Movie WishList App

Krausz a Belak

Statement of originality

We hereby certify that we are the sole authors of this thesis and that no part of this thesis has been published or submitted for publication. We certify that, to the best of our knowledge, our thesis does not infringe upon anyone’s copyright nor violate any proprietary rights and that any ideas,techniques, quotations, or any other material from the work of other people included in our thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices.

We declare that this is a true copy of our thesis, including any final revisions, as approved by our thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

[**Introduction**](#_b4og4zox9pca) **5**

[**What is Movie WishlistApp ?**](#_30h7osj9bn3w) **5**

[**Problem Analysis**](#_dosrrddcf7dq) **5**

[**Problem Statement**](#_hjh1o95u65f9) **5**

[**Risk Analysis**](#_drpdt1rnurrr) **6**

[Risk management plan](#_wdvwu973ise6) 6

[Risk analysis document (Risk Log)](#_yrm5id4w7bp5) 8

[**Technologies**](#_20shqbcw0i2n) **9**

[**Platform**](#_4l968dnm8jun) **9**

[**IDE**](#_iipaxx7bx5s1) **9**

[**Tools**](#_qqgysdnaxba5) **9**

[Trello](#_xo9l01860e2m) 9

[Discord](#_k3xlxmqnlbod) 10

[**Front-End**](#_3jdlojvubgh0) **11**

[HTML](#_wyo27nquq9a9) 11

[Bootstrap](#_d9nmh3dm0vak) 11

[**Back-End**](#_kduwf7svteeb) **12**

[ASP. NET](#_kqpt7ecn0d33) 12

[Microsoft SQL Database](#_vdz7odz1aoxz) 12

[MoviesDB](#_cgyzwocao3fs) 13

[MovieAppDB](#_mmekj0t64k5q) 13

[Stored procedures](#_709vo6bk4z4s) 14

[Populating the MoviesDB database](#_5f0bxo3n2fye) 15

[Rest API](#_k10wm3volqvf) 16

[**Source Control**](#_wms4kt3yqfuv) **16**

[Github](#_rdtwhhtpc7jy) 16

[**Project Architecture**](#_2dm6rj2ac6wa) **16**

[**Server**](#_1nrbr933041y) **18**

[**Client**](#_xqrf8tcy0rsr) **19**

[Client-side VS Server-side](#_cz0vldktur5d) 19

[Rest Sharp](#_r4qhe0k3m3q9) 20

[Models](#_ipr1ptktma8) 21

[**Testing**](#_mdxk75s2a3nx) **22**

[**Unit tests**](#_9b803nqylw2x) **22**

[**Selenium**](#_6r2t373yrd94) **23**

[**Security**](#_qkn0u0oo8950) **24**

[**The need of security**](#_fdbs4h7q5dn3) **24**

[**Our precautions**](#_qf4mzq4y6o0s) **25**

[**Security risks**](#_27pa6rpnrep6) **25**

[**Utilization of our education**](#_oxgk79tlkcaw) **26**

[**Reflections**](#_jj9hjlie1685) **27**

[**Conclusion**](#_bz2p8wdi017t) **29**

[**Bibliography**](#_qplgft583yhb) **30**

[**Appendices**](#_x9i39z6v2on7) **31**

# 

# Introduction

## What is Movie WishlistApp ?

Movie WishList App is an online application that can help users track, manage and keep order of the movies they watched and would like to watch. The application is lightweight with only the basic options mentioned above and is accompanied with a simple, intuitive user interface.

## Problem Analysis

Even though we both had an internship at the same company and therefore were offered to develop some sort of project for the company, we decided to make our own final project that would better suit our needs and criteria.

Our focus for this project was to try to develop an application that would combine all main learning subjects from our two previous semesters at KEA. Those subjects were:

1. Databases

2. Testing

3. Development of Large systems

4. Software integration

Considering these four main criteria we came up with a straightforward idea that would incorporate most of them. Of course many ideas would fit these requirements so please consider this project as a showcase of our skills and not a “trying to fill a hole on a market” project.

## Problem Statement

Can we create simple, yet useful application for tracking watched movies and provide users with basic options to manage their lists?

To come up with an answer for our question, we decided to do a research and register for some of the wishlist applications which are available in many different forms on the internet.

We were looking for the following:

* Which features were the same/similar in the most used websites
* Which features were missing for us as users
* How intuitive was the navigation
* How many mouse clicks were needed to complete an action/goal

Additionally to development, we also had to put into consideration other factors when it comes to the project as a whole, like resources at hand, our knowledge and skills, and the timespan of the project.

* Even though we had sufficient time for the project, the holiday time took away from us some precious time due to us procrastinating. The final application could be more complex with additional features added to it.
* Furthermore, since we wanted to put our skills and knowledge from previous studies at KEA to use, we were focusing on specific tasks and put others on the sidetrack for possible future development.

## Risk Analysis

### Risk management plan

We can define risk as a possibility of an event or condition occurring that might have a negative effect on one’s objectives with undesirable consequences. To minimize the effect of risks on our project, we can take advantage of a risk management plan.

Software risk management objectives are to identify, address, and eliminate software risk items before they become either threats to successful software operation or major sources of expensive software rework. With planning, we can manage how these objectives will be performed, recorded and monitored during the development.

