# OMEGA – MESSENGER/MAIL SYSTEM


## David Helt

## C4a

davidhelt2004@seznam.cz

774 317 072

# Project Overview

The project is a messaging application with a focus on security and user verification. It includes a feature to distinguish between genuine users and automated bots using a captcha mechanism. The application is developed in C# and utilizes a SQL Server database for data storage and management.

## Key Components

**Captcha Verification**

Implemented in the Captcha.cs file, this feature is designed to enhance security by verifying that the user is a real person and not an automated script. When the user interacts with a specific part of the UI (e.g., clicking on a picture box), a message box is displayed as a part of the captcha verification process. Upon successful verification, the application transitions the user to another form, indicating a successful captcha check.

**Database Management**

The Database class, located within the Omega namespace, is responsible for all database-related operations. It employs the **Singleton** design pattern to ensure that only one instance of the SqlConnection is created and used throughout the application. This approach optimizes resource usage and maintains a consistent database connection state.

**Features**

**1.**
**Singleton Database Connection**
A single instance of the database connection is maintained to ensure efficient use of resources and to provide a centralized point of database operations.
**2.**
**Dynamic Connection String Construction**
The application dynamically constructs the connection string based on settings from the application's configuration file. This includes the data source, initial catalog, and integrated security settings. Optionally, username and password can be included for SQL authentication, that depends on environment that you want to deploy my project.
**3.**
**Captcha Verification**
A simple yet effective captcha mechanism is implemented to differentiate between human users and bots, enhancing the security of the application.

**4.**
**Configuration Management**

The application reads settings such as database connection details from the configuration file, allowing for easy adjustments without the need to modify the codebase.

**5.**
**Hashing passwords**
I implemented hashing of password using SHA-256. I hash given string in this case its password as a hexadecimal string and convert it, just before storing into database, where it is inserted securely

**Security Measures**

The project incorporates basic security measures, including **captcha** for user verification and the option for integrated security or SQL authentication for database access. These measures are designed to protect against unauthorized access and automated attacks.

Database.cs class, this Database connection uses Singleton connection through whole app

```
Menu.cs | LoginForm.cs | Help.Designer.cs | Help.cs | ForgottenPswd.Designer.cs | App.config | Database.cs
Omega | Omega.Database | GetInstance()
using System.Threading.Tasks;
using Omega;

namespace Omega
{

    public class Database
    {
        /// <summary>
        /// Holds the singleton instance of SqlConnection.
        /// </summary>
        private static SqlConnection conn = null;

        /// <summary>
        /// Private constructor to prevent instantiation.
        /// </summary>
        private Database()
        {

        }

        /// <summary>
        /// Gets the singleton instance of the SqlConnection. If the connection is not already established,
        /// it creates a new connection using the settings from the application's configuration file.
        /// </summary>
        /// <returns>A singleton instance of SqlConnection.</returns>
        public static SqlConnection GetInstance()
        {
            if (conn == null)
            {
                // Create a new SqlConnectionStringBuilder to build the connection string dynamically.
                SqlConnectionStringBuilder consStringBuilder = new SqlConnectionStringBuilder();
                consStringBuilder.DataSource = ReadSetting("DataSource");
                consStringBuilder.InitialCatalog = ReadSetting("InitialCatalog");
                consStringBuilder.IntegratedSecurity = bool.Parse(ReadSetting("IntergratedSecurity"));

                /* Add username and password if required   //UNCOMMENT ONLY IF YOU WANT TO USE USERNAME AND PASSWORD IN SQL AUTHENTICATION
                if (!consStringBuilder.IntegratedSecurity)
                {
                    consStringBuilder.UserID = ReadSetting("Username"); // Add username
                    consStringBuilder.Password = ReadSetting("Password"); // Add password
                }
                */
                conn = new SqlConnection(consStringBuilder.ConnectionString);
                conn.Open();
            }
            return conn;
        }

        /// <summary>
        /// Closes and disposes the current database connection if it exists.
```

App.config file, that needs to get updated before launching on other device

```
Menu.cs | LoginForm.cs | Help.Designer.cs | Help.cs | ForgottenPswd.Designer.cs | App.config | Database.cs
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.8" />
  </startup>
  <appSettings>
    <add key="DataSource" value="localhost" />
    <add key="InitialCatalog" value="messengerApp" />
      <add key="IntergratedSecurity" value="True" /> <!-- Set to False when using SQL authentication login; Set to True if localhost -->
      <add key="ClientSettingsProvider.ServiceUri" value="" />
    <!--<add key="Username" value="sa" />
        <add key="Password" value="student" />-->

  </appSettings>
  <runtime>
```

