1. Introduction

This document summarises the collaboration of Group 2 for the programming and technology exam of the 3rd Semester. The group consists of 4 members of 4 different nationalities. In spite of the major differences in our opinions (we agreed on certain rules and guidelines to follow, thoroughly elaborated in the accompanying document called group contract), we managed to harness the benefits of diverse ideas and identify multiple possible approaches to certain problems.

The idea of the project is to create a service that handles multiple chatrooms with limited people count capacity where people can discuss topics they are passionate about which the users may access either via the web client or the windows application. Listening to music through the application using YouTube Data API and joining groups are other major user stories of our program.

Some of the problems we had to find the answer to include but are not limited to: Users whose number is greater than the available slots in a given chatroom try to join at the same time (for example, there is 1 slot left but 2 people try to join), a group whose number of people is greater than the available slots in a given chatroom tries to join (for example, a group of 4 attempts to join when there are only 3 or less slots left), preventing SQL Injection, learning how to use callbacks and bindings.

After formulating the problem statement and having it approved by the supervisors we were assigned the task of solving the problem by creating a web service, selecting the appropriate middleware and create both a dedicated and a web client.

2.1 Service Related - irrelevant

2.3. Middleware

2.3.1. Client to service

a. Choice

For this project we created a web service to fulfil the needs of our users. Since our users may have devices running different type of operating systems (Windows, Android, iOS etc.) we needed a middleware that can solve this problem. We have chosen Windows Communication Foundation to be the ‘adapter’ between our users and the service thus it is a WCF Service.

b. Pros

* It can have multiple types of bindings including TCP and HTTP.
* Offers great flexibility as it automatically selects the appropriate type of binding depending on what the user device is compatible with (as long as such type of a binding is configured).

c. Cons

* It can be slower than an API that is designed to use only one type of binding, although the difference is very small that it is not worth sacrificing the compatibility benefits.

d. Other possible choices

WebAPI is not a bad choice as it is easier to understand and learn, it is great for HTTP services. However, it is very limited in terms of compatibility compared to WCF.

2.3.2 Service to database

To access the SQL Server database we need a middleware to communicate between C# and SQL.

1. Choice

For the purpose we have chosen ADO.NET. It is a decent data access API and is fairly easy to use.

1. Pros

* Readily supported by any .NET language.
* It is faster than some other APIs , for instance, entity framework

1. Cons

* Bigger queries when compared to entity
* Less encapsulation compared to entity

1. Other possible choices

We were considering to switch to Entity Framework for better security, encapsulation and to cut down on redundant code. However, Entity Framework was not designed to be flexible in terms of database model changes which can happen quite often when doing agile development.

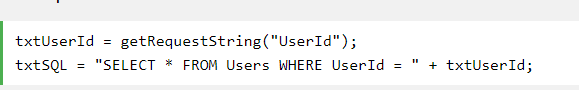
2.4 Security

2.4.1 Password storage

This is the first step we took towards security measures in our project. The last thing we want is if someone manages to compromise the database to be able to obtain the passwords of the unsuspecting users. What we did was to add ‘salt’ (a unique random string) to the password and then hash it before storing it in the database. This ensures that even if someone manages to see all the values in the database they will not be able to make much use of it as the password is hashed. And because of the salt two same passwords will still have different hash values making it even harder to crack. Of course we have to store the salt in the database because it is added to the password every time before it is hashed and checked against the password hash stored.

2.4.2 SQL Injection

SQL Injection is a very wide-spread way of executing malicious code on a database. It occurs when a user enters data in such a way that it executes SQL commands that are not supposed to be executed. The following examples are taken from <https://www.w3schools.com/sql/sql_injection.asp>.



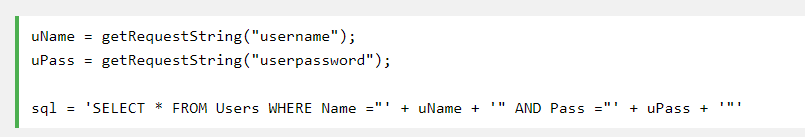
In this case you can cause SQL injection by making a statement always true if the user inputs this for example:



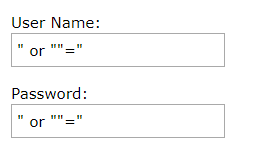
This will execute a statement which looks like this:



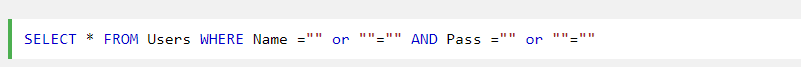
That would be a big problem if that table contains some sensitive information like passwords for instance. The user could enter any command after the id in that textbox. They could drop a table and destroy the database which will probably destroy the program’s functionality.



A statement like this can be bypassed by simply entering the following input:



What will be actually executed is this:



It will select all rows from the users table since the condition after ‘or’ is always true.

Here is an example how we protect our database against SQL injection:



Declaring an SQL command that takes only parameters rather than strings prevents users with bad intentions of executing any code that they should not. Not only is this safer, but it also makes the program more robust by returning an integer of how many rows were affected by this statement. In this way we can easily check if the statement was successfully executed. And if that weren’t enough, it makes the code way more readable and easier to write instead of having a large number of concatenated strings.

2.4.3 DDOS Protection

In due course.

2.6 Performance

To be continued…