Each risk that we identify requires the definition of its :

* Probability (a chance of the risk happening)
* Impact (consequences if it happens)

We have decided to use the following approach to define our risks:

**Probability**

* Rare [1]

Almost no probability of happening

* Unlikely [2]

Very low probability of happening

* Possible [3]

Medium probability of happening

* Likely [4]

HIgh probability of happening

* Almost certain [5]

Very high probability of happening

**Impact**

* Minor [1]

Risks which have almost no impact on the development process

* Low [2]

Risks which have very small impact on the development process

* Medium [3]

Risks which have considerable impact on the development process

* High [4]

Risks which have major impact on the development process

* Catastrophic [5]

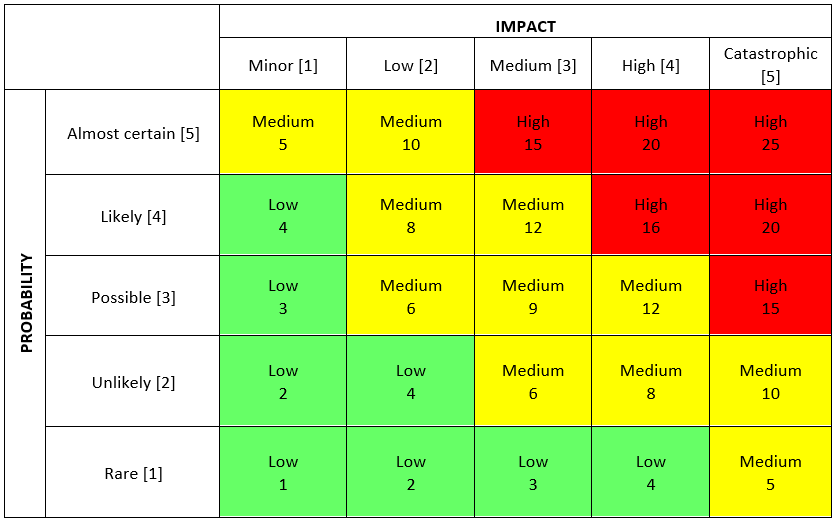
Risks which have disastrous impact on the development process

To identify risk areas in our project, we created the following risk matrix. We calculated the risk level (risk factor) of our potential risks as:

***RISK FACTOR = PROBABILITY OF RISK OCCURRING x IMPACT IF IT DID HAPPEN***

This way we can evaluate, manage and control the risks when they appear during our development process.

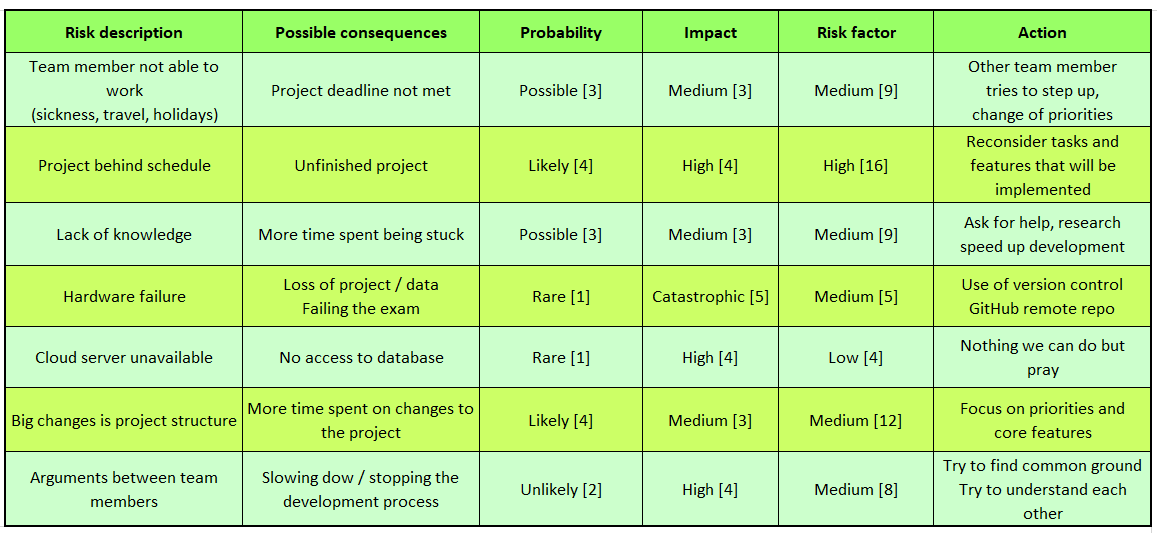
Risk matrix



### Risk analysis document (Risk Log)

To identify possible risks which might appear during our development process, we had created a Risk Log. This spreadsheet helps us to record, analyze and come up with a necessary action to be taken to eliminate or minimize the con sequences of risks.

Risk analysis spreadsheet.



# Technologies

In this section we will give a brief introduction to what software and processes we have decided to use. Most of the technologies are used because we became familiar using them during our studies and thus have previous experience working with them.

Another reason for our decisions was taking into consideration the current newer technologies which are (and will be) supported and developed.

## Platform

We were very straight forward with deciding on which platform our application should be run. For the scope of this project and allocated time frame, we anticipated users would use it on their PC browsers. There is a possibility to make changes and adaptations so that it can be used on mobile devices as well. This would, however, take a considerable time to achieve since none of us have previous experience with ASP.NET MVC mobile features.

Mobile version is therefore possible, but beyond the scope of this bachelor project.

## IDE

Our IDE of choice is Microsoft Visual Studio 2019. The main reason is that we have both used it extensively before and thus have the most experience with. Visual Studio is a powerful IDE with support for testing and team collaboration which was very beneficial for us while working on our application.

Visual Studio and Microsoft Azure cloud services, both part of Microsoft technology stack, provide us with the possibility to develop our app locally and when it is done, publish it to Azure to try it in the cloud with ease.

## Tools

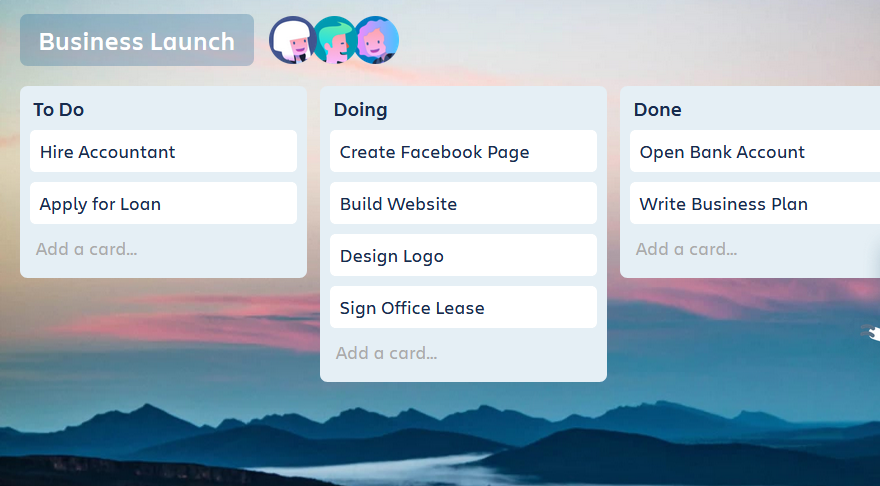
### Trello

To manage our working tasks and to keep the track of our project we used Trello.com. We were able to add tasks “on the go” throughout the development process, discard the ones that we decided were not needed, were too complicated or were out of the scope of our project given its time frame.

