Czech University of Life Sciences Prague

Faculty of Economics and Management

Department of information engineering

Bachelor Thesis

Web Application Solution for communication application

Vait Ameti

CZECH UNIVERSITY OF LIFE SCIENCES PRAGUE

Faculty of Economics and Management

BACHELOR THESIS ASSIGNMENT

Vait Ameti

Informatics

Thesis title

Web Application Solution for communication application

Objectives of thesis

The objective of the thesis is to create a backend side for web application to maintain security and reliability. Server and client will communicate effectively and securely throughout internet. The application will offer to Log in, create and edit features for users. User can communicate with other users with messaging each others. Each user will have own attributes such as password, username, Photo. The application will include the frontend which has developed according to human-computer-interaction design specifics. The design was delivered to end users as efficient as possible and the API will serve between the frontend and backend. The front end will have basic user interface for the application where the most used components will be placed to the most frequently used areas in web browser, which will help the end users easily reach what they need.

Methodology

The aim of the work was to create a web application using the object-oriented approach. The C# programming language with ASP.NET core framework is used to create this web application. MVC(model-view-controller) template was used to keep my files organized. Angular framework for typeScript has been used for the solution for the frontend, alongside with bootstrap framework for CSS to maintain styling. All of the components was connected at the beginning using walking skeleton model. Microsoft entity framework was used to handle with the data and database. JSON web tokens was used to transmit data between parties. After that the Application will gradually developed.

The proposed extent of the thesis

30-40 pages

Keywords

ASP.NET, C#,OOP, MVC, Web application, Encapsulation, Inheritance

---

Recommended information sources

CONERY, Rob. Professional ASP.NET MVC 1.0. Indianapolis: Wiley, c2009. Wrox programmer to programmer. ISBN 978-0-470-38461-9.

DORRANS, Barry. Beginning ASP.NET security. Chichester: Wiley, c2010. Wrox programmer to programmer. ISBN 978-0-470-74365-2.

FARRELL, Joyce. Microsoft Visual C# 2017: an introduction to object-oriented programming. Seventh edition. Australia: Cengage, [2018]. ISBN 978-1-337-10210-0.

PRICE, Mark J. C# 9 and .NET 5 – modern cross-platform development: build intelligent apps, websites, and services with Blazor, ASP.NET Core, and Entity Framework Core using Visual Studio Code. Fifth edition. Birmingham: Packt, 2020. ISBN 978-1-80056-810-5.

Expected date of thesis defence

2022/23 SS – FEM

The Bachelor Thesis Supervisor

Ing. Marek Pícka, Ph.D.

Supervising department

Department of Information Engineering

Electronic approval: 31. 10. 2022

Ing. Martin Pelikán, Ph.D.

Head of department

Electronic approval: 30. 11. 2022

doc. Ing. Tomáš Šubrt, Ph.D.

Dean

Prague on 09. 03. 2023

Declaration

I declare that I have worked on my bachelor thesis titled "Web application solution for communication application" by myself and I have used only the sources mentioned at the end of the thesis. As the author of the bachelor thesis, I declare that the thesis does not break copyrights of any their person.

In Prague on 15th of March, 2023 _____

Acknowledgement

I would like to thank to my supervisor Ing. Marek Pícka for his time, instructions, and advice that was very helpful and essential during the writing of this thesis.

Web Application Solution for communication application

Abstract

This thesis was formed on technical development standards for C# programming language of a communication application. The author was capable of building an MVC model web application using ASP.NET framework as backend configuration for the application, in order to test and have a user interface Angular framework for TypeScript has implemented. The database was implemented with SQLite for better development environment. The Data was handled by Microsoft's entity framework core in order to make the database management easier. SignalR was implemented for better user experience. MVC model has successfully implemented separating model, view, and controller according to MVC standards. Bootstrap design classes have been used according to development standards. Cloudinary platform has been used for photo management which provides free cloud storage and makes easier to do API requests. The security has maintained by JWT for safe server client communication.

The theoretical part was devoted to give basic knowledge for developing any web application and focused mainly on communication type of application. Theory part creates an understanding of general concepts in developing web applications.

The practical part was devoted to create a working and running a secure web application where users can message instantly, provide their photo, and socialize.

Keywords: MVC, ASP.NET, C#, entity framework, Bootstrap, Angular, TypeScript, SignalR, SQLite, API

Řešení webové aplikace pro komunikační aplikaci

Abstrakt

Tato práce byla vytvořena na standardech technického vývoje pro programovací jazyk C# a komunikační aplikace.. Autor byl schopen vytvořit webovou aplikaci s modelem MVC využívající framework ASP.NET jako backendovou konfiguraci pro aplikaci, aby bylo možné otestovat a mít implementované uživatelské rozhraní Angular framework pro TypeScript. Databáze byla implementována s SQLite pro lepší vývojové prostředí. Data byla zpracována jádrem entity framework společnosti Microsoft, aby se usnadnila správa databáze. SignalR byl implementován pro lepší uživatelský zážitek. MVC model úspěšně implementoval oddělení modelu, pohledu a regulátoru podle standardů MVC. Třídy designu bootstrapu byly použity podle vývojových standardů. Pro správu fotografií byla použita platforma Cloudinary, která poskytuje bezplatné cloudové úložiště a usnadňuje provádění požadavků API. Zabezpečení zajišťuje JWT pro bezpečnou komunikaci se serverem a klientem.

Teoretická část byla věnována základním znalostem pro vývoj libovolné webové aplikace a zaměřila se především na komunikační typ aplikace. Teoretická část vytváří pochopení obecných pojmů při vývoji webových aplikací.

Praktická část byla věnována vytvoření fungující a provozované zabezpečené webové aplikace, kde mohou uživatelé okamžitě posílat zprávy, poskytovat svou fotografii, a socializovat se.

Klíčová slova: MVC, ASP.NET, C#, entity framework, Bootstrap, Angular, TypeScript, SignalR, SQLite, API

Table of content

List of pictures

List of abbreviations

MVC- Model view controller

VB- Visual Basic

VS- Visual studio

UI- User interface

API- Application programming interface

CLR- Common language runtime

IM- Instant messaging

E2EE- end to end encryption

CSS- Cascading styles sheet

HTML- Hypertext markup language

OS- Operating system

IDE- Integrated development environment

OOP- Object oriented programming

IIS- Internet information service

ASP- Active server pages

GUI- Graphical user interface

EF- Entity Framework

SQL- Structured query language

ORM- Object relational mapper

GPS- Global position system

CORS- Cross origin resource sharing

JSON- JavaScript Object notation

JWT- JSON web token

REST- Representational web transfer

URI- Uniform resource identifier

URL- Uniform resource locator

WWW- World wide web

SPA- Single-page application

CLI- Command line interface

UTC- Coordinated universal time

HTTPS-Hypertext Transfer Protocol Secure

ASP- Active Server Pages

SOAP - Simple Object Access Protocol

XML - Extensible Markup Language

IANA -Internet Assigned Numbers Authority

1 Introduction

Messaging and socializing between other people from distinct locations has been hard until 21st century. One of the best things that technology has brought to humankind which are social beings is socializing through internet with your family, friends and even people you have never met that are thousands of kilometres away from you, instantly. (Damjan, 2022) [1]. This number can prove that communication over online services is in remarkably high demand especially usually those applications are usually free such as: WhatsApp, Viber, Messenger, etc.

In today's world, communicating through Online services has replaced usual offline services. The reasons for that are that it is free to communicate online, all you need is internet. Also, the complexity of building online services is easier, there are a lot of sources about how to build communication applications, even that you can do it for free. As of 2021 there are 5.3 billion downloads of WhatsApp globally.[2] There are more applications that are globally famous and have millions of downloads.

Developing communication application has many aspects which means you can always upgrade and bring new features to end users. Some of the aspects are:

1.**instancy**: which can be improved by faster implementations

2.**compatibility**: helps the file and text transfer to be in minimum bytes

3.**security**: there are many approaches you can use and always put on more, one of such is E2EE (end-to-end encryption) which makes messages only be read by the sender and recipient.

4.**User Interface** (**UI**): client side of application can be improved with many innovative designs which will help end users to use the application more effectively and enjoy their time they spend in application.

Communication applications can be always revamped that is why it is worth to learn and know how to develop, they have had effects in our lives in 21st century. Every big communication application has getting updates almost every year with new features and it will grow even more.

2 Objectives and Methodology

2.1 Objectives

The objective of the thesis is to create a backend and frontend side for a messaging web application to maintain security and reliability. Server and client will communicate effectively and securely throughout internet. The application will offer to Log in, create and edit features for users. User can communicate with other users with messaging each other. Each user will have own attributes such as password, username, Photo. The application will include the frontend which has developed according to human-computer-interaction design specifics. The design was delivered to end users as efficient as possible, and the API will serve between the frontend and backend. The front end will have basic user interface for the application where the most used components will be placed to the most frequently used areas in web browser, which will help the end users easily reach what they need.

2.2 Methodology

The aim of the work was to create a web application using the object-oriented approach. The C# programming language with ASP.NET core framework is used to create this web application. MVC (model-view-controller) template was used to keep my files organized. Angular framework for TypeScript has been used for the solution for the frontend, alongside with bootstrap framework for CSS to maintain styling. All the components were connected at the beginning using walking skeleton model. Microsoft entity framework was used to handle with the data and database. JSON web tokens were used to transmit data between parties. After that the Application will gradually be developed.

3 Literature Review

3.1 What is communication application?

Instant messaging (IM) refers to an application that allows two or more individuals to interact with one other using two distinct devices running the same program, or through a website or application accessed by an internet-connected browser. [3]

IM is different than public chats because the communication that has been happening is not seen by many users, it can be seen only by the sender and receiver. [3] There are no multiple users that can see and reply to the messages that are being sent between devices, it is only between the users that are interacting with each other.

The simplest goal of instant messaging is two achieve: keep a track of when users are online or offline to send notifications or give a hint that users are being online. [3] The actions that are happening are seen in real-time without a need for refreshing the page. Both users that are sending and receiving the messages will have accordingly different changes on their devices such as when a sender sends one message then the sender's phone will imply that the message is sent, and the receiver user will get a notification at the same time saying that user has an unread message.

Usually, this kind of software relies on centralized servers where it can track the presence of users, and their actions by sending real-time data to the server and back to the user in form of notification or information or an icon symbolizing the presence, unread message, etc. [3]

3.2 Object oriented programming

Object oriented programming (OOP) is a software principle which models the data as a class. OOP is a concept that was founded in the 1967 software crisis as an answer, and it has been developed since. [4] OOP is a way of implementing the code according to some set of practices. [4] OOP can be referred as reusable code, meaning everything is an object which we can use the objects many times, this is what makes OOP flexible software that is satisfying the needs for a dynamic application at runtime. [4] There are different ways how to implement OOP but not just one. We can see real examples, for example, JAVA and C++ have been developed according to OOP standards, but they are implementing OOP in their

own way. There are mainly two approaches when it comes to OOP, class-based and prototype-based languages. Class-based programming languages are C#, C++, and Java. The **class-based** means common properties of objects with the template written in a class. The objects are manipulated inside the software after it is initiated. During runtime objects are initiated but not the classes.

The **prototype-based** applications you do not initiate but you clone objects and that is the biggest difference between class-based and prototype-based approach inside OOP. [4]

Object oriented programming has been around for a while but yet it is such a big concept and have different approaches it is hard to say what it is exactly, although most of the developers believe that there are set of properties that constructs OOP [4]:

-encapsulation,

-dynamic Binding,

-polymorphism,

-Reusability.

Those 4 pillars are believed what is constructing the OOP. Between programming languages because most of them are using class-based OOP, we can say that the object is an independent entity which can be used with every data type even arrays, every object is unique and they can be called inside a method or used as a parameter. [4]

Encapsulation means hiding data. We can create objects that holds some data, but we don't want the data inside an object to be manipulated directly by external view. External view can only see the interface for the object but not the actual data. This security feature is what makes encapsulation one of the features. It is always recommended to use encapsulation while writing your code which can grant a high security to your applications. [4]

Dynamic binding is a mechanism in object-oriented programming (OOP) where the method to be executed is determined at runtime, based on the type of the object. In dynamic binding, the method call is resolved at runtime, rather than at compile time. This allows for a more flexible and reusable code, as objects can be used interchangeably without having to modify the code that uses them. [4]

Polymorphism can be long to learn and implement as inside itself has many kinds of ways how to implement it, but for the beginning, we can say that polymorphism by its name from translation means `*having many forms*`. Usually, in C#, Java, or C++ we can define

polymorphism as a procedure that allows properties to accept and return values more than of one type. We can also say that polymorphism gets help from inheritance as we can have different objects of the same type and vice versa. [4]

The abstraction layer is the process of defining objects and their behaviours in a way that only the essential details are shown and the rest is hidden. [4]

In conclusion, many major programming languages have been evolving around the principles of OOP since it solves many software problems. [4]. OOP is used to make a software more secure, reliable, and reusable. [4]

3.3 Middleware for security and reliability

Middleware is a software component that is handling request and responses. [5] It is a type of pipeline that determines whether the request will be continued to the next components pipeline. Middleware is used as an intermediate that provides security and reliability to a software. In ASP.NET core is composed of many request delegates that is executed one after another. As it is a security fundamental in ASP.NET core MVC after a request is made the middleware is processed in an order as: exception handler, Hypertext Transfer Protocol Secure(HTTPs) Redirection, routing, Cross-origin resource sharing (CORS), authentication, authorization, endpoint and back to response. We can also have a custom middleware that can create extra security. All these steps are fundamental for a secure web application that has a middleware [5] ASP.NET provides a built-in middleware that if a request wouldn't go through all the specified steps within middleware, it terminates that request. [5]

The middleware inside ASP.NET core is an encapsulation within a class that is an extension function. [6] Middleware is called inside *program.cs* file where we use ASP.NET core for building a web application. Middleware class consists of a constructor and a method which returns a task and has a parameter type of HTTPContext. Middleware is a singleton class that runs at the time when it is constructed only. [6] Inside ASP.NET core framework middleware follows dependency injection structure which is an object-oriented approach. [6]

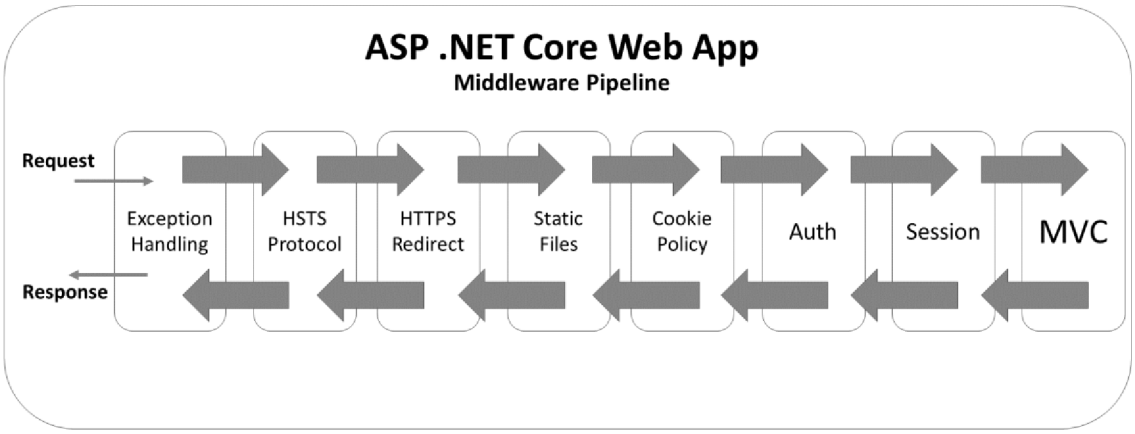


Figure 1. Middleware in ASP.NET Core [7]

3.4 The .NET Framework

.NET Framework is one of the many platforms that has been developed by Microsoft, and its main purpose is used to build and run applications. [8] The .NET framework itself consists of many other programming languages such as F#, C#, VB, C++, etc., and many libraries for building web and desktop applications. [8]

