlatsService)))
{
    host.Open();
    Console.WriteLine("Host has started @ time: " + DateTime.Now.ToString());

    while (true)
    {
        Console.ReadLine();
        Console.WriteLine();
    }
}
```

Picture 15. Service host

We have implemented two different endpoints in order to fulfill requirements for having different clients, so we used different bindings, contracts and base addresses. First Endpoint is configured to be consumed by a web service. We have used wsHttpBinding in order to achieve security and interoperability. This binding uses SOAP over HTTP. With this kind of binding we can achieve message level security since messages are encrypted by default. We have chosen different protection levels to fulfill operation contract requirements. Second endpoint is to be used by administrator. Due to security reasons, we have decided to have clients on the intranet infrastructure. Because of this we used net.tcp binding - it supports TCP and cannot be accessed over the internet. This binding sends binary encoded SOAP messages within intranet computers.

5.5 Client presentation layer

The presentation layer represents all the utilities that ASP.NETs has in order to create our website. We used simple textboxes for user input, implemented gridview in order to show our database content instead of listview. The advantages of the former are that it shows result in a table and is easily filled through the use of datasets. One of the things that the gridview lacks, however, is the fact that it uses its autogenerated buttonfield, which in terms doesn't allow us to use some of the functionalities a simple button might have - f.x. visibility. We easily fixed this through manually putting template fields with buttons inside in the html code. Throughout the rest we put labels that would change depending on the users' actions and static ones, which would be used to show only text. We also made use of the calendar tool implemented in asp.net and dropdown menus connected with our database that allowed us to switch between different items. And of course we also had validators that ensured that the information the user enters would be in the correct format and that the database would not be filled with null values.

5.6 Client layer

The client layer is the end result of the architecture - it is what the user can see and interact with. In our system we use a web client for the landlords and tenants to use. It contains only the functionality needed for all of the actions to be completed. We also use a windows form client, which would be used only by the admins of the system. In here the user can do basic CRUDs of everything in the system and has access to all of the information. We also have a basic console client, from which people can invoke all the methods that could be used in the web client, and GUI console client was implemented for testing purposes.

5.7 Performance testing

In order to achieve the best possible results when it comes to concurrency we have done several tests. We tested adding (registration) 2000 students to the database concurrently by two clients (4000 in total), with 20 predicted errors for each. There are 5 properties by which we determined most suitable set up - successful, failed due to existing email, unexpected error, expected error and time.

We tried several options. Different isolation levels and different methods - implementing transaction directly on SQL Server, SQL Transactions on DAL and Transaction scope and locks. Our starting point was test without handling concurrency at all and default SQL isolation level - read committed. It was not a surprise that having together 422 unexpected errors. 21 seconds was the time we were comparing other results with.

Test number	0	0
Method	/	/
Isolation level	/	/
Number of calls (for 1 client)	2000	2000
Number of Expected Errors	20	20
Succeeded	998	985
Failed (existing email)	779	776
Unexpected Error	203	219
Expected Error	20	20
Time	21	21

1	1	2	2	3	3	4	4	5	5	6	6	7	7
Locks	Locks	TDB	TDB	STDBL	STDBL	STDBL	STDBL	STDBL	STDBL	STDBL	STDBL	TSDBL	TSDBL
/	/	SS	SS	S	S	S	S	RC	RC	RR	RR	RC	RC
2000	2000	2000	2000	2000	2000	2000	2000	2000	2000	2000	2000	2000	2000
20	20	20	20	20	20	0	0	20	20	20	20	20	20
823	1159	987	994	876	1124	1198	873	1070	946	821	1183	871	1166
1157	821	218	193	1104	855	802	1127	910	1034	1052	681	1109	814
0	0	775	793	0	1	0	0	0	0	107	116	0	0
20	20	20	20	20	20	0	0	20	20	20	20	20	20
20s	21s	23s	24s	25	25	23	22	22	22	57	57	20	20

There are 4 concurrency handling methods we used: Locks, TDB (Transactions on the Database layer), STDBL (SQL Transaction on Database access layer) and TSBDL (Transaction scope on Database access layer). We also used different type of Isolation: SS (Snapshot), S (Serializable), RC (Read committed) and RR (Repeatable read).

Locks have proven to be very safe. The test finished with 0 unexpected errors, which is not surprising since there are no threads executing concurrently. Performance was surprisingly good, due to the fact that only two clients were writing at the same time. On a larger scale performance could decrease significantly.

Adding the transaction logic directly to the Database was our second try. Large number of errors and decreased performance surprised us. One of the reasons was the level of isolation.

Many tests were performed using SqlTransaction class. Different isolation level produced different results. Read Committed isolation gave us most reliable results. Therefore we have decided to use it for the most scenarios, unless higher isolation is needed. The disadvantage of using SqlTransaction class was the performance, which was not better in comparison with test number 0 and 1. The most contributing results when it comes to performance and the number of errors were given using TransactionScope class. We have decided to implement this type of transaction whenever we faced a concurrency issue.

6 Conclusion

All in all we managed to create a user friendly, secure system for flat distribution between tenants and landlords without any performance issues.

The journey was long and it had a lot of obstacles, but thanks to the theoretical knowledge that we gained before the start of the project and then adapting them in this practical task, we achieved what we were looking for.