Trello is using boards, cards and lists to organize projects in a flexible way. You can chat with team members and together add tasks on different board lists (To-do, Doing, Done, Tested, Done Done, etc.).

It is a useful tool which we were advised to use even during our course projects to manage our development.

Basic interface of Trello.



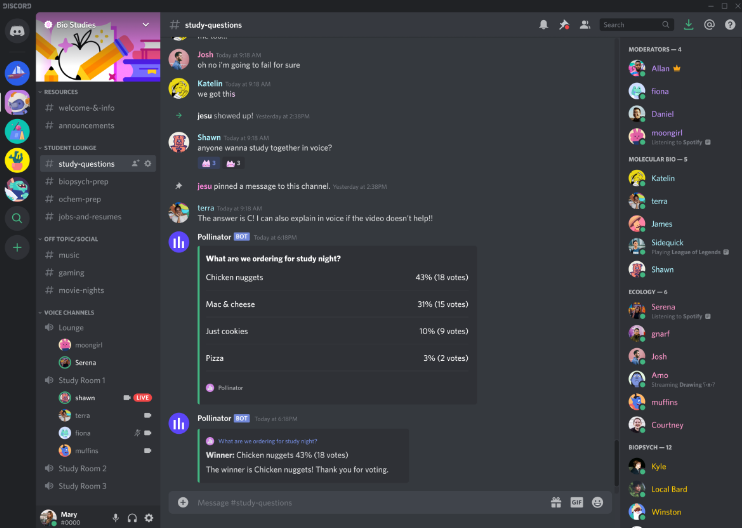
### 

### Discord

Even though Trello provides enough features for a team collaboration, it can not replace spoken dialog when a problem emerges or when an important decision has to be made.

In that case we used a Discord application where we had our own server with a private room. Discord is used widely in many different communities and we are both using it in our everyday lives as well as used to communicate with class members during group assignments.

A discord server with a room and a chat open



## Front-End

Since we are Software Developers our primary focus is going to be on the server side. Still there was a need for basic user interface, so we adopted what we knew outside of KEA classes.

### HTML

Html was our most used language when creating pages for users to interact with.

### Bootstrap

To ease up the process of creating custom css styles we made use of a simple and well established library - Bootstrap provides style classes for all our elements to make the overall look responsive.

Bootstrap is an HTML, CSS, and JavaScript framework that creates consistent-looking, responsive websites. It is automatically installed with MVC 5 applications and immediately ready to use within the default layout.

## 

## Back-End

This is the core of our project. Our preferred language of choice was C#, as we are both well established with it, from our previous education and work experiences. Most of the code is written in C# except for some logic that was put into stored procedures for the sole purpose of enlarging the project, making use of SQL statements and practicing what we learned in the database course.

### ASP. NET

An obvious choice, given our chosen language and nature of the task given.

The only discussion here was between .net and .net core. As the later framework provides better support for development options and is also set to be the one that microsoft will move forward in the future. However, due to the fact that we are both inexperienced with this newer framework, we decided to stick with what we know best. This was to avoid time consuming stumbles and small mistakes, when trying to figure out how some things work in this newer version.

### Microsoft SQL Database

We decided to go with SQL type database (Microsoft SQL Server 2019) for our storage solution. The reasoning behind it is that, even though this application is on a smaller scale, we still need to interpred entities and connection between them. Any relational database like MySQL or PostgreSQL would be suitable for us, but since we have most experience with SQL Server, it was our database of choice.

Additionally Microsoft offers to host SQL Server databases on Azure cloud services which was ideal for our project, therefore we decided to take advantage of this option and use our student account to host our database there. To manage our databases we used Microsoft SQL Server Management Studio (SMSS) because it is a powerful tool and we both have previous experience working with it from our previous studies.

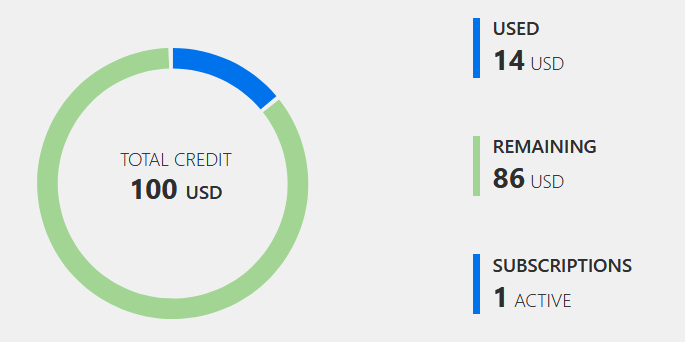
When first deciding about what application to make for the final exam, we considered consuming an API from an already established online movie databases like IMDb or TMDb but decided against it since for us to use their APIs we would have to spend money as a subscription or a “donation”. Therefore we settled for our own “movie database” which we also hosted on Azure cloud services. This decision would also allow us to create a server application which we could develop by ourselves and tailor it to our needs.

We ended up with two SQL Server databases hosted on a student account (with a balance of 100$), which caused us a complication which we did not anticipate.

When creating an SQL Database on Azure cloud, the default compute + storage setting is set to “General Purpose” with 32GB of storage and 2 vCores. This setting, unbeknownst to us, drained our account out of available resources after one week. Since we did not anticipate this happening, we had to recreate and populate the databases again. Thankfully we both had an student account available, so the new databases were created on the other account, this time set up to “Basic” configuration with only 2GB of storage.