The .NET framework has its own runtime and a library designed to implement object-oriented programming features such as polymorphism, abstraction, encapsulation, and inheritance. [9] These fundamentals of the .NET framework make itself cover all areas of web programming and desktop application programming. Alongside that C# programming language is also specifically designed for .NET technology. [9]

The run-time environment developed for the .NET framework is one of the main parts that makes .NET to lead. Its run-time environment is called Common Language Runtime (CLR) which comes along with implementations such as garbage collector, security, and operability. [9]

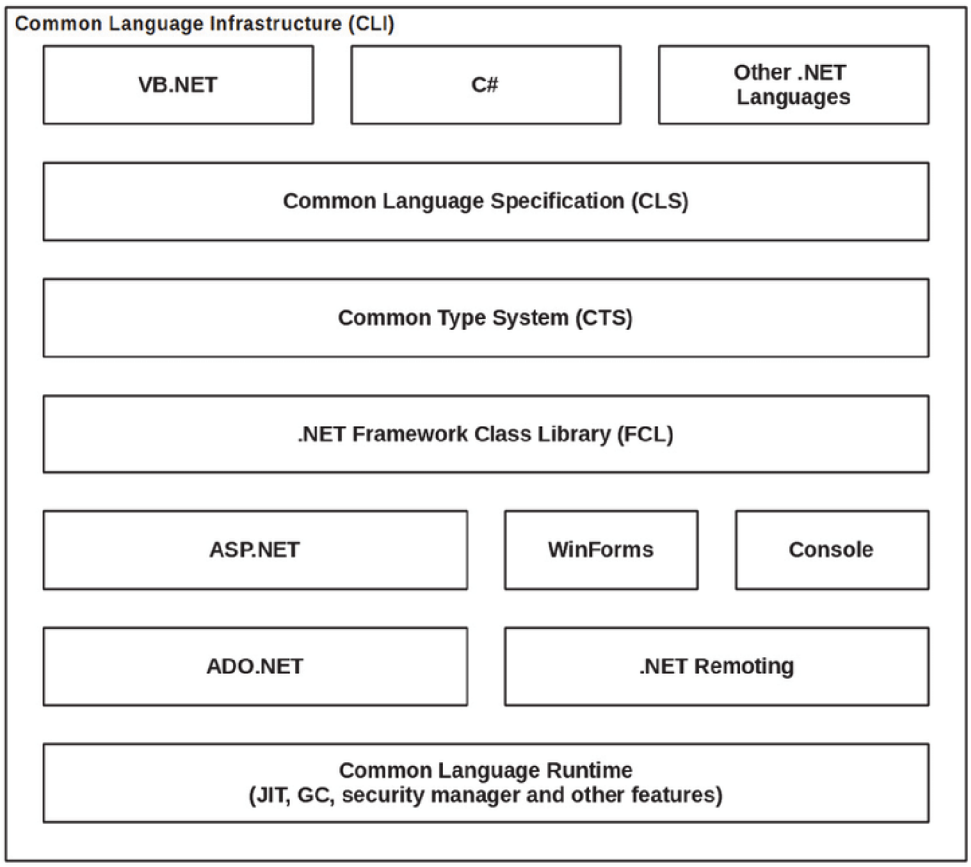


Figure 2. NET Framework architecture [10]

3.4.1 C# programming language

C# is a programming language developed by Microsoft. [11] Its syntax is between C and C++, but many believe that C# has been introduced as an opposition to Java programming language. Java is a similar programming language to C#, they both are modern, object-oriented, high-level programming language. C# is highly customizable still there are areas which doesn't need to be configured by developer and yet is configured by C# itself for example with garbage collector where you are not handling memory as in C or C++ where the developer must specifically take care of memory. C# consists of many several files with a .cs extension meaning source code files of a C# program and those files are containing definitions of classes and other types. [11].Those cs files, which are written in the C# programming language, are processed by the csc compiler to produce executable code that can be run on a computer. The csc compiler is part of the .NET framework and is used to convert the human-readable C# code into machine-readable code that can be executed. By compiling we will have assemblies created for us and we will get the output of executed. cs

file with an extension of .exe or .dll. If we have a file to compile for example named "HelloWorld.cs" we will have the file name "HelloWorld.exe" created by C#'s compiler and we can run this .exe file on any computer by just double clicking on it just like a usual software if we try to run this .exe program on a computer which doesn't have .NET installed, we will have an error. [11]

By the means of object-oriented pillars, C# supports inheritance and multiple inheritance interfaces. [9] C# helps developers develop programs also provides delegates, properties, and events. Usually, the delegates and events work together. Every type in C# and the other .NET programming languages uniforms a type system that treats even primitive types such as integers and characters as objects. [9] Reference types and other objects in C# is stored on method stack rather than heap stack which helps the garbage collector not to look after every little change and creates an efficient friendly environment for the software. C# is one of the first languages which might be extended by attributes. [9] Attributes are metadata that can be used in classes, fields, and methods. [9] For example, it creates an environment for making an API (application programming interface) request, we can give attributes directly to an HTTP request, we can set a return type in an attribute and give authorization through an attribute, etc. [9]

The advantages of the C# programming language are a lot. The popularity of C# comes also from its creator company which is Microsoft. Microsoft is constantly bringing new features to C# every year, being one of the biggest companies in the world, it gives people trust that the programming language will continue maintained and further developed. [12] There are a lot of C# developers around the world which also is a huge community to help each other. You can find many issues that you are facing and find solutions easier without a need to spend time. Another aspect of C# is being a statically typed language. Statically typed languages are way easier to debug, we might have found a bug way easier relatively to dynamically typed languages. In conclusion, C# is a highly secure, easy, maintainable, maintained language with a huge number of users and a helpful community. [12]

The disadvantages of C# also give us thoughts about whether we would like to learn it. Every programming language has its own disadvantages but when it comes to C#, its own disadvantage is that it might be slower due to memory management. Sometimes the

developers prefer to use their own memory management styles rather than relying on C#'s own garbage collector. [12] In the one of the disadvantage of C# used to be that it was serving only in windows, and you couldn't code on Linux or macOS because C# was closed source and you didn't have an Integrated development environment (IDE) or editor which were supporting C# but today you can create applications on any operating system without any problem. [12]

3.4.2 Data types of C#

The data types in C# can be divided into two simple categories as reference types and value types. [13] From its name we can understand that there is a very simple difference between the reference types and value types. Value types stores their data inside that value but reference types of stores only the reference which points to the actual data. [13] Another difference is that value type stores its value inside the stack memory, but reference type stores its referenced value inside a heap memory, in this way we can have two or more reference type variables that reference exactly same value. [13]

The value types can be grouped as structs and enumerations in to two categories. [13] The structs has different kinds of built-in types. For logical operations we can use **Boolean** types, key word as *bool*, we can store inside only two values either true or false which is really handful when we do operations such as loops, logical algorithms etc. [13] For bytes like values we can use **integer** types, for storing integer values we have many options depending on whether how we want to manage the memory because we can use different kind of integer types that ranges of number of bytes it can hold inside for example we can have a *byte*, *byte*, *Int16*, *Int32* keywords and from left to right they expand in range of how many bytes we can hold inside, for single use, we can use *char* keyword which is a type of integer value and it stores only one-byte value. [13] For storing numbers, we can use a **floating-point** which can further divide if we want monetary calculations using decimal type using keyword *int* or if we need precise calculations, we can use *float* or *double* types where there comes a number after the decimal point. [13] Apart from structs, Enumeration type is where we can make our coding more maintainable and easier just by creating constant values for a set of numbers or words, for example, we can create an enumeration by the keyword *enum* with the name "weekDays" and assign to it all seven days of a week and we wouldn't need to remember every day's name but only the variable name "weekDays". [13]

The second group is reference types for data types in C#. There are three built-in and three declarable types. [13] **String** type is to store text values with the keyword *string* which is a built-in type. The **Object,** built-in type, type is the base entity for every other data type both value and reference type, the base entity means that we can have an Object type and set its value to a *string, int, float,* etc. [13] **Dynamic,** built-in type, data type allows us to check the type during run-time rather than during compilation, it is most useful when we deal with APIs. [13] A **class** data type is a template for creating our objects where developers can declare them using the *class* keyword. [13] **Interfaces** are the template for class meaning we must implement every property and method inside the interface to a class also with the keyword *interface*. [13] **The delegate** data type is mostly used to encapsulate a method that applies the object-oriented principle, and we can do that with the keyword *delegate*. [13]

3.4.3 ASP.NET framework

Before the version of ASP.NET core came ASP.NET and it was not as popular as today because it was not open source and you couldn't develop in Linux and macOS rather you had to develop your web applications in windows operating system(OS) because the IDE's and editors supporting C# was way more in windows OS, for other operating systems you had to have some third party editor which was not as better and efficient as the ones in windows such as Visual Studio, notepad with C# and etc. ASP.NET core framework, on the other hand, is open source and highly maintained and used in cross platforms. [14] It doesn't matter where you develop because you can use visual studio code in every major operating system, and it has maintainability and security. There is no more need for third-party applications for developing web applications with C# in Linux and macOS. [14] Today ASP.NET core is more secure, faster, ready to cloud hosting and there is no longer needed to host only on Internet Information Services(IIS), but you can also host your applications on Kestrel web server. [14]

One of the fundamental parts that is inside .NET technology is ASP.NET. ASP.NET is specifically designed framework to handle web applications through the internet. ASP.NET used to be called only active server pages (ASP). [9] The old version and the new version have in common is the core idea behind ASP.NET which is providing creation of assemblies directly on run-time when we need the assemblies. [9] This has an impact for

efficient application. We can ask for data from server only when we need, and it makes our application way faster than usual ways. ASP.NET has been getting constant updates from Microsoft and it has a lot of users with a helpful community, meaning ASP.NET is going to be around for a long time. ASP.NET is essentially different than ASP in its architecture such as [9]:

Object orientation: ASP.NET pages compiles into classes which gets features from .NET framework as it inherits from *system.class*. ASP.NET also is very compatible for front end graphical user interface (GUI) elements such as buttons, text fields etc. because they also are compiled into classes.

Web Controls: ASP.NET has a huge library for web controls which exceeds even Hypertext markup language (HTML) web controls, every web control ASP.NET offers is compatible with browsers. All the web controls offered by ASP.NET is treated as object which makes for us easier to implement and understand. By working with web controls offered by ASP.NET we can create many events for any web control we use and in this way thanks to object oriented way we can have control of our application and build any web control according to our needs.

Separation of layout and logic: since ASP.NET we have a clearly separated layout and logic as the layout is narrated in *aspx* file and the logic written in any .NET specific language such as visual basic (VB) or C# or F# is existing as a clean solid code without any HTML. With this way it is way easier for backend developers and frontend designers to create their own work without the need of interruption one and another. In the ASP version there was no clear distinction between the layout and logic.

Interactive design of web pages: With the ASP.NET version we can create web pages and by only dragging the components we would like to appear in the web page without any need of coding we can achieve interactive design. This way we can save a lot of time rather than just coding one by one for minor changes.

Compilation instead of interpretation: During the ASP version the code created in ASP was interpreted into JavaScript or VBScript but since ASP.NET the code is directly compiled without the need of interpreting it into a different script language and this way we get to have a faster application by just compiling the application with its original language that has written usually it is C# or visual basic. Before we also couldn't access to .NET

library since we were interpreting, now we have access to all .NET libraries which is a plus for ASP.NET.

Rendering of application: In the old version of ASP developers after changing the GUI, they had to manually stop and start the application or restart it so they would see the new rendered page as they changed something. Now, with ASP.NET developers don't have to restart their application as it is rendered every time there is a new change in the frontend components. [9]

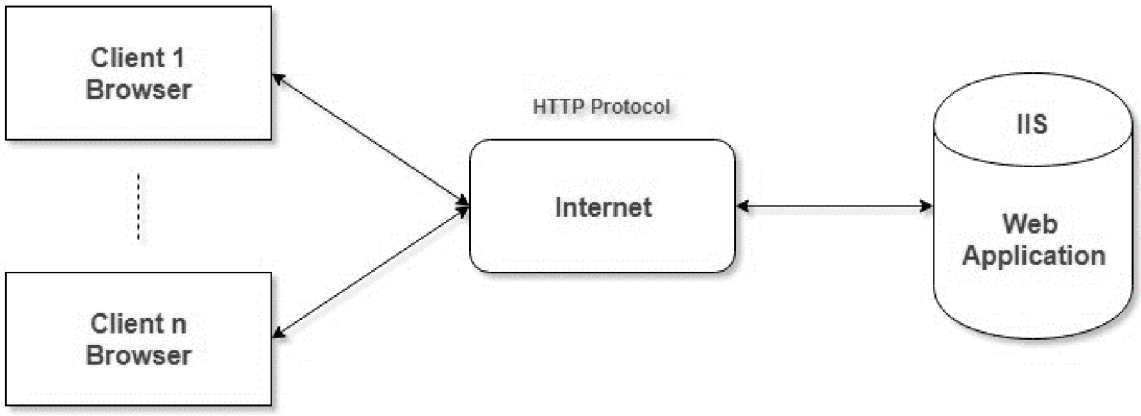


Figure 3. ASP.NET on IIS server architecture [15]

3.4.4 Entity framework core

Entity framework core (EF core) is a library developed by Microsoft for helping developers access databases way easier than usual ways. [16] Entity framework (EF) is an object-relational mapper meaning it connects the software written in an object-oriented way with a relational database with its own API and it makes it easier for developers to access the database. Before entity framework core it was called entity framework, but it was accessible only on windows OS today with entity framework core we can use it on Linux and Apple also. Another part of EF core is that you can do anything that you do with SQL (structured query language) but you don't have to know SQL in order to use EF core because you get to write it for example if you are a C# developer, which is most developers using with EF core, you can write your code as C# syntax rather than SQL syntax. [16] There are of course some downsides of EF core mainly being object relational mapper because the software code and server code has different architectures. For example, if you create a class in C# by the name convention it is always going to be unique but when you need to create a table in database server you have to create a primary key for every table. After you get use

to EF core, developers sometimes seems to forget that they are actually doing also SQL in the background which bring another issue for developers for example if you want to set a name and surname together in C# class you can do it easily by combining two name and surname property in one other property with expression body which is working in C# without any flaws and developer can do that if they don't work with a server, but this would not work in SQL server. [16] These points are the biggest problems of EF core yet what it offers is far more than its downsides. EF core can work both with relational and nonrelational databases, it can do both SQL and NOSQL operations, but this is going to change with EF 6 which is already in production. Since there are many differences between relational and non-relational databases, also usage of relational databases with EF exceeds the non-relational usage, Microsoft has decided with EF 6 and further they will solely develop EF according to relational databases. [16]

For installing EF library, you are going to need to use either visual studio or visual studio code, with NuGett packet manager you can download the library into your application and start using it, of course for testing or run your application you can choose any database server you want. [16]

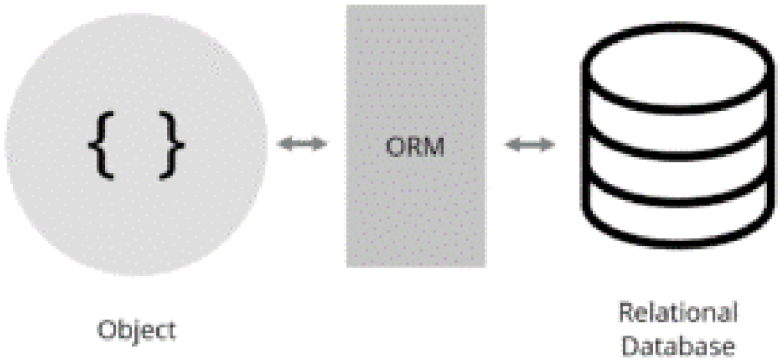


Figure 4. object relational mapping [17]

3.4.5 SignalR

SignalR is an interactive library which was built individually by two ASP.NET developers and later integrated in Microsoft library family. [18] SignalR is using asynchronous techniques to achieve multiuser real time web applications. It can be used even with other technologies rather than only web applications. Library is open source on GitHub,

and anyone can contribute to the source code. The library has been getting updates from Microsoft since the start of it.

