TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Single-page Aplikace sloužící pro komunikaci mezi uživateli

Single-page application designed for communication between users

Studijní program: B2646/Informační technologie

Studijní obor: 1802R007/Informační technologie

Autor práce: Michal Červinka

Vedoucí práce: Ing. Jan Hybš

Anotace

Tato práce se zabývá zprovozněním multiplatformní webové SPA (Single-page aplikace) sloužící pro okamžitou komunikaci mezi uživateli. Projekt využívá architekturu server/klient, kde server, realizovaný pomocí frameworku ASP.NET, poskytuje API, které klient využívá. Komunikace v reálném čase je zajištěna pomocí webových socketů.

Annotation

The goal of this project is to develop a responsive cross-platform web SPA (Single Page Application) designed for an instant communication between users. This project uses a server/client architecture, where the server, realized by ASP.NET framework, provides an API, which the client uses. The real-time communication is provided by web sockets.

Prohlášení

Byl jsem seznámen s tím, že na mou ročníkovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé ročníkové práce pro vnitřní potřebu TUL.

Užiji-li ročníkovou práci nebo poskytnu-li licenci k jejímu vy-užití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mě požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do je-jich skutečné výše.

Ročníkovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé ročníkové práce a konzultantem.

V Liberci dne …….

Podpis: ……

Poděkování

Pan inženýr Jan Hybš byl díky svým zkušenostem a radám skvělý vedoucí této ročníkové práce a rád bych mu tímto způsobem poděkoval za jeho ochotu a trpělivost.