Lastly, there is an issue with firewall settings on Azure SQL server. Everytime we wanted to make a connection from a new IP address, we would have to create a new firewall rule and add that IP address to the list. This is a problem that needs to be addressed when thinking about inviting users to test and use our application.

Screenshot of Azure account balance after ~1 month of active subscription



Our two SQL databases are:

* MoviesDB
* MovieAppDB

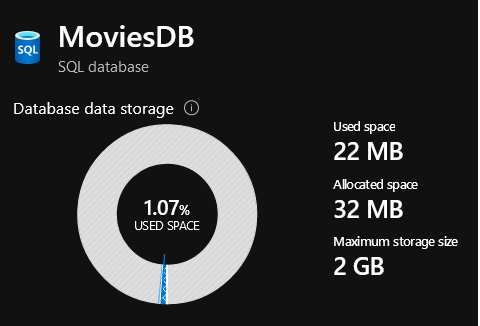
#### MoviesDB

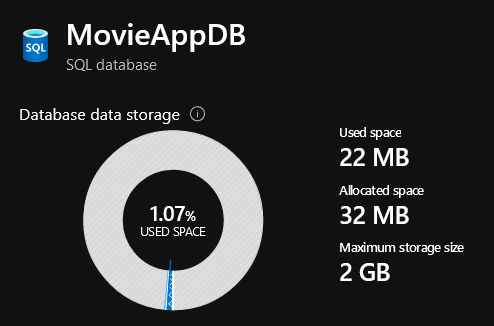
This is a very simple database which consists of only one table - Movie table. The purpose of this database is to simulate a service where we would consume its API to get access to movies and their details (Title, Genre, Director, Duration, etc.). In our case the storage size of 2GB was sufficient enough, since we are storing only a sample of the data for development and testing purposes. If we considered this database for the end user, we would have to vertically scale our database to fit for a larger amount of data.

#### MovieAppDB

This is the database we use for our application to store relevant data about users and their wishlist. It is quite similar to MoviesDB because it has only two tables - User table, Wishlist table. User table contains basic user information like Username, Gender, Age, … The idea behind collecting all this data was a planned feature where we could create simple statistics like which kind of movies specific gender or age group prefers.

* Talk about how wishlist gets populated

Screenshots from Azure SQL database overview



#### Stored procedures

There are several things to take into consideration when deciding if and when to use stored procedure instead of just coding the logic in the application. We took into consideration these main factors:

* **Performance**

We found out that there is no difference in performance. In SQL the query engine optimizes the procedures and reuses them if possible

* **Implementation**

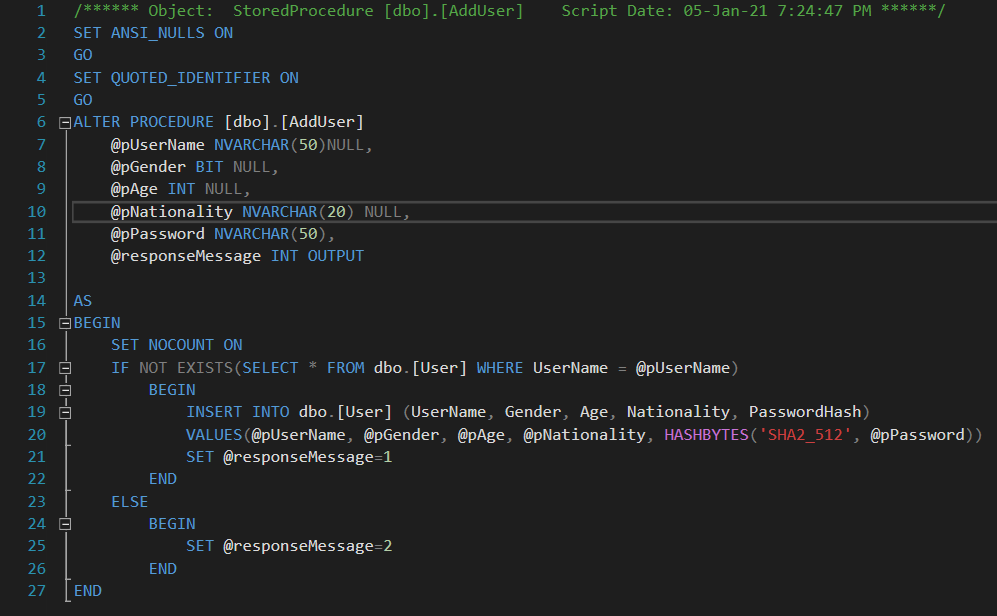
It is faster to write code only, since using stored procedures adds another layer that needs to be maintained and writing them can lead to additional errors.

* **Safety**

We learned that stored procedures are safer due to only needing to grant execution rights on stored procedures to a SQL login while using Entity Framework, full read and write access must be granted.

In conclusion, even though in our case it would be easier to just write code and avoid stored procedures, we thought that we should use them to practice what we learned in our databases course on KEA.

Stored procedure to add user into the database (register user)