SignalR is isolating the developers from low-level programming and itself deals with what is the best way to make the server and client use which can be long polling, WebSockets, etc. [18] Those ways are called components and they communicate through each other. With SignalR we can have a continuously running server dedicated to real-time usage. The connection is constantly running so it can capture every real-time data needed. The library is most known for its usage in messaging, presence tracking, uber like real-time global positioning system (GPS) tracking. [18] The connection is running through end users and servers via hubs rather than HTTP requests. Every server that can install ASP.NET 4 and later is also capable of executing SignalR on their side. [18] The connection is running most smoothly on the IIS server. SignalR creates every user one special ConnectionId which keeps track of every user individually, the ConnectionId usually is used for messaging features of applications as every sender and receiver has a special ConnectionId which thanks to SignalR we can keep track of it. [18] Because SignalR is focused not only on messaging functions between two users, but anything related to real-time, we have the Groups property where we can create connections between many users as a group. When it comes to the client side, we can use SignalR based on JavaScript and even pure HTML, there is loads of client application supporting the implementation of SignalR and that makes the usage very efficient as we don't need third-party providers. [18] We can even run SignalR outside of the server where the original script is running, by having the ability to adjust CORS. All these specifications make SignalR a persistent real-time data transmitter. [18] The library also offers authorization, client authentication, dependency injection, etc. where we can create our real-time data transmission in a secure, reliable, and fast way. [18] There are many programs that are using SignalR, especially Microsoft is insisting on this technology as it is doing a great job with the office365 applications, where most of them use SignalR. These give us a basic understanding of what, how, and where we can use SignalR or any real-time data transmitter technology as it is one of the fundamental technologies needed for creating social and communicative applications. [18] This library has been around since 2013 and it is getting much attention from the ASP.NET community, it seems that this technology will grow each year. [18]

3.4.6 JSON Web Token

JavaScript object notation (JSON) web token is a standard for passing claims between servers and users. [19] It is supported by every major programming language and even frameworks such as Angular and React. The architecture of JSON web token (JWT) is what makes it compact and simple. JWT architecture is using claims which send data in a gibberish-looking way but, it its claims are representing data that is a compact signature for authentication. The main job of JWT is authentication. [19] Claims are definitions of server-side or client-side objects. There are many standardized claims such as signatures. In JWT every server and client have its own unique token created and when we send data from a client to the server or vice versa only the machines with tokens can decode the encrypted JWT. It is a very important standard as it is compact and standard, sending and receiving claims between different machines is one of the fundamentals of why JWT. [19] We can use JWT not only to send claims between objects but also for authentication, authorization, client-side secrets, etc. [19] which means it is a revolution for backend development. Every JWT is established of three main components which are always in JSON objects except the signature[19]:

Header: The header is containing information about which algorithm has been used in that claim, depending on claim it may contain more than one field. But this section is only relevant to the encrypted tokens, if the JWT is not encrypted the header must be settled as null. The encryption usually is a HMAC-SHA256 encryption method.

Payload: This JSON object usually carries out information that we set for extra data, it can be used for Data transfer objects such as when we need an API call that is returning us only name and surname even though we have more than those properties in the object itself. Payloads can include some specific data called registered claims where we can give an expiration time, set an issuer of JWT etc. The payloads can be also public or private if they are not a registered claim. Private is where user defines the exact keywords according to their need and public ones are claims that are registered in Internet Assigned Numbers Authority (IANA) registry, which is official registry for JWT.

Signature: This is one of the important part of a JWT because this is where we create authenticated JSON web tokens, signature is the actual algorithm combined with the data from payload and header and the algorithms inside a signature. The token's signature is what

helps the validation process. Even if third party users can see the token, they will not have a validator by JWT which is the way to decode a token and see the data inside. This way the data with encryption is meant to be decoded between parties that have their own validators and can see what the actual data is and because of this reason, JWT is a tool for authenticity. If we choose to send data that is not really needed to be encrypted, we can directly send JWT without a signature and create so-called unsecured JWT.

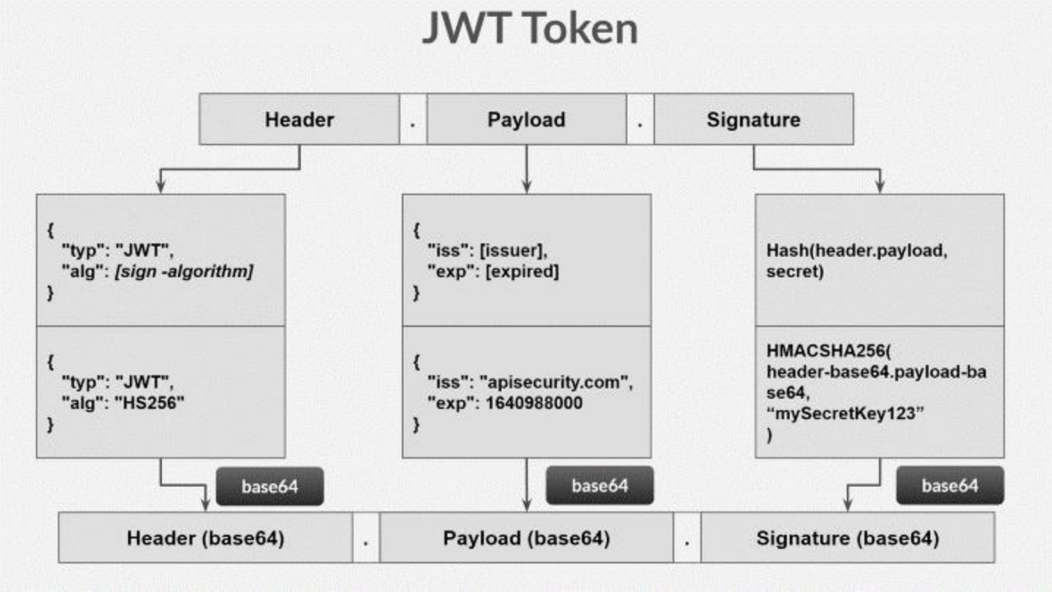


Figure 5. JWT structure [20]

3.4.7 Web API (REST API)

Web API or Representational state transfer (REST) API is a set of control unit architecture for transferring data over the architecture of the world wide web. [21] REST architecture is built mainly as the interface for world wide web (WWW) that is based on HTTP. [21] The REST API should be designed in such a way the Uniform Resource Identifier (URI) would be efficient. Every resource over a web page can be identified with URIs, and that is why the logical implementation in the backend can be designed in such different ways there are templates like Restful or Simple Object Access Protocol (SOAP). The good part of web APIs is that they are reusable and there is no need for creating new APIs for every resource. The job of API is to retrieve or create a resource in the database. This process is one of important processes in software development as we might have big resources to retrieve, and it might be inefficient but by developing API in an efficient way

we can have way faster applications with the abstraction-free process. For retrieving resources over web HTTP there is four simple keywords such as [21]:

GET: By the name we can understand that the GET keyword is used as retrieving the data in a cacheable way thanks to WWW. With the GET keyword, we give a command to API through HTTP, and it retrieves the data from the database accordingly.

Post: Post keyword creates a new resource in the database the same way the GET keyword acted out.

PUT: This keyword is very similar to POST but it is used for updating the database resource so we can use PUT method N-times and yet have the same result N-times unlike POST method.

Delete: the last method for HTTP is delete where it deletes a resource through API. The web API can be returned in many ways such as Extensible Markup Language (XML), XHTML, RSS/Atom but today what usually is used is JSON media type as it is text-based media format, it is very compatible with browsers, and it is compact ways of transferring data. Usually, API is understandable from host extension for example if we have a GET method that retrieves users we can understand that "*www.WebAPIExample.com*"will have a suffix as "*www.WebAPIExample.com/Users*". We can understand from the example that URI has changed in suffix and there was a GET method used as retrieved all the users.

3.5 Model-view-Controller

Model-View-Controller (MVC) is a design pattern architecture used to create interfaces and keep applications organized. [22] With MVC we are separating application into three layers that relate to each other. This design pattern is related with client end to end meaning client uses a request, after that data is manipulated and after that in third layer data is again shown to the client. Because of that separation of layers MVC is a design pattern because every layer is connected yet independent meaning when we need a small change, we can just change it on one layer and the application would still be running, developers use this approach as it is simple to use and it helps developer build their applications faster. The three way process is:

Model: This layer is the purest form of data as it not manipulated, in the model the application rules are defined, and data is not manipulated. [22]

View: The View process is to take the data from model layer accordingly what user has asked, basically it is accessing the model. It can be used as a graph, chart or diagram, any data visualizer can be built in View section. [22]

Controller: The controller layer is interconnecting the model and view layers. Controller is manipulating data according to what the client has asked, and it is sending the manipulated data to the model to be modelled and then the view is triggered to retrieve data from the model. So, the controller is what takes information and turns it into something that the client asked for. [22]

ASP.NET has its own template for MVC and it is a framework that helps developers build web apps and APIs with the MVC design pattern. Microsoft has provided this template according to ASP.NET core standards and it is a pattern constructed design that is helpful for dynamic website development. With ASP.NET core MVC we are provided routing for a very efficient URL mapping within the controller. [23] We can do it by just adding attributes over every HTTP request we do. The HTTP request is a type of *IActionResult* interface inside a controller and we can collect every endpoint under one routing. With ASP.NET core MVC we get to have model validation which gives us power on how to arrange our model. We can identify with basic attributes that our model's property can be null or with [*required*] attribute it must be fulfilled before it is sent to API. If we need a property of type email, we can directly access it from ComponentModel.DataAnnotations attribute called [EmailAdress] and the client will have to send the API a text that is written as an email standard name. We can see that Microsoft has made ASP.NET core MVC framework widely usable by providing routing, dependency injection, filter, and web APIs that help developers build a web application with MVC design patterns. [23]

One of the primary advantages of using the MVC design pattern is its ability to enable parallel development. As the layers are separated and independent, multiple developers can work on different layers simultaneously without affecting each other's work. This makes the development process faster and more efficient. Additionally, as each layer has a specific responsibility, the code becomes more maintainable, and changes to one layer do not affect the other layers. This makes it easier to debug and fix issues in the code, as the responsibility is localized to a specific layer.

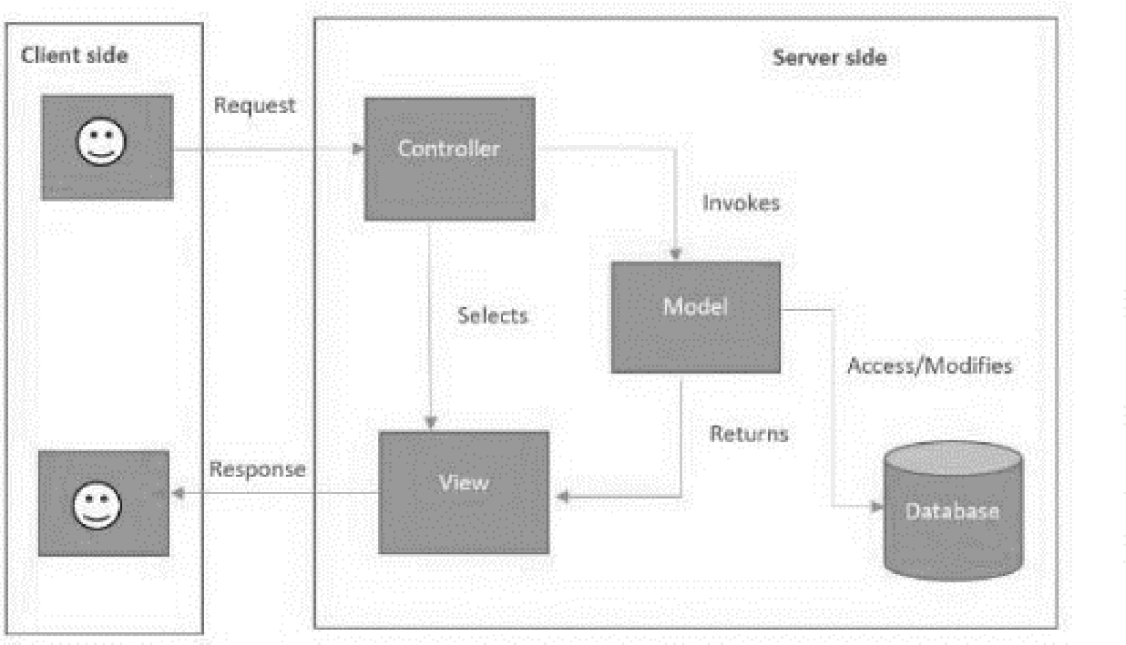


Figure 6. ASP.NET Core MVC architecture [24]

3.6 Repository pattern

Repository pattern is an abstract layer between the controller and database that separates the data logic from create, read, update, and delete (CRUD) operations. [25] Repository pattern is injecting data from the database by the call coming from controller. [25] Entity framework is also supporting the repository pattern. Main logic to separate the data logic is that we would have a cleaner code and we wouldn't have to have such large amounts of code inside the controller. C# is supporting the repository pattern using LINQ operations. By using Language Integrated Query (LINQ) in the repository pattern, we can write queries that are compatible with almost every database and that saves us a lot of time during migrations. That is why it is important to use repository pattern while developing such large applications. [25]

Another advantage of the Repository pattern is that it promotes code reusability. By abstracting the data access layer in the Repository, we can use the same repository across multiple controllers or even multiple applications. This can save development time and reduce code duplication, as we can reuse the same code instead of writing similar code for different controllers or applications.

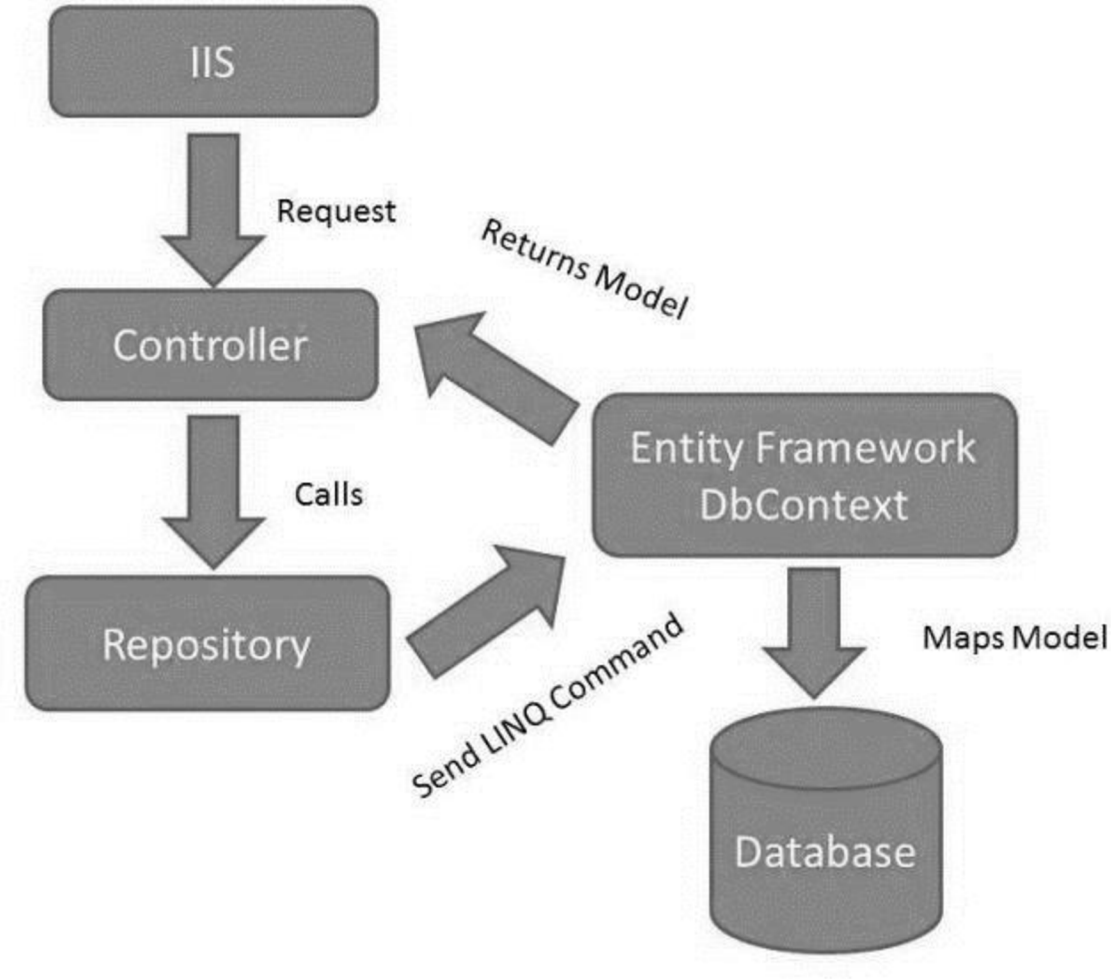


Figure 7. Repository pattern [26]