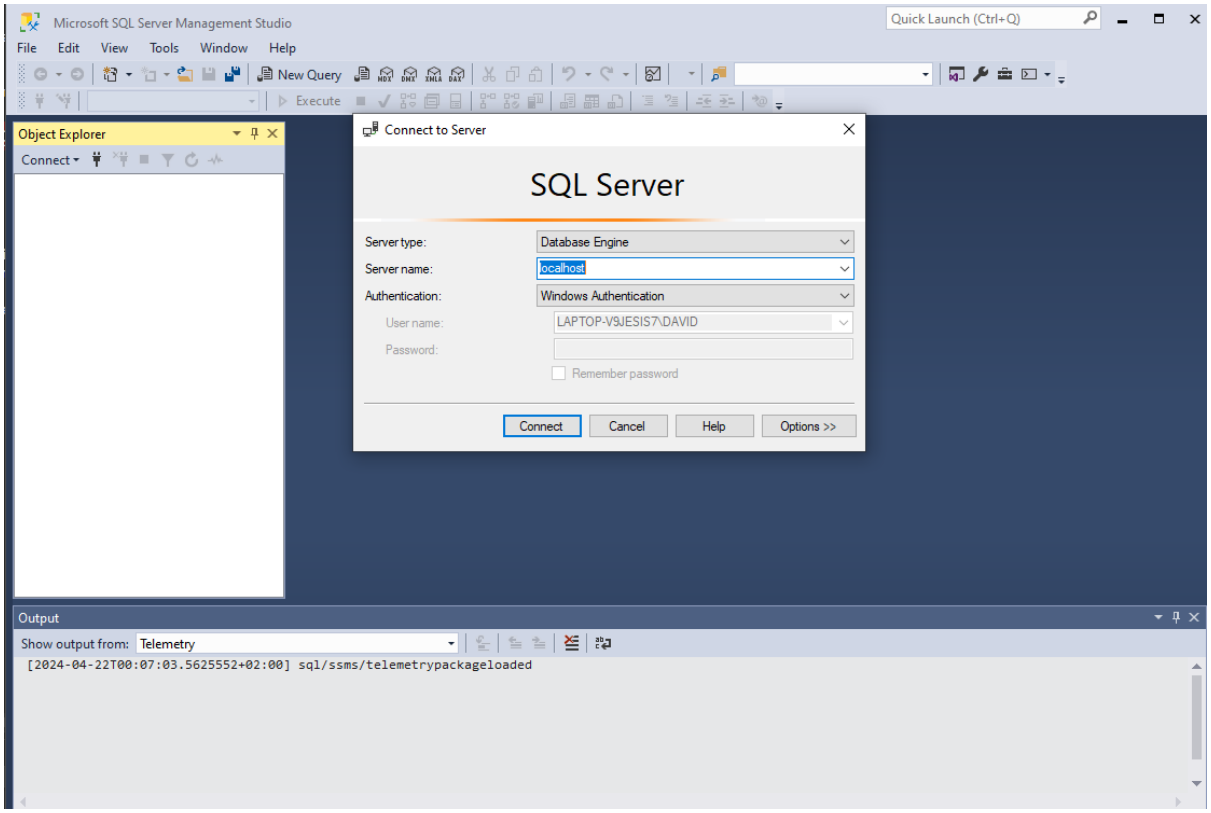
This is the mentioned hashing method, that I use in ForgottenPswd.cs and Register.cs

```
/// <summary>
/// Computes the SHA-256 hash of a given string and returns the hash as a hexadecimal string.
/// This method is used for creating a secure hash of passwords before they are stored in the database.
/// </summary>
/// <param name="rawData">The input string to hash.</param>
/// <returns>The hexadecimal string representation of the SHA-256 hash.</returns>
Tabnine | 1 reference
private static string ConvertToSha256Hash(string rawData)
{
    using (SHA256 sha256Hash = SHA256.Create())
    {
        byte[] bytes = sha256Hash.ComputeHash(Encoding.UTF8.GetBytes(rawData));
        StringBuilder builder = new StringBuilder();
        for (int i = 0; i < bytes.Length; i++)
        {
            builder.Append(bytes[i].ToString("x2"));
        }
        return builder.ToString();
    }
}
```