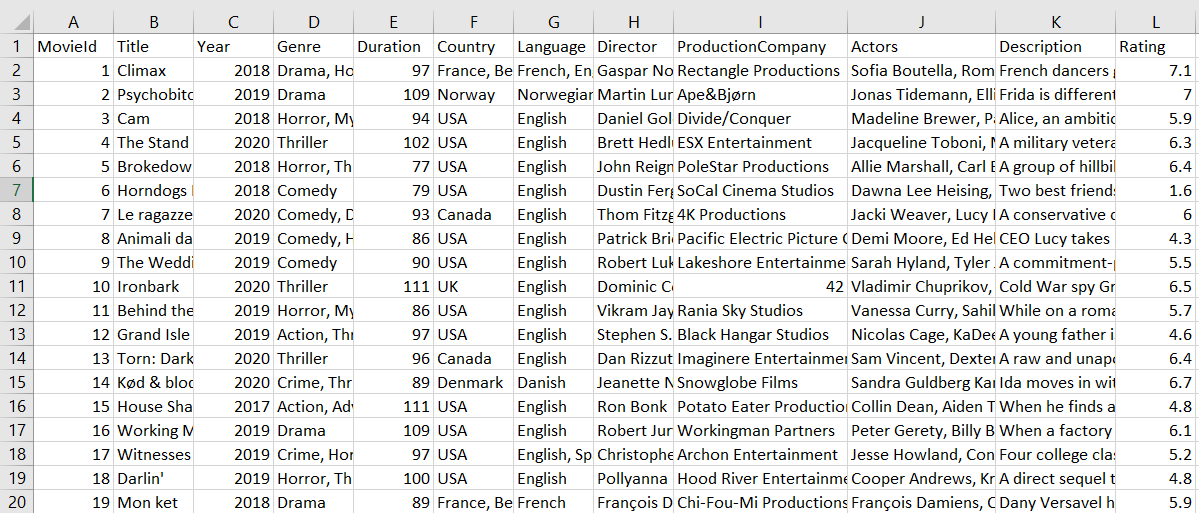


#### 

#### Populating the MoviesDB database

If we wanted to populate the MoviesDB manually, it would take us a tremendous amount of time to do so. Even with only twenty to thirty rows (bare minimum for testing) it would be very ineffective. Therefore we opted for the use of a dataset. The specific dataset we used is “IMDb movies extensive dataset” from kaggle.com. The file we used was a .csv file which we shortened and edited to suit our needs, because we did not need all the data. SSMS offers an easy way to import .csv files which was another reason for us going with the dataset option.

.csv dataset snippet



### 

### 

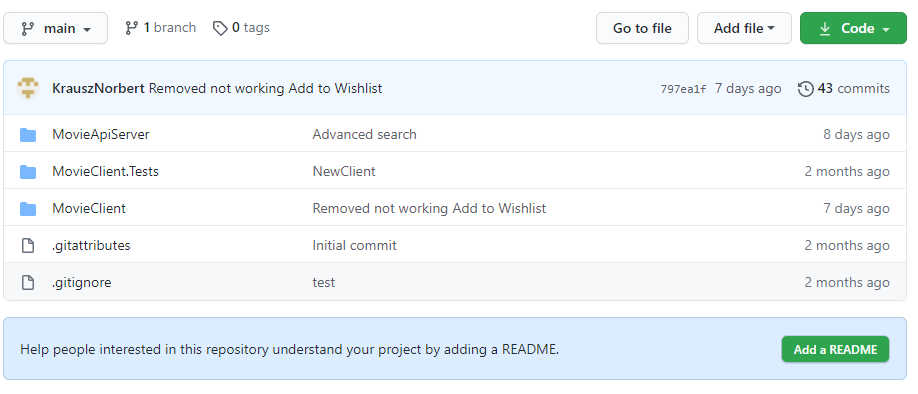
### Rest API

Was used as the main communication tool among parts of the project. Usually by initiating the request with the client and receiving a response with data.

## Source Control

### Github

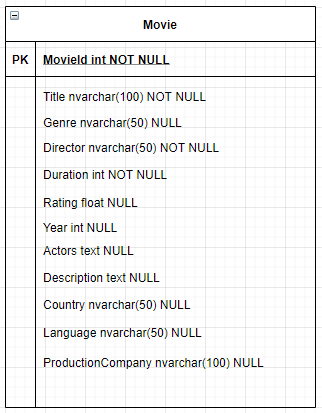
Github is the most popular version control software which allowed us to work independently and collaborate in an effective way. By creating a private repository we were always able to track new changes to the code done by the other person.



# Project Architecture

In this chapter we will try to elaborate on design decisions and explain how the whole application works. Moviewishlist consists of two parts. The server providing data and Client making requests to access this data, movies to be exact. At first we tried to find a free service that would provide us data that we need. After unsuccessful search where only paid providers would show up, we decided to create our own server.

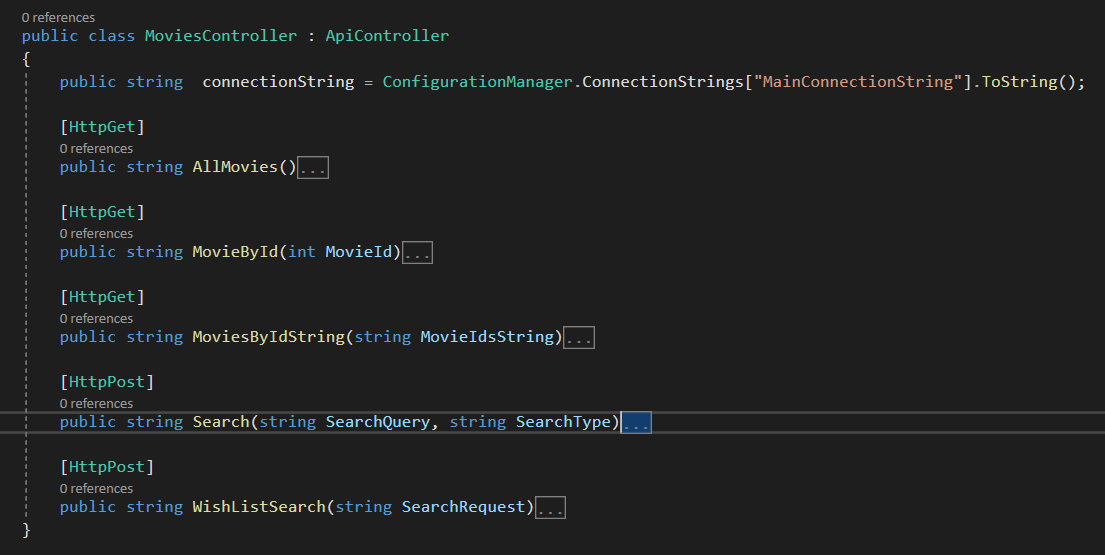
Before we begin this explanation we must first define what a movie is in our system and what properties it holds.