3.7 TypeScript

TypeScript is a superset for one of the most used scripting languages called JavaScript. [27] TypeScript has been using every component from JavaScript and for extra it adds to language more features. There are many supersets for JavaScript as but none of them has been like typescript because usually the supersets were not supporting every JavaScript feature, or they had to disable some of the features so they could add what they needed. All the JavaScript written code can be executed in TypeScript. [27] That means every JavaScript developer can also develop and understand TypeScript applications. The biggest feature of TypeScript is that it is a statically written programming language as every type is checked before it has assigned invalid value for a type. The variables are declared same way in TypeScript as it is in JavaScript. [27] TypeScript is a language for object-oriented developers, there are interfaces, classes, enumerations etc. TypeScript is transpiled back to

JavaScript and that is why TypeScript can be hosted over any browser as they are designed around JavaScript. [27] Behind TypeScript lies the Type theory where each term has a type and they are restricted based on their type, this way we get to have a more understandable code and less errors within an application. With this theory TypeScript allows us inferring and specifying types meaning we can have a variable that is unassigned in type, but TypeScript will assign a type to a variable as it will infer the type from its content. [27] TypeScript allows us both dynamic and static variable creation which is a huge plus to JavaScript's power. TypeScript also uses structural typing, which is a contrast to most C-like languages as they are using nominal type system, for example in most C-like languages for a class to implement an interface the developer has specifically state that the class is implementing the interface which in structural system you don't have to explicitly state that as long as structure matches specified type. [27]

Object orientation of TypeScript creates a template especially if you are using a different backend such as C# or C++ and you want to keep your code in order than TypeScript is the way to go. TypeScript implements all the tools that are relevant for object orientation such as [27]:

-Classes

-Inheritance

-Encapsulation

-Delegation

-polymorphism.

All these fundamentals show that TypeScript can be used as an object-oriented programming, thanks to Microsoft we can be sure that TypeScript is going to be developed and maintained for a long time. [27]

Once the code written in TypeScript is translated into plain JavaScript, it can be run on any browser or server, but the developers must think about how JavaScript has different features while running on a server or browser. The developer must make sure whether they are going to use the browser approach or not. [27] It has a built-in garbage collector such as C# and it is doing the job of handling memory which makes our applications more efficient. The APIs we are experiencing in TypeScript are usually followed asynchronous architecture but if we need to use an API with a synchronous pattern then we must manage the resources. [27] TypeScript offers the exception handling where we can handle problems occurring in

our application, basically, every object can implement the *throw* keyword which throws an error, but the best practice is to create our middleware or use a try-catch statement exactly like in C#. [27]

Since there is a huge community for JavaScript, there are many frameworks, libraries, and functions created and shared among other developers. This is a huge plus also for TypeScript as almost every framework, library, and code sample created for JavaScript will work with TypeScript also. During the runtime, both the TypeScript and framework will become JavaScript code. [27] With TypeScript, you are going to need to create type definitions for libraries because the frameworks or libraries are not converted to JavaScript as they already are. Type definitions are stored inside .d.ts files. Since JavaScript is a wholly dynamic language when you are going to convert your code to TypeScript you might face many errors that you wouldn't find in JavaScript as easily. [27]

Testing with TypeScript is also another aspect of the language as there are many testing frameworks such as Jasmine, Mocha, Qunit, etc. As it is a product of Microsoft, there is a huge community supporting TypeScript because it is obvious that the language offers a lot and has been backed up by a huge corporate. [27]

The biggest downside of TypeScript is that it is not very suitable for small-scale applications rather the big ones, so it wouldn't be optimal to start as the first language with it. Although, TypeScript is a widely used programming language. [27]

3.7.1 Angular

Angular framework has first started to be around as AngularJS which was a JavaScript framework, but it seemed that the web applications has been moving to a direction where the complexity was growing exponentially as it the applications were doing many tasks. Angular is a programming language built by Google and it is one of the first frontend technology that was providing solutions for SPAs (Single-Page Applications). [28] SPA has been a solution for many web applications to optimize their efficient applications because the web applications has become really complex it was getting slower but since SPA era came along it was a solution for many development problems. [28] Angular has implemented SPA to its architecture. After AngularJS, Angular has completely written from ground for the new age of web applications, it is a solution for complex web applications. [28]

Angular has become a successful framework as it is providing developers custom components, two-way data binding, dependency injection and many more features. For setting up the environment of Angular application, the machine needs to have Node.js installed as it is used as a base for build environment. Of course, we need TypeScript as it is base programming language for Angular. For best practice developers should also use angular CLI, which is one of the features of Angular unlike AngularJS. Angular's structure when we serve our code separates to four [28]:

1. **Root HTML:** Root HTML is *index.html* file created within angular, the code inside the index.html is a neat code without any dependency and holds the code for what is needed to be rendered in the browser.
2. **Entry point:** This piece of code is created as *main.ts* file and it holds the code responsible for which file to be loaded when the project is started. It activates the rest of the application from this point.
3. **Main module:** This particular module is where the projects actual source code starts in an angular project and the applications components are declared in this module. We can say that the Main module is the core part of the project configuration. At creation the file is called *app.module.ts*.
4. **Root component:** The component inside an Angular project is where developers create their logic and functionality inside their code. A component in Angular consist of three files which are a .HTML, .TS and .CSS. The HTML is a template for a TypeScript file and cascading style sheets (CSS) is the local style for a component. They are connected within components inside the .TS file. We can say that components are Typescript classes that are decorated with different metadata and attributes.

Another feature that comes with angular is **life cycle hooks**, they are pre-defined interfaces that we can use inside our components. [28] We can hook our needs according to life cycle hooks. For example, if we place a function inside an *onInit* interface it means we are going to initiate that function when the component will be rendered. We can also have a life cycle *onDestroy,* inside this interface we can specify to unsubscribe to an event only when the component is destroyed meaning closed. There are eight different life cycle hooks in total. [28]

Directives is another feature that Angular is offering to developers. It allows developers to create their own functionalities for elements. [28] There are two different directives which are Attribute directives and structural directives. **Attribute directives** are usually used to change the appearance and style of an element for example we can add to an element *NgStyle* directive in which developer can add CSS styles. Structural directives are a bit different as they add or remove elements from the Document Object Model (DOM), for example, developers can add *NgIf* directive which according to logic that element will be entered into DOM or removed from DOM. [28]

The object-orientated applications can be created easily with the angular framework as it has built-in functionalities for supporting dependency injection, routing, parent-child relations, etc. Angular has special injectors called services and inside usually API calls are placed. Property and event binding is also built-in functionalities inside Angular. [28] It is very easy to create a full object-oriented program with Angular or we can use it also only for the client-side frontend. We can basically do authentication the same as we do on the server side, we can do it also on the client side. All these powerful built-in mechanisms make Angular a framework to develop object-oriented web applications. [28]

Angular also provides routing functionalities with so-called Angular route guards. We can create custom route guards when we don't want users to be able to access a Uniform Resource Locators (URL) without permission. There can be a scenario where a component inside a project is only needed to be seen by admins. Imagine we have a website called "www.angularRoute.com" and there is no button or link to access a route called "angularRoute.com/admins", the button or the link would be needed to be accessible only to admins. Any user can custom type the URL and access inside the route but if we use *canActivate* route which is used for client-side protection where it knows which users can activate that specific route. If a user is not an admin that means that they will not be able to access that specific route. [28]

All these features and there are way more than these is what makes angular a framework that provides solutions for complex web applications. [28]

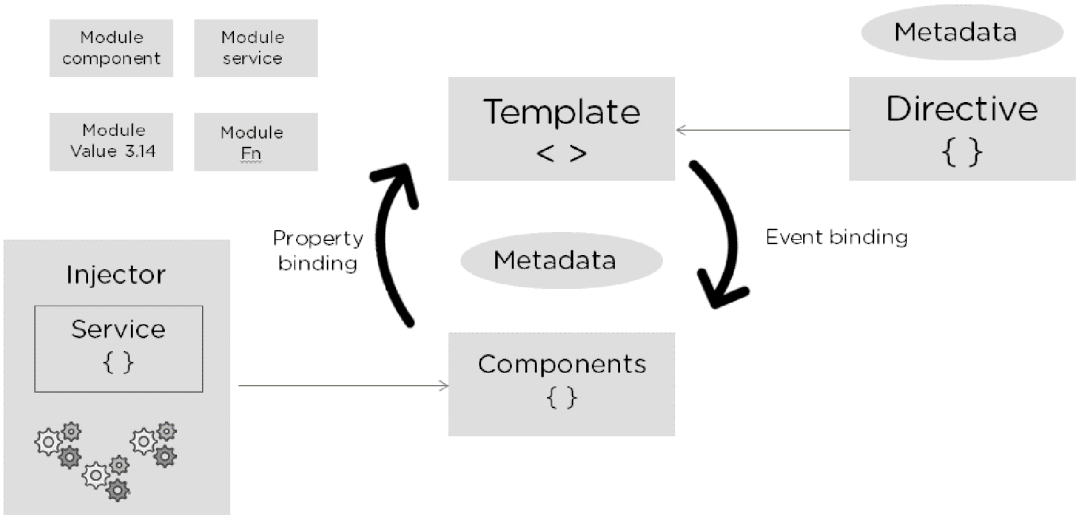


Figure 8. Angular Architecture [29]

3.7.2 Bootstrap

Bootstrap is a framework that is open source and developed for styling responsive web pages and since it is providing dynamic development it can be used on different screen sizes also. [30] Bootstrap is a framework designed to be able to work with HTML, CSS, and JavaScript. Bootstrap is providing developers fast integration on styling good frontend applications. There are many templates and components which can be used within bootstrap that is pre-defined, this way it helps developers build their applications way faster than usual. [30]

Usually, developers do the grid design on how the application would look on phones, tablets, laptops, desktops and design their application according to those grids, this way developers get a very good responsive design that can be used on different platforms and different sizes of screens no matter what. As of 2022 bootstrap has the 5th version and it is a responsive and dynamic version since its creation. That is why in last year's bootstrap has added more on itself and there are millions of developers developing their application with bootstrap as it is framework for most used styling cascade. [30]

3.7.3 Walking skeleton model

A "walking skeleton" in software development refers to a basic version of the application's structure that has the essential components connected together. [31] The goal is to have a functional system with a basic framework, but it does not have all the features

and functionality of the final product. It is essentially a starting point to build upon, with incomplete subsystems that are still connected to each other.

A walking skeleton must be able to perform the core functions that are expected of the final system. This includes tasks such as retrieving data, making requests, and performing functions. Additionally, necessary automation tools like exception handling and continuous integration should already be implemented and functional. The essential components of the system should be tested to verify they are working properly, such as checking that data is returned when making a query. After the essential components have been successfully demonstrated, the development process can start, and both the architecture and functionality of the application can be built and improved simultaneously. [31]

There are several benefits to finishing a walking skeleton before the start of the first sprint in software development [32]:

1. It showcases the structure and patterns of the architecture.
2. It verifies that the essential technologies are functioning correctly.
3. It sets the foundation for continuous integration and automated deployment.
4. It provides a shared framework for the development team to work with.

The main advantage of creating a walking skeleton is that it enables the development team to quickly concentrate on developing new product features.[32] By completing the walking skeleton before the first sprint starts, you have already confirmed that the skeleton can run in a production-like environment and have set up the necessary automation for ongoing development. This saves time and resources and helps to ensure a smooth and efficient development process.[32]

4 Practical Part

Practical part is devoted to design a web application for communication, and it is a real-life application example designed with principles and technologies described in literature review.

4.1 Analysis

The Analysis section of this thesis presents a comprehensive examination of the technical and functional requirements for a web-based messaging application. The purpose of this analysis is to evaluate the feasibility of the proposed solution, identify any potential challenges, and make recommendations for the development of the application. The methodology used in this analysis is based on a review of relevant literature, technical specifications, and user requirements The Analysis section of this thesis will first outline the technical requirements for the messaging application, including the software tools and technologies that will be used. The system design, including the class diagram and entity-relationship diagram will be described. Finally, a feasibility study will be conducted to assess the technical, economic, and schedule considerations for the development of the messaging application.

As a messaging application developed using ASP.NET, SignalR and Angular, the application leverages the power of these modern technologies to provide a seamless and intuitive user experience. With ASP.NET as the backbone, our application is able to handle large amounts of data and traffic, ensuring that messages are delivered quickly and efficiently with a secure API. The application is designed to meet the needs of today's digital communication landscape.

As the backbone of the messaging application, ASP.NET provides robust and scalable infrastructure for handling data, processing requests, and serving dynamic content to users. With its built-in security features, the application can handle sensitive information with confidence, ensuring that all user data is protected from unauthorized access. The use of SignalR in conjunction with ASP.NET allows for real-time communication between users, providing a seamless and responsive experience for sending and receiving messages. Additionally, the ASP.NET framework provides powerful tools for managing and maintaining the application, including its powerful routing capabilities and support for a

variety of data access technologies. Overall, ASP.NET offers a robust and reliable foundation for the messaging application, providing the stability and scalability needed to meet the demands of today's fast-paced digital landscape.

The goal of the application is creating an end to end application where the users can chat between other users, be able to see their real time presence inside the application. Each user can send many messages to other users. Every user can have multiple photos and they can choose which of their photo they want to have as main. Users will be displayed in different page than messages and the users page will be default home page. Users can have their profiles edited so they can change information about their self's. Users can both sign up and log in depending on whether they are already a user or not. For security reasons the application will have its own exception middleware. Application will be developed in neat, documented way following the Model-view-controller design pattern standards. Requirements of project:

- Use walking-skeleton model for designing the application
- Use OOP standards while creating application
- Users can have many pictures
- Users can select their main picture
- Users can edit their profile
- Create login and registering
- Users can see other users' presence
- Users can send a message to one user and see when they have sent
- the message receiver will see the message in Inbox or unread if they haven't read the message
- Adjust date time to be in UTC
- Users can delete their own messages
- Users are able to see whether their message has been read by the receiver
- Users can log out
- Users can see other users' profiles
- Users are able to see the unread messages, inbox and outbox
- Users will have notification when they try to close the page without saving their edits.
- Real time chat

All these requirements are fundamental for a communication application because it a social application users must have a profile and photo for the identification. Real time presence is also important for this kind of application. Having date time adjustment is another core part since users might be from different time zones even the different browsers have wide variety of date and time implementation, and it is important to create a global date and time implementation. Users must have an editable profile as they might change their location or interest etc.

4.2 Design

The proposed system is a messaging application that is developed using ASP.NET as the server-side framework and Angular as the client-side framework. This messaging application is designed to allow users to communicate with each other in real-time, exchange messages, and provide photo management.

The messaging application has a three-tier architecture, consisting of the client-side, server-side, and the database. The client-side is developed using Angular, which is a front-end framework that provides a rich user interface and interactive features. The server-side is developed using ASP.NET, which is a server-side framework that provides a secure and scalable platform to build web applications. The database is used to store user information, messages, and files.