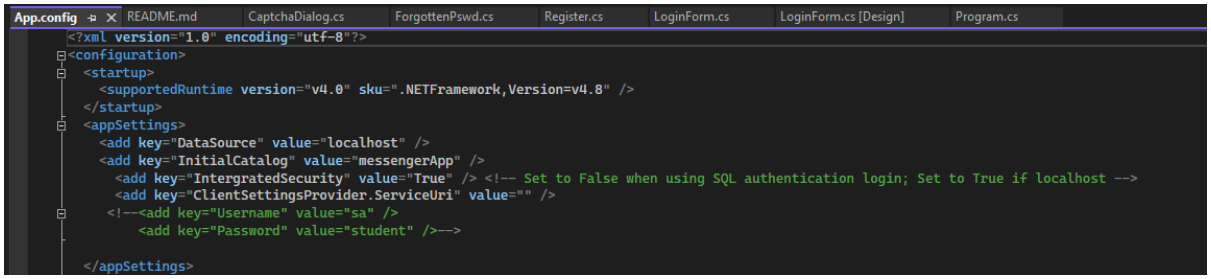
# *Setup of project*

To launch project on your device, you either follow Readme file, that is cointained in Omega directory, or follow instructions below. I recommend reading this .Pdf

Firstly, you launch MSSQL, and when this window to connect to server appears, copy Servername and paste it in app.config - DataSource, which is screenshoted below

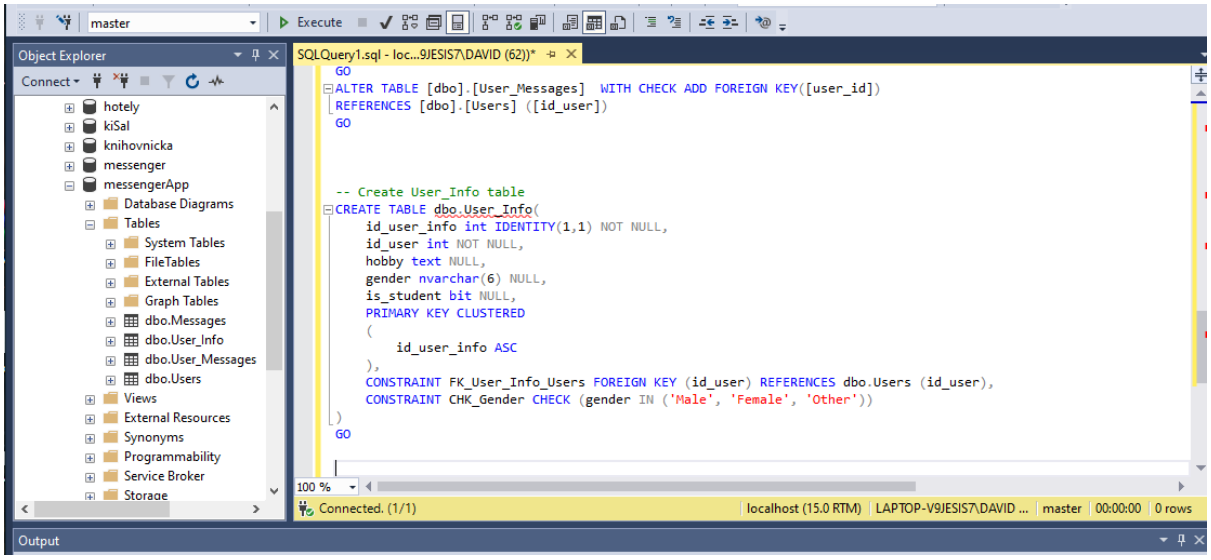



If you use different connection than Windows authentication, you will be asked for username and password. That depends individually, but in my case, it will be SQL authentication, with username and password, that I just need to uncomment and rewrite IntegratedSecurity to false