A Movie represented in Server database looks like this:

It was our decision to keep only the Title, Director and Duration as required. We consider this as the bare minimum for most movies and should cover some rare scenarios, for example a movie that is animated therefore does not cast any actors (however voice actors might be considered here), or an old movie with no exact known year of release. Director might be a debatable field as well since there is a possibility for a movie produced by an unknown person. However we considered this a very rare occurrence and therefore designed it as a required field.

## Server

Running ASP.NET application configured to respond to Rest API Requests and return data.



This is the main and only server controller, note how the methods are written in a simple manner this is to simulate the conditions if we didn't have access to our custom made server and instead were using third party services.

All responses are in json format.

**AllMovies** method returns 100 movies from the server database. We are manually limiting the number of movies returned.

**MovieById** accepts Id of requested movie and returns the whole movie with all of its properties

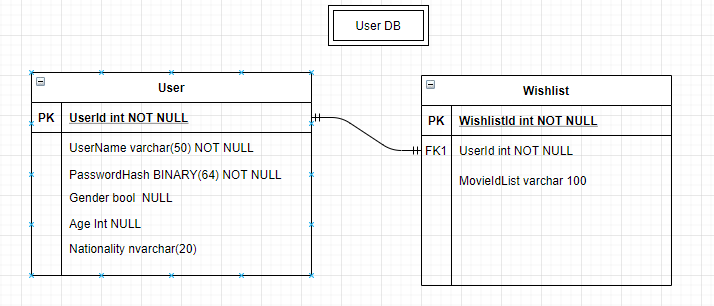
**MoviesByIdString**  is specifically used by the wishlist, it accepts a string of movie ids and returns all movies that match those ids. This method was created by us to increase efficiency, we could use the previous method to loop through a list of Ids and achieve the same result.

**Search**  works by forwarding the search expression and type to the database where the search itself happens. Search is executed by a “like” statement. The type in this context means whether the search expression is compared against title, genre or actors.

**Wishlist Search**  Works similarly to previous search however this time we are searching in wishlist table.

## Client

The core of our application is built with the help of Rest API request fetching data from the server. We also have a database for storing client data and his wishlist movies. The two tables look as follows.



When a new user is creating their profile, we wanted to make sure that the information he enters into the database is the correct one. We wanted to avoid mistakes like first and last names containing numbers or special characters, or age being either a negative number or number too big to make sense. Empty fields were also not permitted.

This was taken care of in UserModel class using Attribute classes like Stringlength, Range or specifying a RegularExpression.

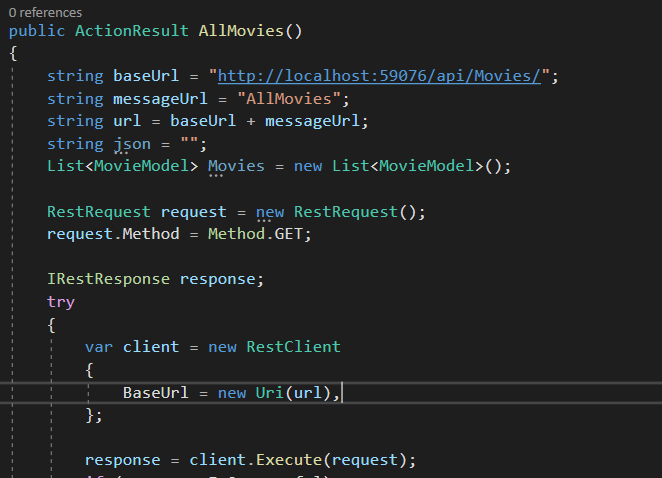
### Client-side VS Server-side

There was an important discussion on how to handle the requests and responses.

On one hand we could use ajax to asynchronously send the request when the user loads the page, the request would be triggered by javascript. This option would ease up the load on Client Server but would make it harder for us to test and “work” with the response, since we would lose the strongly typed models. In the end we ended up going with a server side request. Using an external library.

### Rest Sharp

This free to use package provides support for Rest Services and has easy to use documentation. Our usual request construction looked like this:



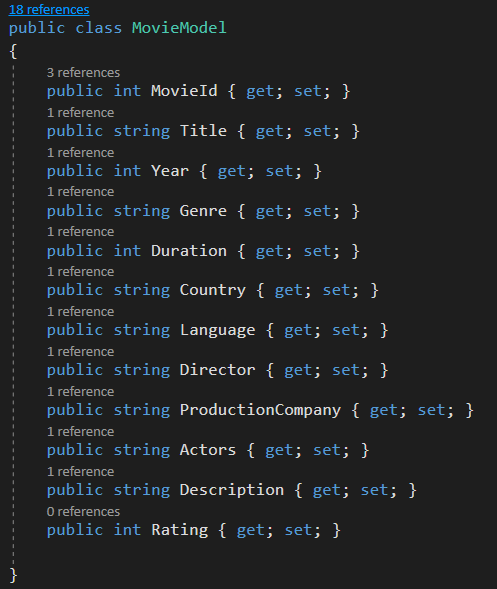
First we would create the url with the resource that we are trying to access then create a new client with this url, after executing the request there would be the need to handle both successful and failure scenarios. You can find examples of these in MovieController.cs in the MovieClient Project.

### 

### 

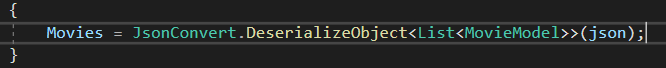
### Models

Since responses/movies coming from Server are in a json format. We needed to cast these into strongly typed models for better manipulation and error handling.



To ensure smooth conversion we opted for simple types as strings and integers.

The conversion itself happens after successful request response:

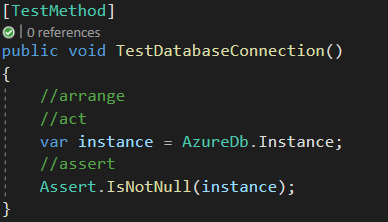


# Testing

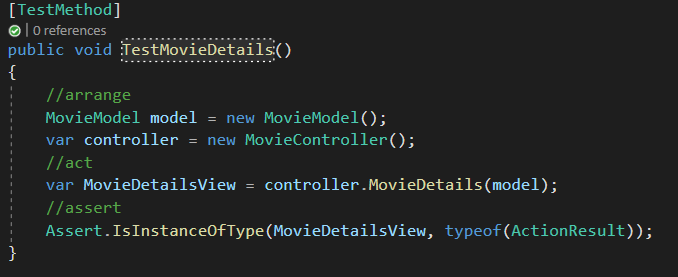
As we mentioned in the beginning we tried to combine knowledge and practice from different subjects, this naturally means we had to test our solution in some ways. The two most straightforward and best suited methods in our opinion are Unit Tests and User Interface Tests.