The data flow in the messaging application starts from the client-side, where the user inputs information, such as messages and photos. The Angular framework then sends the data to the server-side, where it is processed and stored in the database. The ASP.NET framework is responsible for handling the data transfer between the client-side and the server-side and ensuring that the data is secure and protected. The server-side then retrieves the data from the database and sends it back to the client-side, where it is displayed to the user. The architecture of the application is as it follows:

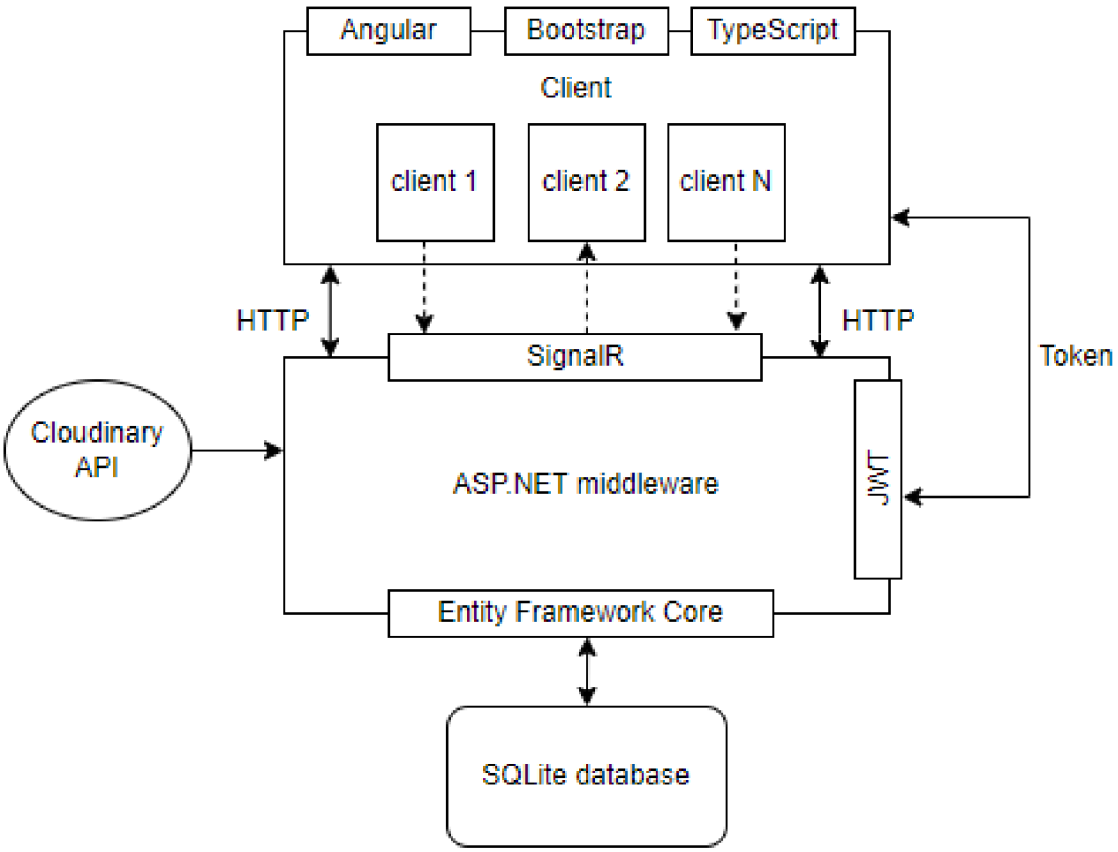


Figure 9. The architecture of application

The application has developed around five classes which are **User, Photos, Messages, SignalRGroup, UsersSignalR**.

The User class is used for authentication and information for users. The Hash and salt properties are used for encrypting the password, when user is registered the password entered is hashed and then the hashed is salted which 2 of them are encrypting algorithms. When user logs in the password sent to backend is hashed and the salt password from database is being decrypted back to hash which creates extra layer of security. The User class has ICollection properties of Photo and Message for holding the collection Photos and messages of user. The other properties are username, sex. DoB stands for (Date of birth), hobbies, city, and country in order to store users' information.

The Photo class has Adress property for the main photo of user and the main photo can be changed with property leading. Public address is used to store the photo addresses from Cloudinary API. The UserId property is used as identifying the owner of the photo.

The message class has properties such as IssuerId and IssuerUsername for identifying who is the sender of the photo and TargetId and TargetUsername is the receiver of the message. TextRead property is used for showing the user when the user read their text and TextSent is used to show the user when message was sent. Text property stores the actual message.

SignalRGroup and UserSignalR classes are used to create a group between two users so they will have real time chatting feature.

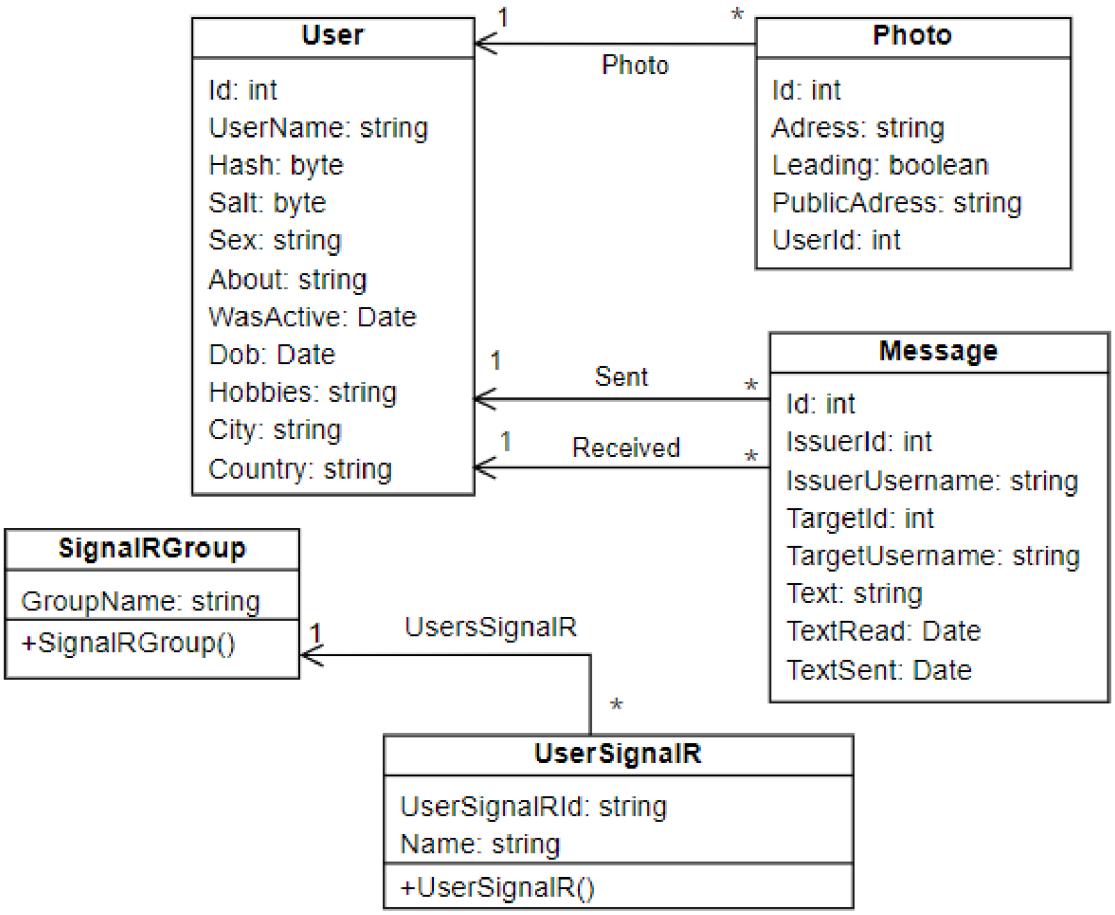


Figure 10.UML Class diagram

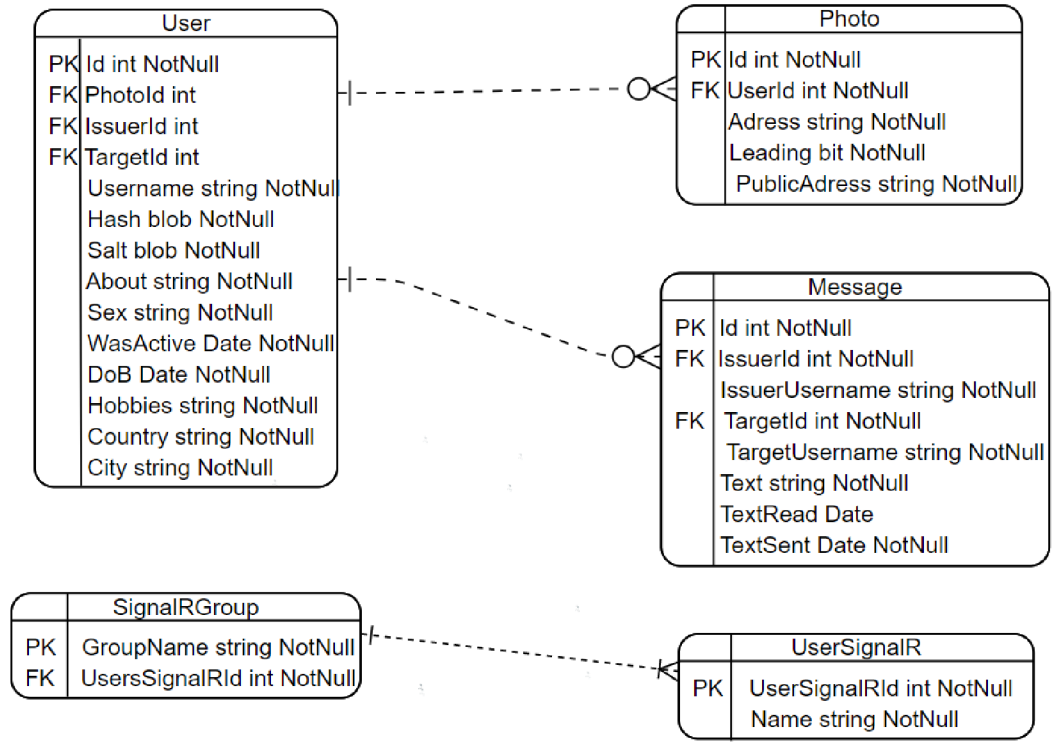


Figure 11. Entity relationship diagram

Diagram visualized as a **One-to-Many** relation between User and Photo, User and Message and **One-to-One** relation between UsersSignalR and SignalRGroup.

Entity framework core has been used for generating the code needed implementing this UML diagram into Data definition Language. These Entities has been used all around the application, the User class is the fundamental class as it gives our application a functionality needed in almost every web application. Messages and photos class has been implemented as a fundamental part for a social communication application. The UserSignalR and SignalRGroup classes has been specifically designed to implement functionality for real time communication between the users.

4.3 Implementation

4.3.1 Setting up the environment and application

The web application has been built using C# programming language 10.0 version and .NET version 6.0.400 for backend purposes. For the front end, the Angular 14th version has been used with bootstrap 5.1.3 as styling. NodeJS 16.13 is used for package management for Angular. Git 2.35.1, version control has been used for keeping track of codes and

maintaining applications without a need to change the original source code. Visual studio code used as editor for the application development.

4.3.2 CLI

.NET CLI is included with the .NET framework SDK installer, and Angular CLI, which can be installed using the npm installer which we have used 14.2.5 version, has been used to maximize efficiency while developing the application. Another useful CLI is EF CLI which is used to make entity framework transactions, we can install it using the `*dotnet tool install --global dotnet-ef*` shell command.

4.3.3 Dependent technologies

Entity framework core, JSON web tokens and SignalR has been used as external frameworks and libraries for development.

The **NuGet Gallery** is a public repository of packages for .NET. It provides a central location to discover and download packages, which includes libraries, tools, and frameworks that can be used to .NET applications. NuGet gallery is used to include libraries and the references for files are stored in .csproj file.

SQLite is a serverless database, which means that there is no separate server process that needs to be started, configured, or managed. Instead, the database is stored as a file on disk, and can be read and written to directly by the application. SQLlite has used as database provider for the application.

Cloudinary is a cloud-based platform that provides an API for managing and transforming media assets such as images, videos, and audio files. The Cloudinary API allows developers to programmatically upload, store, manage, and deliver these assets in the cloud, reducing the burden of infrastructure management. Cloudinary is used to retrieve and store the photos for users.

4.3.4 Initial set-up of project

The initiation of project can be done using any command line with CLI for .NET Framework. The template short name for ASP.NET framework MVC is `*mvc*` keyword. To initiate an MVC template we can adjust our directory in the command line and type `*dotnet new mvc*`, this way the files needed for ASP.NET MVC will be created which includes *program.cs, a .csproj* file, folder for controllers, models, and views. We can modify the names of the files and one of the most important would be the *program.cs* file as it is including the code which starts our ASP.NET application.

For setting up the visual studio code C# by omni sharp, Angular language service by Angular and angular snippets package created by John Papa, and since vs code doesn't come with packet manager for C#, NuGett gallery extensions has been installed into vs code. For developing the application in vs code these extensions are needed.

4.3.5 Walking skeleton model for backend

The application has used the walking skeleton model for building the application by connecting and installing all the parts needed for backend without all the functionalities set.

At first a class for user entity has created which holds only two properties since we are developing using walking skeleton model. After that, from NuGett gallery we can add packages to our application. All the packages used in an application is stored under .csproj. For example, because the application needs entity framework design and entity framework for SQLite you can see that from NuGett gallery after we installed those packages they appeared in .csproj file:

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>net6.0</TargetFramework>
    <ImplicitUsings>enable</ImplicitUsings>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.EntityFrameworkCore" Version="6.0.8" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.Design" Version="6.0.8">
      <IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>
      <PrivateAssets>all</PrivateAssets>
    </PackageReference>
    <PackageReference Include="Microsoft.EntityFrameworkCore.Sqlite" Version="6.0.8" />
  </ItemGroup>

</Project>
```

Figure 12. .csproj file after we installed packages

First of all before we do the initial migration inside the *appsettings.Development.json* file connection string has been configured and database assigned a name as follows:

```
"ConnectionStrings": {
  "DefaultConnection" : "Data source= messagingApp.db"
},
```

Figure 13. Connection strings configuration

After the project's connection string has configured, the first initial migration has been created with the terminal using `*dotnet ef migrations add InitialCreate*` which creates our entity as a table and assigns a primary key. For initiating the database with SQLite `*dotnet ef database update*` command has been executed in the terminal. After that, for testing reasons three records as a user has been created using SQLite and this is the reason why the user class wasn't implemented fully at first as we didn't need to spend time on creating table. Now, we can see the table in our application as:

ID	UserName
1	Vait
2	Ameti
3	Jan

Figure 14. user table after first creation

DataContext.cs class for deriving from DbContext was developed as it is a class that helps us to do basic transactions with entity framework and needs to be injected.

```
public class DataContext : DbContext
{

    public DataContext(DbContextOptions options) : base(options)
    {
    }

    public DbSet<User> Users {get; set;}
```

Figure 15. DataContext.cs file

The DbSet of user type also has added as a property inside the DataContext class. For injecting this class everywhere in our application, we need to do it from *program.cs* using *builder.Services* which is a predefined code written in ASP.NET applications since .NET 6 as we don't have *stratup.cs* file anymore. By injecting this class, the application was connected to the database and made sure that entity framework understand that the class Users is an entity and can be configured within application and this way the repository pattern also configured.