Yes, sometime the implementation of the code looked like trial and error, but sometime it was also improving the existing code (example: changing locks with transactions; connection to database switched to reader) which results in much better code to use and also easier to change if needed in the future.

The most demanding part was how to actually create the project. We have learned that testing has a really interesting feature – it doesn't only provide the feedback whether our system runs without errors, it also provides a great base for the future changes of the code.

We have also learned that communication is also important part of the creation of the development – if we had asked more questions at the beginning we wouldn't have to change several time the code. Side by side with communication we have team-work. We are especially proud that we managed to develop as a team through the whole process of creation the system.

We are especially proud as we managed to implement most of the ideas for the system that we had set up before we started the project. If we had more time, I am sure that we would implement all of them, including payment web service and manual confirmation of the profile updates.

7 References

[1] Group of authors: Bill Evjen, Scott Hanselman, Devin Rader: "Professional ASP.NET 4 in C# and VB", Wiley (2010)

ISBN: 978-0470502204

[2] Group of authors: Deitel M. Harvey, Deitel J. Paul, Choffnes R. David: "Operating Systems", 3rd edition, Pearson (2003)

ISBN: 978-0131246966

[3] Group of authors: Coulouris George, Dollimore Jean, Kindberg Tim, Blair Gordon: "Distributed Systems: Concepts and Design", 5th edition, Addison Wesley; (27 April 2011)

ISBN: 978-0132143011

[4] Miles, Robert: C# Programming Yellow Book publisher: Department of Computer Science (2015), free version downloaded from <http://www.csharpcourse.com/>

[5] Regan, Patrick: "Networking with Windows 2000 and 2003", 2nd edition, Prentice Hall (21 July 2003)

ISBN-13: 978-0131124622

[6] Sebesta, Robert W.: "Programming the World Wide Web", 8th edition, Pearson (2014)

ISBN: 978-0133775983

[7] Troelsen, Andrew: Pro C# 5.0 and the .NET 4.5 Framework, 6th edition, Apress (2012)

ISBN: 978-0133943030

[8] Material from the course folders on eCampus

[9] Different websites (information retrieved in December 2015):

- <https://www.simple-talk.com/sql/database-administration/sql-injection-how-it-works-and-how-to-thwart-it/>
- <https://crackstation.net/hashing-security.htm>
- <https://zeltser.com/firewalls-for-multitier-applications/>
- [https://msdn.microsoft.com/en-us/library/ms733027\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/ms733027(v=vs.110).aspx)
- <http://stackoverflow.com/questions/4270883/transport-level-vs-message-level-security>
- http://www.w3schools.com/xml/xml_services.asp

Pictures inserted in the main part:

Picture 1. Domain model.....	6
Picture 2. Architecture design of our system	7
Picture 3. Relational model of our system	11
Picture 4. An example of a stored procedure	12
Picture 5. Connection to the database.....	13
Picture 6. Data adapter example.....	14
Picture 7. Isolation property.....	15
Picture 8. Protection levels of operation contracts	15
Picture 9. Message encryption check.....	15
Picture 10. Encrypted message	16
Picture 11. Message not encrypted	16
Picture 12. Message on hosts Console application to give an overview to the administrator of what and when is being called	16
Picture 13. Scheduled update of the system	17
Picture 14. Calculating the score using a parallel foreach loop	17
Picture 15. Service host.....	18

8 Appendix

Company research / Questions and answers

Question 1:

What is the general structure of your online system? Do you require both people renting and looking for a plan to register on your site; which info they need to provide?

BoligPortal: All users register with their e-mail + a password. For all landlords, we also ask for their own private or company address + a phone number for our internal use, according to the guidelines provided by the Danish Consumer Ombudsman. Landlords register their properties for free. Tenants can sign up and or use our app for free. However, in order to create a search profile visible to the landlords and to contact the landlords on the newest ads, they must register a premium (paid) profile, either on a subscription base or as a fixed two months profile.

AKU: Users (Students) register with the register form which is online (check it on AKU site) and then they have to send every year on the mail that they are still students. Housing accommodation send an email to AKU when they have new flats to rent out or when status of flat changes (old leave, new come)

Question 2:

What is the information you need to have regarding the accommodation?

BoligPortal: Most of the information on the ads on our website is mandatory. We also possess the specific address of the property for rent, for our internal use (safety + statistics).

AKU: All of the information required on AKU website is required.

Question 3:

Are you satisfied with search option on your site?

BoligPortal: We know from our customer satisfaction surveys that the search options and general navigation are some of the most valued features by our users. This being said, we are constantly working on improving the way we present data to the users.

AKU: Yes.

Question 4:

Do you keep any archive on the website? If yes, what information do you store in it?

BoligPortal: We keep all profiles and data on the properties in a database. However, if a user contacts us, we can erase the personal data, while still complying with the general rules of data safety and documentation. The information of properties are used for statistics regarding rentals, prices across the country etc.

AKU: Yes, all of tenant information is kept as a record, even if the apartments are not in use any more.

Question 5:

Do you think that the website should be both in Danish and English?

BoligPortal: Yes :) And we could still improve on this.

AKU: Yes.

Question 6:

Is there any problems that you are having with your system and would like to fix (improve)?

BoligPortal: Our main issue is attraction more properties to the site. This is a general problem on the current rental market; the offer of vacant properties no meeting the demand. As stated in the answer to question 3, we work on a continuous improvement / deployment plan to optimize the services delivered to our customers. Due to competition factors, I cannot reveal what we are currently modelling.

AKU: No.