Please note that as in many other companies and projects, our testing was concluded at the end of the development period and had low priority due to the fact that we had to deliver the project first. Still we have small test coverage, unit testing vital methods of our code with the popular “AA (triple A) method”. Arrange, Act, Assert. All of our tests are conducted on the client application.

## Unit tests



In this first example we can see a simple test of our database instance through which all our calls to the database are executed. Failing test of this sort would raise a red flag and we would have to investigate the connectionstring and if the azure database is still running



The second example shows testing of one of our views. This is a bit tricky since this method ( MovieDetails ) has a return type of ActionResult. Under different circumstances we would be able to assert if two values like string or integers are equal. In this case however we test a successful view creation and we verify this by checking on type.

## Selenium

Because our project is an MVC web application and users interact with it via the web interface, it is necessary to perform some sort of UI Tests. During our Software Testing course we were introduced to browser automation with Selenium.

Selenium provides many different components for automation but we decided to use Selenium IDE, an extension for Chrome (or add-on for Firefox) which allowed us to do simple recordings and playbacks of most interactions with the browser. There is also an option of Selenium WebDriver, but for our scope the IDE was more than sufficient.

We had tested four main scenarios which we believed to be most used by our users. Those are:

* Ability to Login and Logout
* Add movie to a Wishlist
* Search for a specific movie
* Search for a specific genre

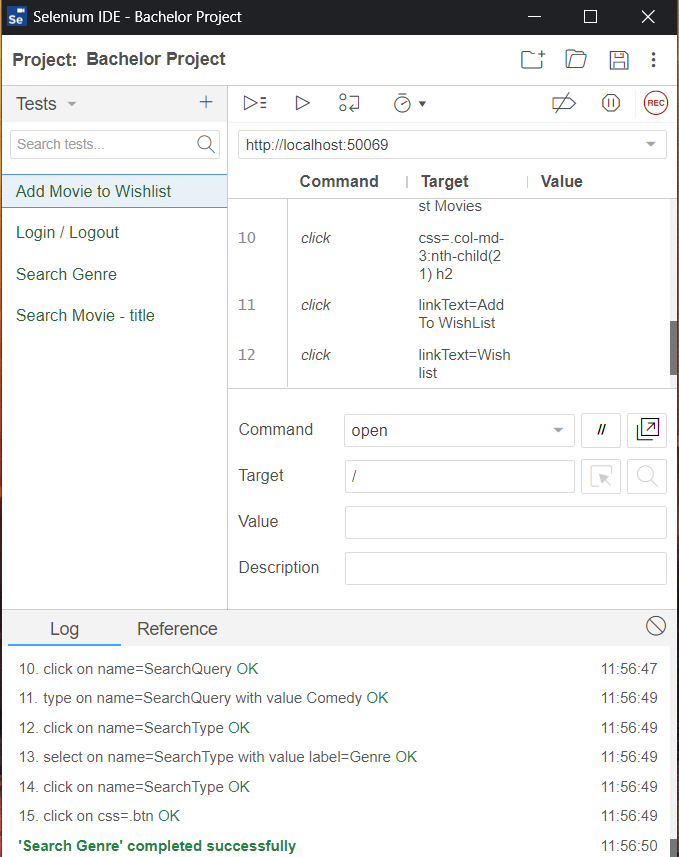
These test scenarios are pretty simple. There is a general “rule of thumb” that many teams use:

*“Automated acceptance tests* ***should not*** *run longer than* ***1 minute*** *on your local resources.”*

Tests that complete in 2 minutes or less are twice as likely to pass as tests lasting longer than two minutes. In other words, the longer a test takes to run, the more likely it is to fail.

We aimed to create autonomous, atomic tests that are quick, so when a test fails, we have a very clear idea what feature needs to be fixed.

Selenium IDE extension with tests.



# Security

In this section we will talk about why security is important in our scenario and what procedures we put in place to ensure the safety of user information. We will also mention possible security risks that could arise.

## The need of security

The main reason for security in our project is to protect user personal information from being made public and to ensure that people can not access accounts that do not belong to them.

The General Data Protection Regulation (GDPR) distinguishes between ‘personal data’ and ‘sensitive personal data’. Personal data is defined as follows:

*“‘Personal data’ means any information relating to an identified or identifiable natural person (‘data subject’).”*

In other words, any information that can be used to describe a particular person.

The sensitive personal data, also known as Personally Identifiable Information (PII) is a specific set of “special categories” that must be treated with extra security. They include:

* Addresses
* Names
* Financial Information
* Login IDs
* Biometric identifiers
* IP addresses
* etc...

As we can see the difference between the two is very subtle, but it is clear that we are dealing with user information that needs to be taken good care of, because if made public, they can have varying impact on a person's wellbeing.

## Our precautions

To gain access to one’s personal information, they need to log in into their account. Therefore our focus was on this first ‘gate of defense’ to protect our users from malicious attacks.

When users register and set a password for their account, this password is encrypted to be stored in a secure way. For encryption, we decided to go with Secure Hash Algorithm 2 (SHA-2), more specifically the SHA-512 hash function.

SHA-512 is a novel hash function computed with 64-bit words. It is a very secure algorithm which is used in present development.

The actual encryption is done using SQL Server stored procedure which takes advantage of HASHBYTES function. User enters his password, which is immediately encrypted and stored in a table in a varbinary type.

## Security risks

Even though our passwords are hashed, they can still be cracked using pre-computed hash attacks, for example a Rainbow table attack. To defend against this kind of attack we have an option to add a “salt” (a random bits added to a password before hashing) to our passwords. It would force the passwords to be unique (in case two different users choose the same password), increase their complexity with no increase to user requirements.