```
builder.Services.AddDbContext<DataContext>(options =>{options.
    UseSqlite(builder.Configuration.GetConnectionString("DefaultConnection"));
});
```

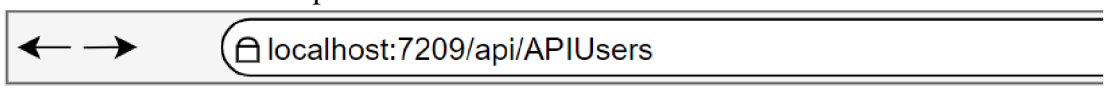
Figure 16. Injecting DataContext as Service

Last thing that was configured in walking skeleton model for backend was the user's controller. The controllers derive from ControllerBase class which gives extra attributes to be used. *ApiController* attribute is always on top of the class and under it the route is defined. Inside the controller the constructor injects DataContext to access database. Initially only two HTTP requests were made to return all users and only one user by id in the controller as follows:

```
[ApiController]
[Route("api/[controller]")]

public class APIUsersController : ControllerBase
{

    private readonly DataContext _dataContext;

    public APIUsersController(DataContext dataContext)
    {
     _dataContext = dataContext;
    }
    [HttpGet]

    public async Task<ActionResult<IEnumerable<User>>> GetAllUsers()
    {
       return await _dataContext.Users.ToListAsync();
    }
    [HttpGet("{id}")]

    public async Task<ActionResult<User>> GetOneUser(int id)
    {
        return await _dataContext.Users.FindAsync(id);
    }
```

Figure 17. APIUsersController

After the controller configuration, when we hit the *GetAllUsers* end point inside the route with the localhost port 7209 the result is:



[{"id":1,"username":"Vait"},{"id":2,"userName:"Ameti"},"id":3,"userName":"Jan"}]

Figure 18. HTTP request for getting all the users

4.3.6 Walking skeleton model for frontend

To add the angular CLI, the installment has been done through terminal using the command `*npm install -g @angular/cli*`. After that we for including angular client into our project `*ng new*` command has been used. This command creates all the necessary files in our application to be able to use Angular.

After that we need to include HTTP module to be imported as we need to make API calls. When Angular has installed, a file called *app.module* was created where we can import any module available. Below you can see how we imported HTTP module:

```
@NgModule({
  declarations: [
    AppComponent,

  ],
  imports: [
    BrowserModule,
    BrowserAnimationsModule,
    AppRoutingModule,
    HttpClientModule,

  ],
  providers: [],
  bootstrap: [AppComponent]
})

export class AppModule { }
```

Figure 19. importing HTTP client

To be able to inject API calls from .NET, CORS policy has been added in programs.cs file according to Angular's Live development server which is `*http://localhost:4200/*`:

```
app.UseCors(policy => policy.AllowAnyHeader().AllowAnyMethod()
    .AllowCredentials().WithOrigins("http://localhost:4200"));
```

Figure 20 CORS configuration

Lastly, to finish walking skeleton model bootstrap 5 was added using `*ng add ngx-bootstrap*` command and to use ready-made icons `npm install material-icons@latest` command was used. Finally, everything related to connecting application and setting up was developed with walking skeleton model.

4.3.7 Authentication and JSON web token security

The authentication to be able to communicate with frontend and backend the JWT token has been implemented. The claims have been assigned to every user with the expiration date and signing key so it can be authenticated in server. The Token class has a method to create a token which takes a parameter which is the User type. In the constructor we are generating the security key that is needed to be assigned. Inside the method first a new claim was registered with the username which is unique and used as a login parameter. The signing credentials has been settled with the security key that assigned in constructor and given an encryption algorithm. The description of token has assigned its subject, the expiry date and the Signing credentials which is used as a sign to communicate with the server. For creating the token the JwtSecurityTokenHandler class has been used as it has a method written CreateToken() which creates our token according to the parameters assigned. In the end the method has returning the token by using WriteToken() method which is also another JwtSecurityTokenHandler class method. The implementation of token is in the figure below:

```
private readonly SymmetricSecurityKey securityKey;

public Token(IConfiguration configuration)
{
    securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(configuration["TokenKey"]));
}


public string NewToken(User user)
{
    var claims = new List<Claim>{
        new Claim(JwtRegisteredClaimNames.NameId, user.UserName)
    };

    var credentials = new SigningCredentials(securityKey, SecurityAlgorithms.HmacSha512Signature);
    var description = new SecurityTokenDescriptor{
        Subject = new ClaimsIdentity(claims),
        Expires = DateTime.Now.AddDays(5),
        SigningCredentials = credentials
    };
    var handler = new JwtSecurityTokenHandler();
    var token = handler.CreateToken(description);
    return handler.WriteToken(token);
}
```

Figure 21. Token class

The application has used `*[Authorize]*` attribute on top of a controller and made it accessible to only users who has send a token to server. The controllers are always deriving

from *controllerBase* class using inheritance and the authorize attribute has been added on top of every controller that needs authentication. For example:

```
[Authorize]
[ApiController]
[Route("api/[controller]")]
0 references
public class MessageController: ControllerBase
{
```

Figure 22. Authorize attribute

4.3.8 Registering and login

The registering in application has implemented has been setup in using action result in controller that returns a user view item. The method had been designed to accept the parameter of object RegisterUser which includes properties that are needed for registering such as username, password, Date of birth, hobbies, about, country, city, and sex. The uniqueness of user has been ensured first in the application and then the registerUser object has been mapped to the user entity through autoMapper which does the mapping of objects automatically. Then the token has created and returned to user as they register.

```
[HttpPost("register")]

public async Task<ActionResult<ReturnUser>> RegisterUser(RegisterUser registerUser)
{
    if (await Existance(registerUser.UserName))
        return BadRequest("Username exists, please choose a differnet username");
    var mappedUser = Mapper.Map<User>(registerUser);
    using var encryptedPassword = new HMACSHA512();
    mappedUser.UserName = registerUser.UserName.ToUpper();
    mappedUser.Hash = encryptedPassword.ComputeHash(Encoding
        .UTF8.GetBytes(registerUser.Password));
    mappedUser.Salt = encryptedPassword.Key;
    UsersDatacontext.Users.Add(mappedUser);
    await UsersDatacontext.SaveChangesAsync();
    return new ReturnUser
    {
        Username = mappedUser.UserName,
        Token = Token.NewToken(mappedUser)
    };
}

private async Task<bool> Existance(string username)
{
    return await UsersDatacontext.Users
        .AnyAsync(user => user.UserName == username.ToUpper());
}
```

Figure 23. Register request

The login has implemented as it accepts the LoginUser object as a parameter which has properties of username and password. To see whether user is in database entity frameworks SingleOrDefaultsAsync() method has been used to get the user from database by the username as it is unique for every user. For authenticating the password HMACSHA512 algorithm has used to compute the salt password and we hash the password sent from user which will be decrypted version of salt password and we are going to return the username and token back to user so they will be authenticated. For small scale applications the author made sure that the authorization and authentication of users are scalable.

```
[HttpPost("login")]

public async Task<ActionResult<ReturnUser>> Login(LoginUser loginUser)
{
    var user = await UsersDatacontext.Users.SingleOrDefaultAsync(user =>
     user.UserName == loginUser.Username.ToUpper());
    if (user == null) return Unauthorized("The username doesnt exist!");
    using var encryptedPassword = new HMACSHA512(user.Salt);
    var decryptedPassword = encryptedPassword.ComputeHash(Encoding.UTF8.GetBytes(loginUser.Password));
    for (int a = 0; a < decryptedPassword.Length; a++)
    {
        if (decryptedPassword[a] != user.Hash[a]) return Unauthorized("wrong password");
    }
    return new ReturnUser
    {
        Username = user.UserName,
        Token = Token.NewToken(user)
    }; ;
}
```

Figure 24. login controller

4.3.9 Photo management

Photo management has been setup using Cloudinary API. Through this API the users photos had been uploaded to their database and the photos fetched through http using cloudinary API. The service for Cloudinary had used a Task of ImageUploadResult type which is a cloduinary API and as a parameter it developed to access IFormFIile object. In this method it is made sure that the photo is transformed and uploaded to Cloudinary. Configuration for Cloudinary has set up in appsettings.json file and made sure it will not be uploaded to github as it is a private information according to Cloudinaty account. To actually send the files from Cloudinary, ng2-file-upload library has been used in frontend which accepts the input as a photo and made sure that it sends the file to backend. Using Cloduinary

photo management the project has achieved its goal to add a photo, delete a photo and choose a main photo for the user object.

```
[HttpPost("newPhoto")]
0 references
public async Task<ActionResult<PhotoReturn>> AddPhoto(IFormFile file)
{
    var values = await UserDataRepository.GetByUsername(User.FindFirst(ClaimTypes.NameIdentifier)?.Value);
    var response = await PhotoService.PutNewPhoto(file);
    var image = new Photo
    {
        Adress = response.SecureUrl.AbsoluteUri,
        PublicAdress = response.PublicId,
    };

    if (values.Photo.Count == 0)
    {
        image.Leading = true;
    }
    values.Photo.Add(image);
    if (await UserDataRepository.SaveChanges())
    {
        return CreatedAtRoute("GetUser", new { username = values.UserName }, Mapper.Map<PhotoReturn>(image));
    }

    return BadRequest("cannot add this photo!");
}
```

Figure 25. Adding new Photo endpoint

4.3.10 Repository management

To separate the database access of application the repository design pattern has implemented and made sure that the database queries have been separated from controllers. For example, userDataRepository has created Task of types related which Gets all the users, get user by Id, get user by username, profile update and save changes. Entity framework methods to fetch data from DataContext has been used for all individual tasks. `*Include*` method has used to make sure when users are returned also their collection of photos will be returned. `*FindAsync*` method was used to get an int value of id and return it, since the id has configured to be unique only one user would be returned. `*SingleOrDefaultAsync*` method was used to either get one user with username or return null if no user with that username exists, this method used because the username also configured to be unique throughout the development. EntryState class was used to be able to update the users as it tracks whether the object provided has been modified or not. `*SaveChangesAsync*` method was used to return a true or false according if the dataContext had been saved different values

and this was ensured by using the grater than operator and setting the left side to zero as if any change has made or not.

```
private readonly DataContext DataContext;
0 references
public UserDataRepository(DataContext dataContext)
{
    DataContext = dataContext;
}
1 reference
public async Task<IEnumerable<User>> GetAllUsers()
{
    return await DataContext.Users.Include(e => e.Photo).ToListAsync();
}
10 references
public async Task<User> GetByUsername(string username)
{
    return await DataContext.Users.Include(e => e.Photo)
        .SingleOrDefaultAsync( e => e.UserName == username.ToUpper());
}
1 reference
public void ProfileUpdate(User user)
{
    DataContext.Entry(user).State = EntityState.Modified;
}
5 references
public async Task<bool> SaveChanges()
{
    return await DataContext.SaveChangesAsync() > 0;
}
```

Figure 26. user repository

4.3.11 Messaging between users

The messaging of users has been implemented using SignalR library for making sure the real presence and real time message send and receive. For SignalR instead of controllers, hubs were created as it is not using Http but web sockets and since it is a singleton it has implemented as a service inside the program.cs file On connected async methods were used to make sure when users are online and when they send a message update the message component without re-rendering whole application. The groups has created according to SignalR standards to make conversation between users. The method's called inside onConnected async has been developed in same class. The method for new message designed to accept a parameter which is NewMessageViewItem type which is a Message entity type. The method developed to get the sender username and the receiver username to save it in database as a new message. For the convention of signalR, the group names has created which has to have unique for each user talking between that is why it is constructed as the name of issuer username and target user name. The messages has sent to the repository in order to be added to database.

```
public async Task MessageWithUser(NewMessageViewItem newMessageViewItem)
{
    var name = Context.User.FindFirst(ClaimTypes.NameIdentifier)?.Value;
    if(name == newMessageViewItem.TargetUsername) throw new HubException("Cant message to yourself");
    var issuer = await UserDataRepository.GetByUsername(name);
    var target  = await UserDataRepository.GetByUsername(newMessageViewItem.TargetUsername);
    if(target == null) throw new HubException("Target user couldn't found");
    var newMessage = new Message
    {
        Issuer = issuer,
        Target = target,
        IssuerUsername = issuer.UserName,
        TargetUsername = target.UserName,
        Text = newMessageViewItem.Text
    };
    var chatName = ChatName(issuer.UserName, target.UserName);
    var chat = await MessageDataRepository.AddNewSignalRGroup(chatName);
    if(chat.UsersSignalR.Any(p => p.Name == target.UserName))
    {
        newMessage.TextRead = DateTime.UtcNow;
    }
    MessageDataRepository.newMessage(newMessage);
    if(await MessageDataRepository.Save()){

        await Clients.Group(chatName).SendAsync("NewChat", Mapper.Map<MessageViewItem>(newMessage));
    }
}
```

Figure 27. Message with signalR

4.3.12 Custom Middleware for error check

For error handling, a class has been created to give detailed view for an error such as httpCode, error and detailed message of error.

```
public class ApiErrors
{

    public ApiErrors(int httpCode, string error = null, string detailedMessage = null)
    {
        HttpCode = httpCode;
        Error = error;
        DetailedMessage = detailedMessage;
    }

    public int HttpCode { get; set; }

    public string Error { get; set; }

    public string DetailedMessage { get; set; }
}
```

Figure 28. Error message class

The Custom middleware designed to check every http request to make sure if there is an error, print it to console. The class Custom middleware in its constructor has implemented RequestDelegate to access the http calls and ILogger has implemented to be able to log error to the console. The InvokeAsync() method used as convention for middleware and the method's parameter designed to accept HttpContext object. Try catch statement used to catch all the errors. Inside try block the Requestdelegate was called, and the parameter passed inside. Inside catch block the logger was used to log the exception and http context was configured to get detailed view of the possible errors.

After logging the error, the middleware sets the HTTP response content type to "application/json" and the status code to 500 (Internal Server Error). It then creates an instance of the ApiErrors class, which is a custom class used to format error messages in a standardized way. The ApiErrors constructor takes two parameters: the HTTP status code and a message string. The middleware then serializes the ApiErrors instance to JSON using the JsonSerializer.Serialize method, and writes the resulting JSON string to the HTTP response stream using the httpContext.Response.WriteAsync method.

```
private readonly RequestDelegate RequestDelegate;

private readonly ILogger<CustomMiddleware> Logger;

public CustomMiddleware(RequestDelegate requestDelegate
    , ILogger<CustomMiddleware> logger)
{
    Logger = logger;
    RequestDelegate = requestDelegate;
}

public async Task InvokeAsync(HttpContext httpContext)
{
    try
    {
        await RequestDelegate(httpContext);
    }
    catch(Exception exception)
    {
        Logger.LogError(exception, exception.Message);
        httpContext.Response.ContentType = "application/json";
        httpContext.Response.StatusCode = (int) HttpStatusCode.InternalServerError;
        var resp = new ApiErrors(httpContext.Response.StatusCode,"internal server error");
        var serialization =
            new JsonSerializerOptions{PropertyNamingPolicy = JsonNamingPolicy.CamelCase};
        var result = JsonSerializer.Serialize(resp, serialization);
        await httpContext.Response.WriteAsync(result);
    }
}
```

Figure 29. Custom error middleware

The middleware was added to program.cs as `*app.UseMiddlware*` *to let the application handle the class as a middleware.*

4.4 Client Implementation

This part is going to show an overview of the end product with the summary of code. The images are from working application.

4.4.1 Home and register

The frontend implementation of authentication the angular application has been made using the **AuthGuard** canActivate router helper which only lets a user view certain page if they have the token and username sent from backend. The loginService of angular has the route for the API that will fetch the username and Bearer token to the client. The userValues observable will pipe and then map the properties. This way it is made sure that users that are not authenticated are not authorized to change URL and proceed to the section that needs authentication. The authentication guard has setup to check the authentication as follows:

```
canActivate(): Observable<boolean> {
  1 reference
  return this.loginService.userValues$.pipe(map(u =>{if(u) return true}));
}
```

Figure 30. AuthGuard

To make use of this guard the application has implemented it in the router class of angular as follows to see the users only authenticated users can activate that path:

```
{path: 'users', component: AllUsersComponent, canActivate: [AuthenticatedGuard]},
```

Figure 31. router.app

The navbar component has used as an input area for login and password which gets values from template with angular form and double binds the values which are used in component to connect the service. Local storage has been used to keep the user values for authorization throughout the lifetime of application until the user logouts.