```
<services>
  <service name="WcfEFlatsService.WcfEFlatsService" behaviorConfiguration="mexBehavior">
    <endpoint address="WcfEFlatsService" binding="wsHttpBinding" contract="WcfEFlatsService.IWcfEFlatsService">
    </endpoint>
    <endpoint address="WcfEFlatsService" binding="netTcpBinding" contract="WcfEFlatsService.IWcfEFlatsServiceAdmin">
    </endpoint>
  </service>
```

Picture 16. Endpoint

STUDENTS																		
EMAIL(FK)	PASSWORD	CONFIRMATION_CODE	CONFIRMED	STUDENT	SCORE	NUM_OF_CHILDREN	PET	NUM_OF_COHABITORS	DISABLED	DATE_OF_CREATION	Name	Surname	Address	PostCode	City	Country	Phone	
ak@u.com			1	0	100	1	0	0	0	15/02/2015	John	Smith	...	9000	Aalborg	Denmark	41910789	
ak@u.com			0	0	200	1	0	0	1	24/04/2015	Kate	Brown	
ak@c.com			1	1	100	3	0	0	0	24/09/2014	Jack	Perry	
LANDLORDS																		
EMAIL(FK)	PASSWORD	CONFIRMED	DATE_OF_CREATION	Name	Address	PostCode	City	Country	Contact Person	Phone								
ak@u.com		1	17/02/2015	Henry	...	9000	Aalborg	Denmark	...	4199889456								
ak@u.com		0	22/04/2015	Marlene								
ak@f.com		1	29/09/2014	Hugo								
FLATS																		
ID(FK)	LandlordEmail(FK)	Type	Address	PostCode	City	Rent	Deposit	Available From	Date Of Creation	Description	Status	Date Of Offer						
1	ak@f.com	house	...	9000	Aalborg						
2	ak@f.com	flat						
3	ak@f.com	flat						
APPLICATIONS																		
ID(FK)	StudentEmail(FK)	FlatId(FK)	Date_Of_Creation	Score	Queue_Number													
1	ak@u.com	1	15/11/2015	100	1													
2	ak@u.com	2	15/11/2015	100	2													
3	ak@c.com	2	15/11/2015	200	1													
CONFIRMED																		
ID(FK)	StudentEmail(FK)	FlatId(FK)	LandlordEmail(FK)	Date_Of_Creation	Date_Of_Offer													

Picture 17. Relational Schema

```
Use EFlats_v2
Go
Create procedure spInsertNewFlat
(
    @LandlordEmail char(50),
    @Type char(50),
    @Address char(50),
    @PostCode char(50),
    @City char(50),
    @Rent float,
    @Deposit float,
    @AvailableFrom char(50),
    @Description char(500),

    @MessageOutput char(100) output
)
With Encryption
As
Begin

    declare @UserCount int
    /*check if landlord email exists*/
    select @UserCount = Count(Email) from Landlords where Email = @LandlordEmail
    if @UserCount > 0
        begin
            Insert into dbo.Flats
                ([LandlordEmail], [Type], [Address], [PostCode], [City],
                [Rent], [Deposit], [AvailableFrom], [DateOfCreation], [Description])
            values
                (@LandlordEmail, @Type, @Address, @PostCode, @City,
                @Rent, @Deposit, @AvailableFrom, getDate(), @Description)
            select @MessageOutput = 'Successfully added.'
        end
    else
        select @MessageOutput = 'Unable to add flat due to nonexisting landlord email.'
    End
End
```

Picture 18. SQL procedure