For the scope of this project we decided it was not needed to implement this precaution, but with understanding that it would be a much needed change in case of our application scaling. The needed change would be a simple alteration of our stored procedure which stores user passwords.

Second attack that concerned us is SQL Injection, where harmful SQL statements are written into the entry field and executed in a way they were not meant to be. In our case the login form is the vulnerable form.

SQL Injection is a very common type of attack and therefore we agreed that we need to protect our database and mitigate this form of attack. The method we implemented is by using SqlCommand and its .Parameters property. This way all the work to check for sql injection is handled by these classes with an added bonus of easy implementation.

## Utilization of our education

During our studies at KEA Copenhagen, while attending Software Development course with its classes and modules, our lecturers introduced us to many different

technologies, tools and methodologies. We got a lot of advice on how to improve ourselves and deepen our skills and experiences in particular areas of study. We took this knowledge and experiences and utilized them in this bachelor thesis and application.

Our first semester was divided between Databases and Testing courses. The knowledge gained from these courses was a tremendous help not only during the development of this application, but while doing our internship as well, where some of our tasks were to perform tests. We learned how to do Selenium tests as well as how to write stored procedures, transactions and triggers. Furthermore, we practiced designing and structuring databases but sadly we could not take full advantage of this knowledge since our databases are quite simple without the need for normalization.

Second semester we got introduced to Development of Large Systems and System Integration. In both of these courses we briefly talked about Representational state transfer (REST) architectural style which we used in our application and which is used widely by developers.

In the Development of Large Systems course we learned the theory about modelling of our systems and gathering requirements which helped us with the development process for our application. Last but not least, we were introduced to different kinds of architectural patterns like MVC design pattern, in which our application is created.

# Reflections

This section is dedicated to us reflecting on the solution as a whole, what would we do differently next time, what went bad / what went surprisingly good and what concepts were scratched due to the time limit of the project (understand our procrastination) and our incompetence.

I would like to start with the idea of the project itself. When we were deciding what we would like to do for the bachelor thesis, we were considering collaborating with our internship company, but in the end we had decided against it. Looking back this might have been a bad idea, because working on a project for the company would give us a clear idea what to do, have feedback and help from senior developers and also force us to start working on it sooner. It would also give us an opportunity to work on something unique, that would possibly grow and become a part of a working product. Instead, we settled on an idea for an application in which we would put knowledge gained during our studies to test. Only later we found out that it might be harder than we first anticipated.

What went good? Since we both studied alongside each other for a longer period of time we were both comfortable with the technologies that we chose for this project. This was a huge plus when deciding programming language, architectures, tools, etc. Building this application in ASP.NET, using MVC design pattern went very smoothly for us since there is also a lot of documentation and help available online. The same is true for the choice of our database.

Moreover, during this time we had tried to follow a proper set of guidelines and methodologies for project development such as SCRUM. Because of that, we had minimal problems with writing the report as we had some guidelines to follow.

Lastly we believe that working on this project together was a good choice, considering that during our studies at KEA and during previous education, group work is something not only supported but also promoted. We were able to learn from each other and gain a different perspective on problems.

What went wrong? Due to the nature of our project, we only needed a small database to store the data we needed. It didn’t make sense to keep more than just a handful of information about users since this is supposed to be a personal wishlist application without any additional features. Because of that, we did not take full advantage of the skills and knowledge we gained during our database course which is not ideal for a project this important. There was a solution to amend this, but we will talk about it in future perspectives subsection.

As mentioned earlier, our studies were focused primarily on back-end development. This meant that we spent considerably more time on UI than what would be ideal (even though our front-end is very simple) and we were unable to implement some features that we very much wanted to.

One of the problems that we were always postponing and inevitably postponed indefinitely is pictures. At the early stage of development we realized there would be a need to store at least one picture for each movie. We believe this is the only feature that was always moved to the end of the priority que, even though we had a solution drawn for it.

To summarize, we were unable to finish everything that we planned for many different reasons, therefore we are quite disappointed in ourselves as our initial design was something we liked.

What would we do differently? We think that one particular thing could be the change we would be glad to make; to collaborate on the bachelor project with our internship company and create a product which would be used in real life situations. More complex features could be thought of at the beginning, to implement more of what we had learned during the semesters, like microservices and more complex databases. Also we would plan the whole project with better understanding of how much more time could things like travelling and Holidays take away from the development process than what we anticipated.

Future perspectives. What future holds for this application comes hand in hand with the original idea that sparked this project. It was to create some kind of “wishlist of wishlists”, where the user would be able to save not only movies they would like to watch, but also TV series, music, podcasts, books to read, games, gifts, etc… with the possibility to create and customize their own lists. This would allow for various interesting statistics about how many books he/she read in a year, how many new music bands they found and so on.

One way the work on this application can continue is to try to get as close to the original design with a possibility to use it as a part of our developer’s portfolio or to host it on a cloud and provide it for friends for feedback to learn from and develop even further.

The other way is that after we finished with our studies at KEA, we will discontinue the work on this application and learn from the result and our decisions, be it the good, or the bad ones.

# Conclusion

To start off we are pretty satisfied with the development processes and technologies we used but a little less happy with the result we ended up with. It wasn't hard for us to follow the development guidelines and version control. The collaboration, teamwork and communication worked in our favor. However time management and motivation are factors that are hard to stimulate without proper higher management.

**To look back to problem statement:**

“Can we create simple, yet useful application for tracking watched movies and provide users with basic options to manage their lists?”

We tried to solve this problem to the best of our skills, trying to implement knowledge obtained during our studies. Minding holes in our design such as potential of database which is not fully utilized and everall little functionality partially due to the lack of foresight.

**How did we fulfill the scope of this Bachelor thesis with our project ?**

Using our analytical skills we devised a solution to a given problem. We were able to draw out architecture and follow best practices when it comes to the development process. At the end we hope we reached the acceptance level for this kind of work.

# Bibliography

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# Appendices

* Add queries
* Add stored procedures