```
loginUser(logUser: ClientUser){

  return this.httpRequest.post(this.URL + 'register/login' , logUser).pipe(map((
      serverResponse: any) => {
    const response = serverResponse;
    if(response){
      localStorage.setItem('user_access', JSON.stringify(response))
      this.userValues.next(response);
      this.onlineService.onlineConnection(response);
    }
  }))
}
1 reference
setUserValues(clientUser: ClientUser){
  localStorage.setItem('user_access', JSON.stringify(clientUser))
  this.userValues.next(clientUser)
}
```

Figure 32. login Service

```
loginUser(){
  this.loginService.loginUser(this.loginValues)
  .subscribe({next: r => {this.router.navigateByUrl('/users');
   this.userLogged = true}, error: err =>
   this.toastr.error(err.error)})
}
```

Figure 33. login component

```
<form *ngIf="!userLogged" #logUser="ngForm"
(ngSubmit)="loginUser()"
class="navbar-form navbar-left d-flex me-4">
  <input name="username" [(ngModel)]="loginValues.username"
  class="form-control me-1 border-warning" type="text"
    placeholder="username">
  <input name="password" [(ngModel)]="loginValues.password"
  class="form-control me-sm-2 border-warning"
    type="password" placeholder="password">
  <button class="btn btn-outline-warning my-2 my-sm-0"
  type="submit">Login</button>
</form>
```

Figure 34. login template

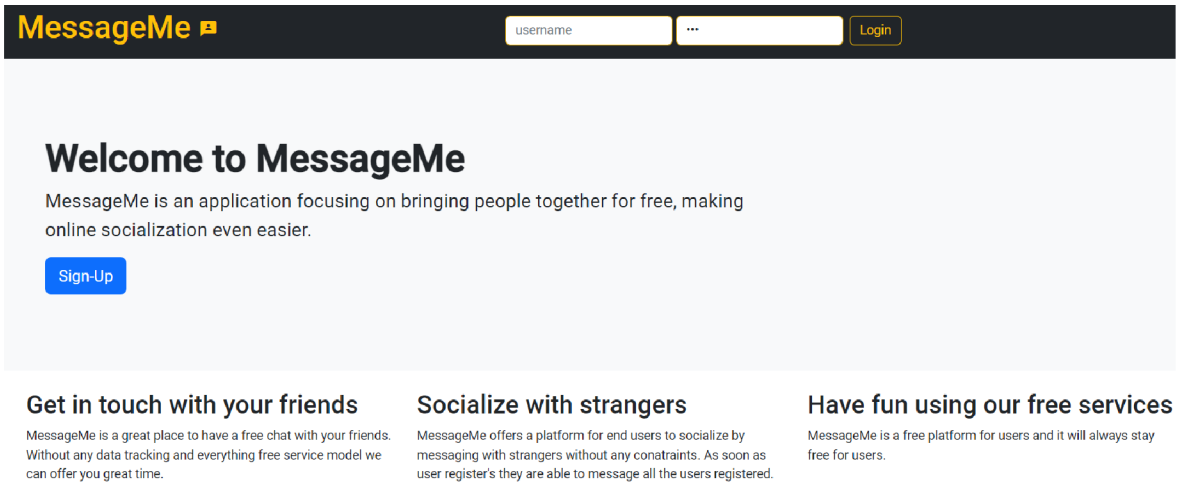


Figure 35. Home page

Angular services have been used to communicate with controller from backend as the services in angular makes the http requests to backend. The components have been used to access the angular services. To register a user angular form has been used inside the template of the angular component. Angular form has designed to have the inputs needed for the object for the controller to effectively communicate with the backend. Inside the component the attributes for template have given and made sure that the angular service is called which will communicate with the controller. This functionality of application from frontend has been achieved as follows:

```
register(clientUser: any){
  return this.httpRequest.post(this.URL+ 'register/register', clientUser)

  .pipe(map((u:ClientUser)=>{if(u){localStorage.setItem('user_access',
      JSON.stringify(u));
 this.userValues.next(u);
 this.onlineService.onlineConnection(u);
}} ))
}
```

Figure 36. register service

```
signUp(registerForm: NgForm){
  this.loginService.register(registerForm?.value).subscribe({next: r =>
    {this.router.navigateByUrl('/users');
    console.log(r)
  },  error: err => console.log(err)})
}
```

Figure 37. register component

```
<div class="form-group">
    <label class="control-label col-sm-2" for="userName">Username:</label>
    <div class="col-sm-10 w-25">
        <input #userName="ngModel" id="username" required type="text"
        class="form-control" placeholder="Enter username" name="userName"
            [(ngModel)]="formValues.username">
            <div  *ngIf="userName.invalid && (userName.dirty || userName.touched)"
            class="alert alert-danger ">
                <div *ngIf="userName.errors?.['required']">
                    Must have a username!
                </div>
             </div>
    </div>
</div>
```

Figure 38. register form

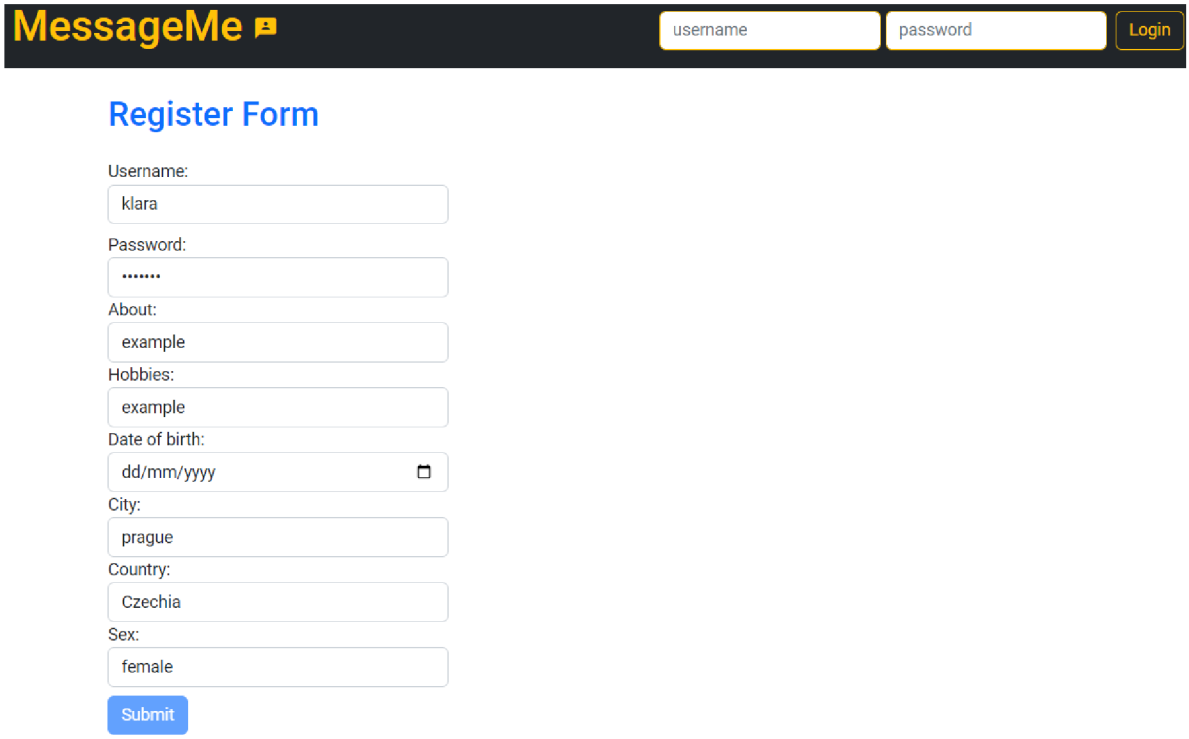


Figure 39.register page

4.4.2 Users Page

The users page displays all the users registered within application. Users can see how their own user card looks and they can directly edit their page from the button. The online users are displaying an icon in green, and the offline users are displaying their icon in black. Every other user has two buttons attached and they can either go to their detail page or they can directly take you into messaging section for the user. This page comes right after a user register or logs in as they are redirected here and the login button disappears. The navbar

displays and greets the user where they can also edit their profile or log out from their profile and three tabs are displayed. The components retrieve data from backend using a HTTP request as an observable through angular service.

```
getMappedUser(username : string) : Observable<MappedUsers>{
  return this.httpRequest.get<MappedUsers>(this.URL + 'APIUsers/' + username);
}
```

Figure 40. Get users

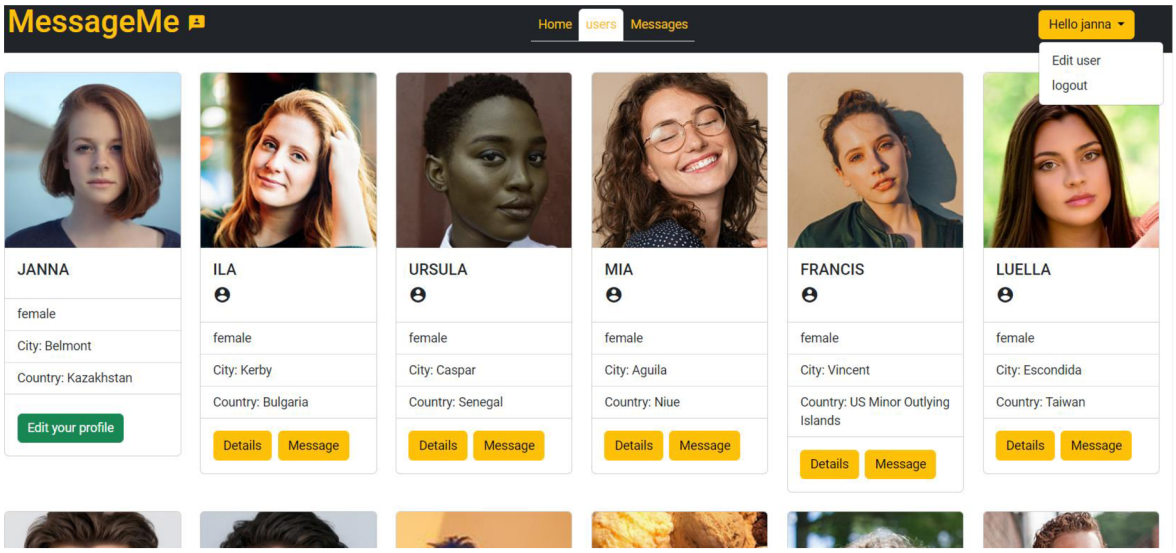


Figure 41. Users page

4.4.3 Users detail page

Users detail page gives a detailed view about the user selected and indicates a small icon whether they are online or offline. The users photo gallery can have more than one photo and they can swipe to other photos. The chat tab is where user can send message to the user they are viewing. The components get specific user according to their username and subscribes to the returned user and retrieves the user photos and messages if there are any and creates a group using signal R.

```
getUser() {
  1 reference
  this.mappedUsersService.getMappedUser(this.activatedRoute.snapshot.paramMap.get('username')).subscribe(e => {
    this.user = e;
    if (this.user) {
      this.userPhotos();
      this.messagesBetweenUsers();
      this.textService.onlineConnection(this.clientUser, this.user.userName);
    }
  })
}
```

Figure 42. Getting user detail

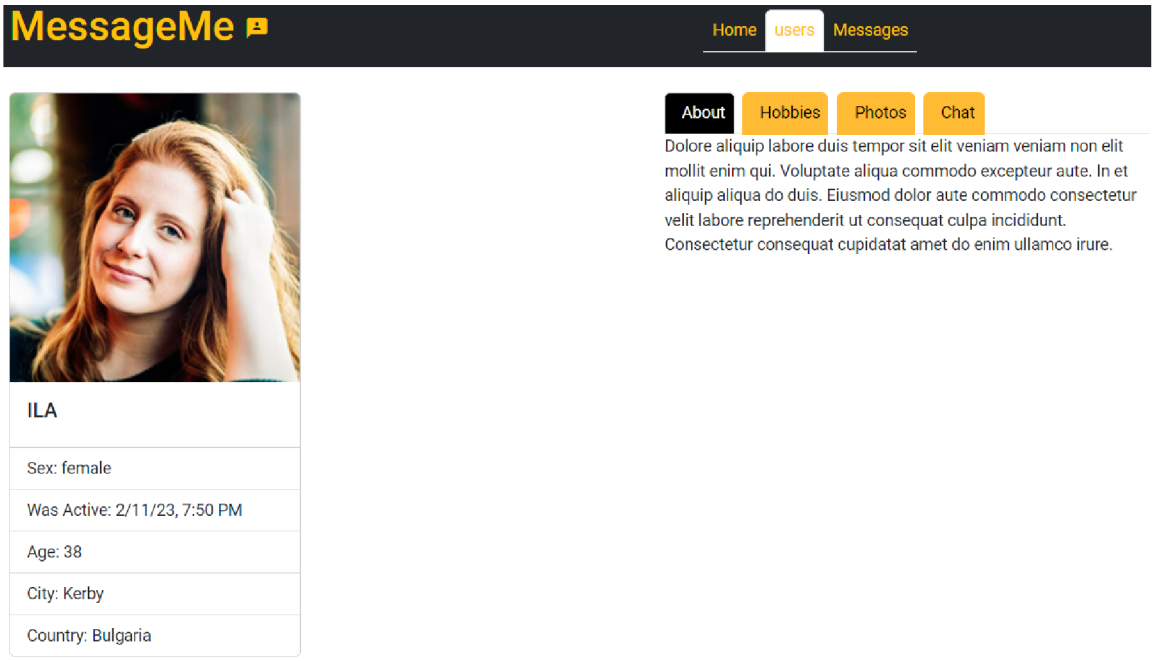


Figure 43.User detail page

4.4.4 Users edit profile

Users can edit their profile and update their country, city, hobbies and about properties using angular form. When user has more than one photo, they are able to change it as main photo or delete the photo that is not the main photo. The photo is sent to backend with the user's token over the API URL.

```
updateMember(){
  0 references
  this.mappedUserService.updateMappedUser(this.mappedMember).subscribe(()=>{
    this.editForm.reset(this.mappedMember);
    this.isForm = true;
  })
}
```

Figure 44. Updating user properties