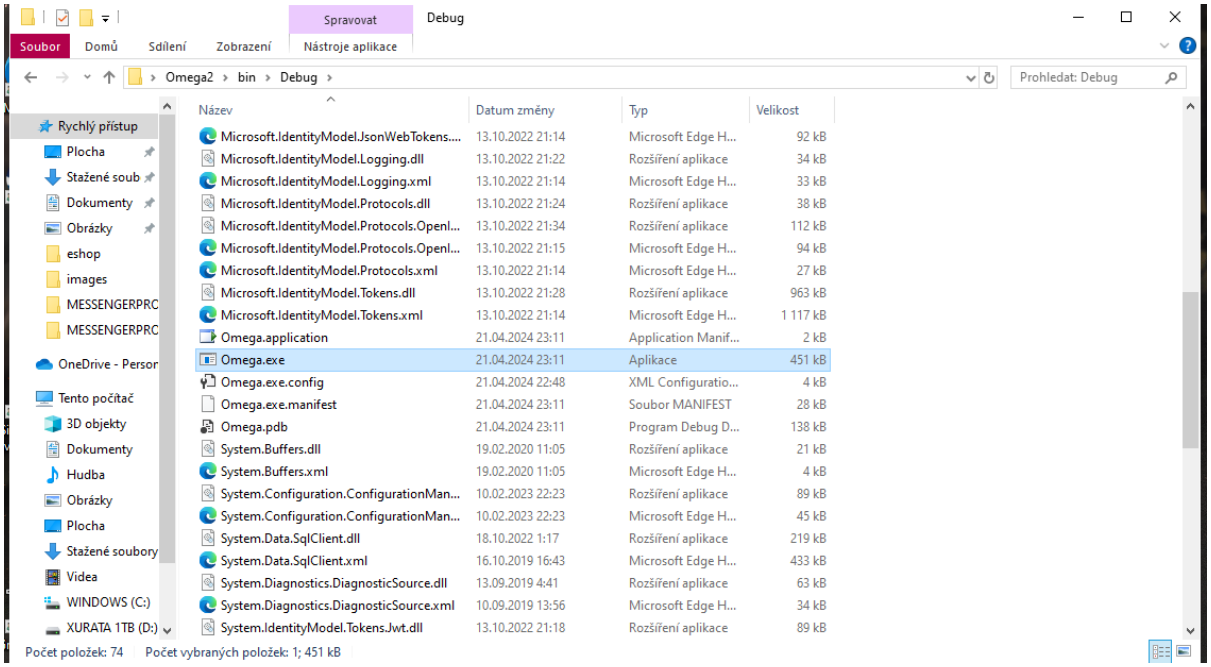



Once you have done that, you need to import SQL to database that you will name messengerApp,

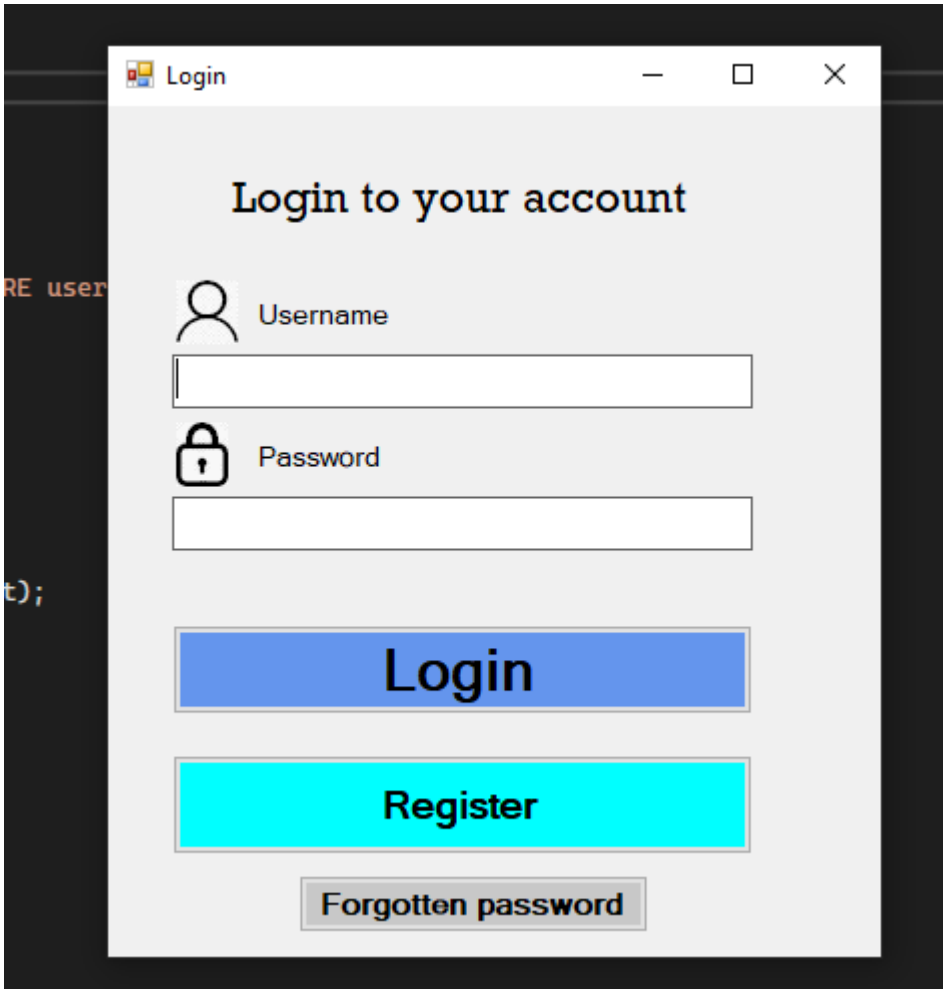
Import SQLImport




And at last, you can start my project by opening omega.exe, which is in bin,debug, omega.exe



If app doesnt load, you need to start VS2022 go to Omega.sln, rebuild project and now run it from VS or .exe file. Depending on which you prefer, program will stay the same



# *Used Technologies*

C#

C# Forms

.Net version 4.8

Microsoft SQL Server Management Studio

SQL