```
Upload(){
  this.uploadConfig = new FileUploader({
    url: this.URL + 'APIUsers/newPhoto',
    authToken: 'Bearer ' + this.clientUser.token,
    isHTML5: true,
    allowedFileType: ['image'],
    removeAfterUpload: true,
    autoUpload: false,
    maxFileSize: 10 * 1024 * 1024
  })
```

Figure 45. Uploading photo

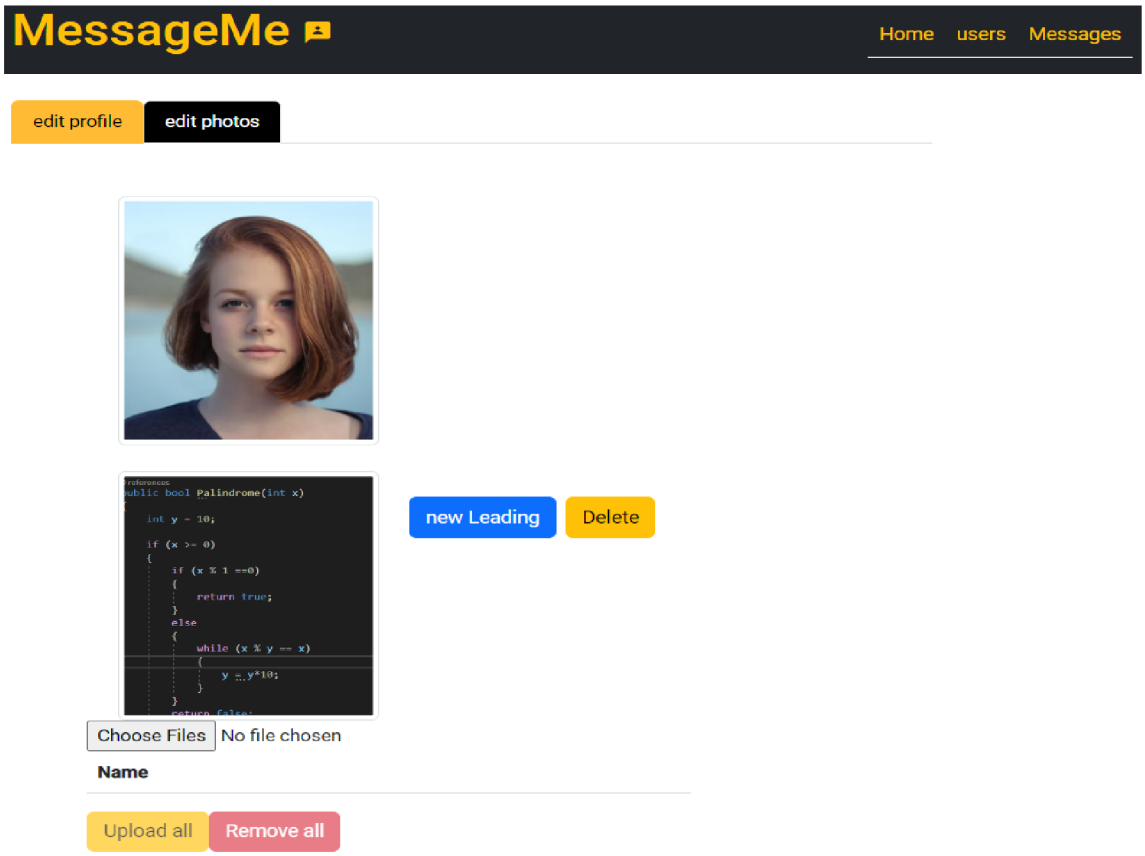


Figure 46. user edit page

4.4.5 Users message

The SignalR library was installed for the frontend with '*install @microsoft/signalr*'. On connected async methods were used to make sure when users are online and when they send a message update the message component without re-rendering whole application. In the message detail section users can send new message and the messages has different styles according to whom written the message. Users can see the date and time the message has sent or received. If the message belongs to the user, it indicates whether the message they sent has been read or not. The order of the messages is handled by using angular structural directives which shows the messages on correct sides whether it is sent or received. The online connection has been setup and target the HubConnectionBuilder() method to access the method on figure 31. The query parameters have sent with the name of the user to message with between individual users. The SignalR used as a fundamental technology for a communicating social application instead of HTTP because, if HTTP had been used then it would be impossible to get messages without rendering the webpage. To access the

websocket user token has sent from the frontend. Angular structural directives were implemented inside the template to show every message in an order and separated from which message is from sender and which is from receiver. `*NgFor*` directive has used to iterate all messages and `NgIf` directive used to separate design of how the message will look. The bootstrap classes were used to design the structure of messages to be separated from sender and receiver. The messages had made sure that includes the time it has been sent and designed to show as `*M/d/yy, h:mm a*` format with angular datePipe property which configured to be short and gave the format wanted. Read and unread properties have added only to the messages that are on the other side of user than the current user since those two properties are essential for a messaging application and angular structural directives was used to handle this function. Inside the component it is ensured that the signalR websocket will run if the component is not destroyed. This way it is ensured that the messages send from one user it to other user they will see the messages in real time as long as they are running the component.

```
<div *ngFor="let item of (textService.clientText$ | async)">
  <div *ngIf="item.targetUsername === user?.userName" class="d-flex flex-row justify-content-end">
    <div>
      <p class="medium p-2 ms-3 mb-1 rounded-3 bg-warning">{{item.text}}</p>
      <p class="small ms-3 mb-3 rounded-3 text-muted">{{item.textSent | date: 'short'}}
        <span class="text-success" *ngIf="item.textRead">*Read</span>
        <span class="text-danger" *ngIf="!item.textRead">*Unread</span>
      </p>
    </div>
  </div>
  <div *ngIf="item.issuerUsername === user?.userName" class="d-flex flex-row justify-content-start">
    <div>
      <p class="medium p-2 me-3 mb-1 text-white rounded-3 bg-dark">{{item.text}}</p>
      <p class="small me-3 mb-3 rounded-3 text-muted d-flex justify-content-end">{{item.textSent |
        date: 'short'}}</p>
    </div>
  </div>
</div>
```

Figure 47. Message template

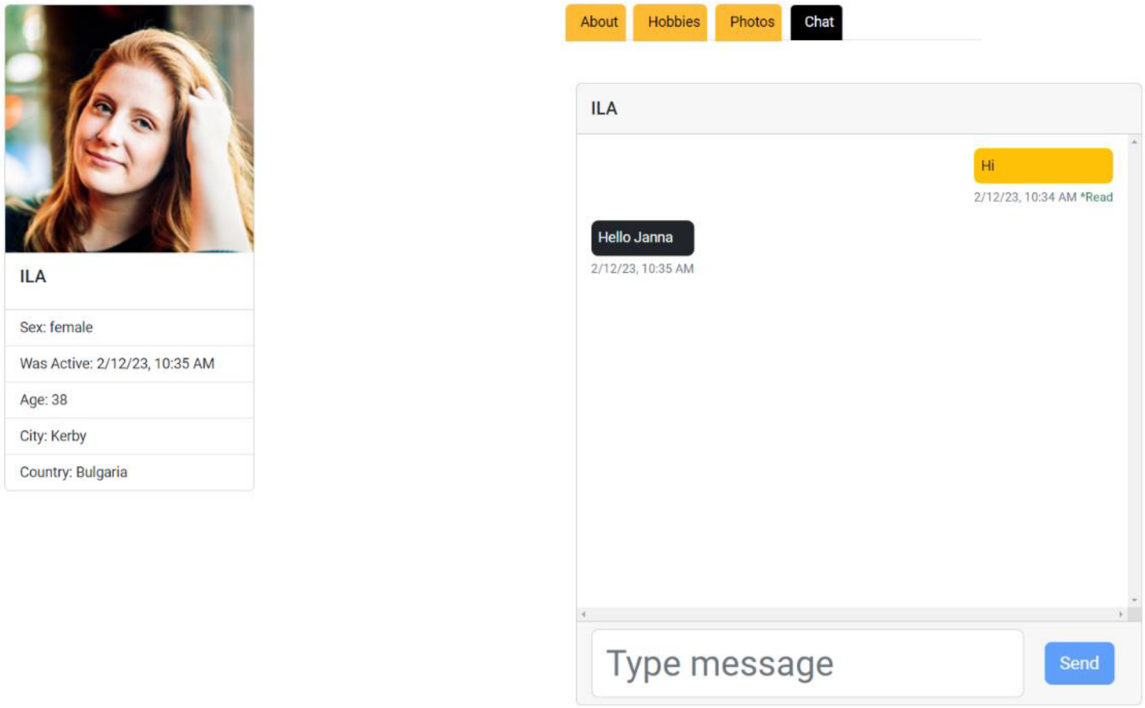


Figure 48.user chat tab

4.4.6 Messages

In the message section users can cruise through Unread, Inbox and Outbox with information related to the message. The user can click the row of any message and they are redirected to the actual user messages. The user can unsend their message by pressing the delete button on their outbox. The Unread, Inbox and outbox messages are sorted out by using QueryParameters. When user selects the outbox messages for example, the angular will do an API call with query parameters which then the data from backend will send only outbox messages. By default the unread messages will be displayed when the component has stars and once user clicks on the message it will take them to user detail component where user can see its chat history and the message will be displayed inside the inbox.

```
<div *ngIf="container === 'Outbox' ">
    <table class="table table-warning table-striped mt-4" style="cursor: pointer;">
        <thead>
            <tr>
                <th style="width: 50%;">Message</th>
                <th style="width: 20%;">To</th>
                <th style="width: 20%;">Time</th>
                <th style="width: 10%;">State</th>
                <th></th>
            </tr>
        </thead>
        <tbody>
            <tr *ngFor="let item of texts" routerLink="/users/{{item.targetUsername}}" [queryParams]='{tab:3}'>
                <td>{{item?.text}}</td>
                <td>{{item?.targetUsername | lowercase}}</td>
                <td>{{item?.textSent | date:'short'}}</td>
                <td>Sent</td>
                <td><button (click)="deleteText(item?.id)" class="btn btn-danger me-4">Delete</button></td>
            </tr>
        </tbody>
    </table>
</div>
```

Figure 49.Outbox section of messages

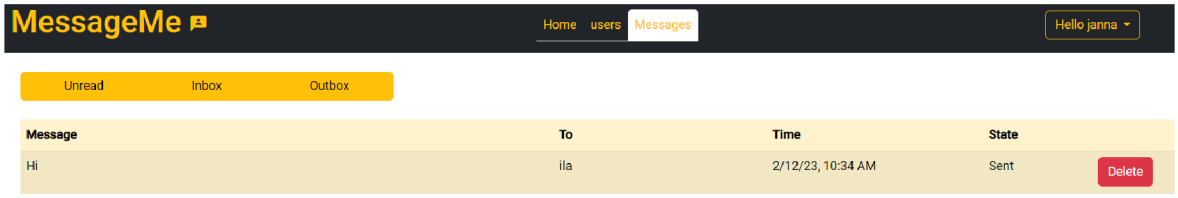


Figure 50.User outbox messages page

5 Results and Discussion

A functional messaging web application with a secure and reliable backend that effectively communicates with the frontend through the API was created. Users are able to create their profiles, edit their attributes, and communicate with other users through messaging. The frontend designed in a way that provides an efficient and user-friendly experience. The basic user interface was designed with frequently used components placed in the accessible areas of the web browser, allowing users to easily find what they need. Furthermore, the use of the MVC template and the object-oriented approach allowed us to keep our code organized and easy to maintain. This made it easier to add new features and fix any issues that arose during development. The use of Microsoft Entity Framework also made it easier to handle the data and database, reducing the amount of manual coding required. Additionally, the use of JSON Web Tokens provided a secure and efficient way to transmit data between the backend and the frontend. Overall, the combination of these technologies and approaches helped us create a robust and reliable web application that meets the requirements of modern web users. The final product can be seen in client implementation section, see figure 34, figure 35 , figure 40, figure 42, figure 45, figure 47, figure 49.

However, there are also some limitations and areas for improvement in the development of this messaging web application. The application could benefit from more advanced features such as video chat and file sharing. In addition, the security of the application could be further improved by implementing more advanced authentication and authorization methods. Overall, this thesis provides a foundation for the development of a messaging web application with a secure and reliable backend and an efficient and user-friendly frontend. Further improvements and enhancements can be made based on the specific needs of the application and its users.

6 Conclusion

The objective of this thesis was to create a messaging web application that allows users to log in, create and edit their profiles, communicate with other users, and share their attributes such as password, username, and photo. Another objective of the thesis was to create a web application with a secure and maintainable backend. The frontend used also to have usability between users. The application provides real time messaging and photo management. The application created is the initial creation for a web application and can be improved in many ways which is every web application need. Every web application created are having updates almost one or two time a year at least but providing the initial code that can be usable is the hard part.

C# programming language with its features used to provide a secure backend, Angular framework used to provide usability for users and entity framework used to create functionality for the application to easily migrate to another database provider as well as to communicate with the SQLite database. The MVC design pattern has provided the application to be developed in less abstract way and created an easy folder management. The repository design pattern used to keep the actual logic for application different than the data fetching from database. SignalR was used to create the real-time text messaging feature. It also provided the functionality to be able to see the user's real-time presence. The Cloudinary API was used for storing, retrieving, and deleting the photos for users and their profile photos.

In conclusion, the messaging web application was developed with ASP.NET and SQLlite as backend and Angular framework was used to create the client side of the application.

7 References

[1] Damjan (2022, March 29). messaging apps statistics for 2022. Statista. [online] https://kommandotech.com/statistics/messaging-apps-statistics

[2] Iqbal, M (2022, August 31). WhatsApp revenue and usage statistics (2022) [online] https://www.businessofapps.com/data/whatsapp-statistics/

[3] Larson, G. W. (2016, July 14). instant messaging. Encyclopedia Britannica. [online] https://www.britannica.com/topic/instant-messaging

[4] CRAIG, Iain D. and SpringerLink (ONLINE). Object-Oriented Programming Languages: Interpretation. London: Springer London, 2007. ISBN 9781846287749;184628774X;1846287731;9781846287732;

[5] Anderson R. and Smith S.(2022, August 19) ASP.NET Core Middleware [online] https://learn.microsoft.com/en-us/aspnet/core/fundamentals/middleware/?view=aspnetcore-6.0

[6] Anderson R. and Smith and Hasan F. (2022, June 03) Write custom ASP.NET core middleware [online] https://learn.microsoft.com/en-us/aspnet/core/fundamentals/middleware/write?view=aspnetcore-6.0

[7] Middleware in ASP.NET Core [online] https://wakeupandcode.com/middleware-in-asp-net-core/

[8] Thompson, B (2022, August 25). What is .NET Framework? [online] https://www.guru99.com/net-framework.html

[9] MÖSSENBÖCK, Hanspeter. NET application development: with C #, ASP.NET, ADO.NET and web services. Harlow: Addison-Wesley, 2004.

[10] A diagram of .NET Framework Architecture with ASP.NET displayed. [online] https://www.researchgate.net/figure/A-diagram-of-NET-Framework-Architecture-with-ASPNET-displayed_fig2_325534947

[11] NAKOV, Svetlin and KOLEV, Veselin. Fundamentals of Computer Programming with c#: Programming Principles, Object-Oriented Programming, Data Structures. Faber Publishing ,2014.

[12] Payne, J (2021, May 5). Benefits of C#. [online] https://www.codeguru.com/csharp/benefits-of-c/

[13] JAMRO, Marcin. C# Data Structures and Algorithms: Explore the possibilities of C# for developing a variety of efficient applications. Birmingham: Packt Publishing, Limited, 2018.

[14] POLAT, Engin and Stephane BELKHERAZ. ASP.NET Core MVC 2.0 cookbook: effective ways to build modern, interactive web applications with ASP.NET Core MVC 2.0. 1st. ed. PACKT Publishing, 2018. 10.

[15] Introduction to ASP.NET. [online] https://www.geeksforgeeks.org/introduction-to-asp-net/

[16] SMITH, Jon P. and Julia LERMAN. Entity Framework Core in action. Second. ed. Shelter Island: Manning, 2021. ISBN 9781617298363;1617298360;

[17] object relational mapper. [online] https://softuni.org/dev-concepts/object-relational-mapping-orm/

[18] Aguilar M. Jose. SignalR Programming in Microsoft ASP.NET. Microsoft Press, 2014. ISBN: 978-0-7356-8388-4

[19] Peyrott S. The JWT Handbook. Auth0 Inc. 2016-2018 [online]
https://auth0.com/resources/ebooks/jwt-handbook

[20] JWT structure. [online]
https://medium.com/batc/jwt-for-dummies-ok-not-100-dummies-1f08d3279a0b

[21] FLANDERS, Jon. RESTful.NET. Sebastopol: O'Reilly, 2009. ISBN 0596519206;9780596519209;

[22] Pedamkar, P (2021) What is MVC Design Pattern. [online]
https://www.educba.com/what-is-mvc-design-pattern/

[23] Smith, S (26, June 2022) Overview of ASP.NET Core MVC. [online]
https://learn.microsoft.com/en-us/aspnet/core/mvc/overview?WT.mc_id=dotnet-35129-website&view=aspnetcore-6.0

[24] MVC Architecture. [online]
https://code-maze.com/asp-net-core-mvc-series/

[25] Shekhawat. S (2020, December 2) CRUD using the Repository Pattern in MVC. [online]
https://www.c-sharpcorner.com/UploadFile/3d39b4/crud-using-the-repository-pattern-in-mvc/

[26] Overview of Entity Framework[online]
https://www.c-sharpcorner.com/UploadFile/3d39b4/crud-using-the-repository-pattern-in-mvc/

[27] Fenton, S., 2018. Pro TypeScript. Berkeley, CA: Apress.

[28] Seshadri, S . Angular Up and Running. O'Reilly Media, 2018. ISBN 9781491999837

[29] Angular Architecture. [online]
https://www.simplilearn.com/tutorials/angular-tutorial/what-is-angular

[30] Zola A (30, August 2022) Bootstrap. [online]
https://www.techtarget.com/whatis/definition/bootstrap

[31] TechTarget Contributor (17, October 2016) Walking skeleton. [online]
https://www.techtarget.com/whatis/definition/walking-skeleton

[32] Dave Todaro(2, January 2020) Using A 'Walking Skeleton' To Reduce Risk In Software Innovation. [online]
https://www.forbes.com/sites/forbestechcouncil/2020/01/02/using-a-walking-skeleton-to-reduce-risk-in-software-innovation/?sh=5c09cd003b1c