Obsah

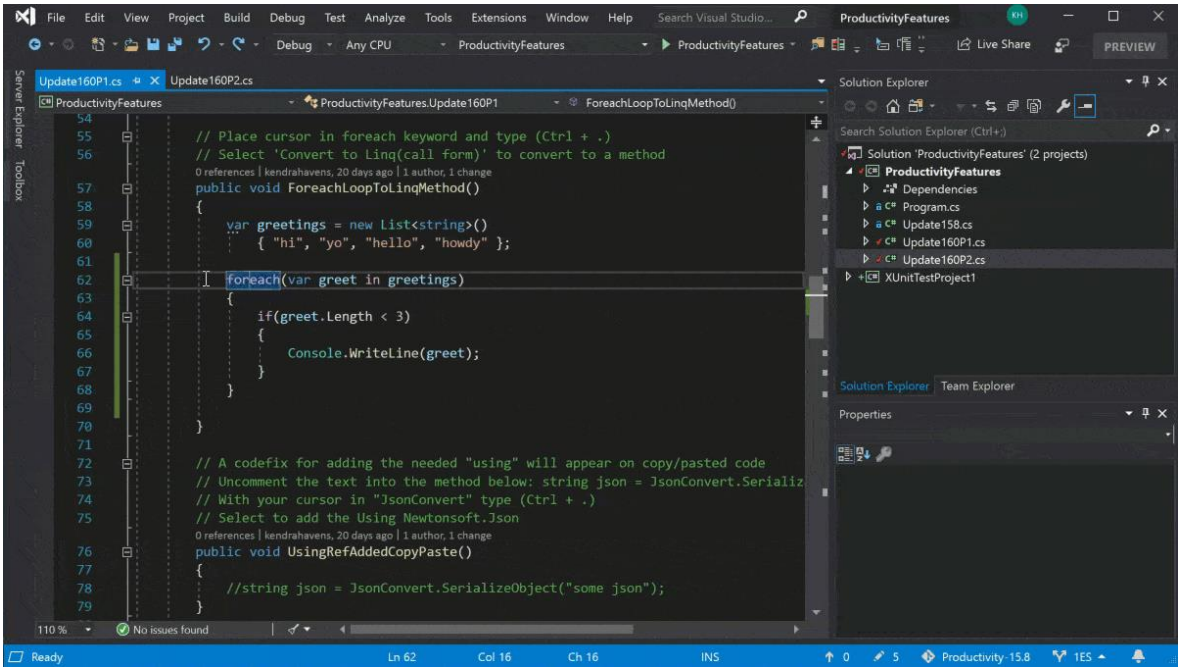
Úvod

Tato práce popisuje proces vývoje aplikace pro okamžité posílání zpráv mezi uživateli přes internet. Jak už ze zadání vyplývá, tato aplikace je vyvinuta pomocí frameworku ASP.NET, který je pro vývoj aplikace s takovými požadavky jeden z nejlepších kandidátů. Výsledkem tohoto projektu je single-page aplikace (aplikace, která mění obsah svých stránek bez potřeby znovu načtení), která umožňuje přihlásit již existujícího uživatele, zaregistrovat nového uživatele, odhlásit uživatele a pro každého uživatele nabídnout možnost napsat okamžitou zprávu jakémukoliv jinému uživateli. Z těchto požadavků nepřímo vyplývají další požadavky jako např. Bezpečné ukládání hesel, nepřihlášený uživatel nemůže nikomu napsat zprávu, vytvoření intuitivního uživatelského rozhraní pochopitelné i pro technicky nezaloženého uživatele. Součástí práce je i databáze, která se nachází na školním serveru a obsahuje informace o všech založených účtech a o všech zaslaných zprávách. V neposlední řadě se tato práce také zabývá všemi technologiemi, které byli použity pro vývoj aplikace splňující všechny dané požadavky ze zadání. Konkrétně jde o technologie Visual Studio, SQL Server, ASP.NET, MVC, SignalR, WebSocket, SHA-256 a Entity Framework. Dále se tato zpráva zabývá popisem uživatelského rozhraní aplikace. Je zde popsána struktura databáze a organizace kódu. Je zde také popsána metodika systému a také je tu popsán postup vývoje aplikace od začátku až do konce. Motivací tohoto projektu bylo získání zkušeností ohledně práce s ASP.NET frameworkem, což se v dnešní době dá brát jako užitečná zkušenost.

1. Použité technologie

1.1. Microsoft Visual Studio

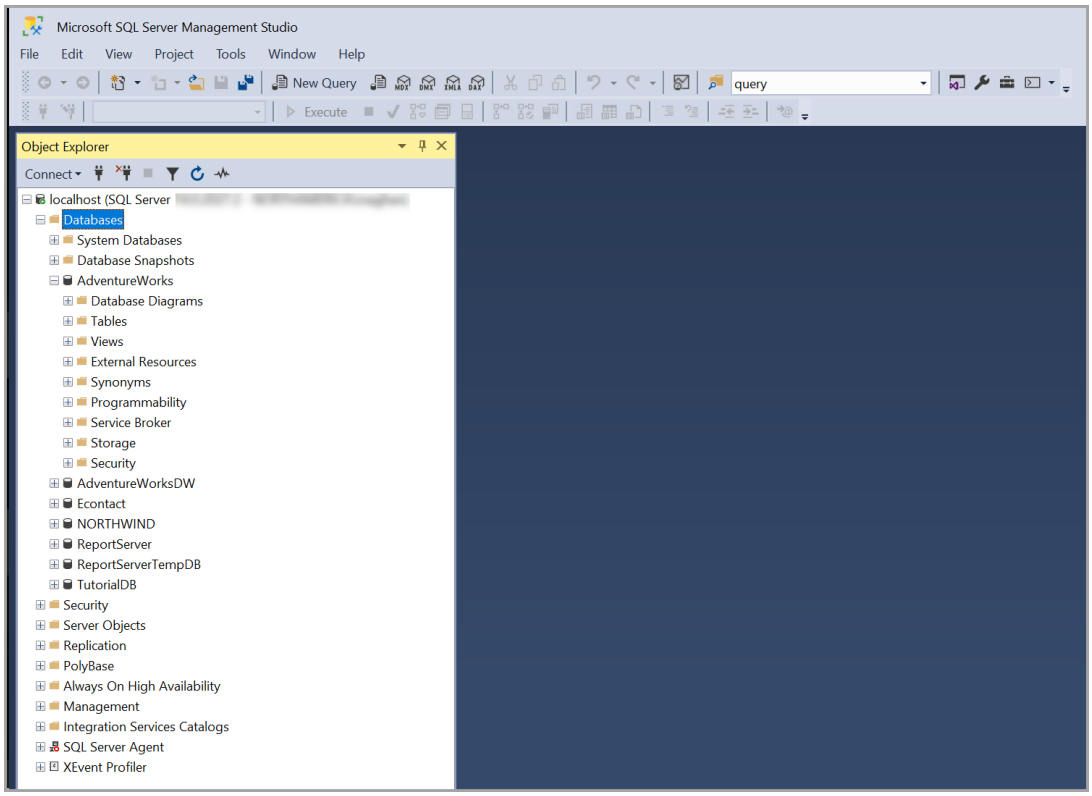
Microsoft Visual Studio (dále jen Visual Studio) je vývojové prostředí (zkratka IDE, anglicky Integrated Development Environment) od společnosti Microsoft. Používá se pro vývoj konzolových aplikací, webových stránek/aplikací, grafického rozhraní atd. Obsahuje editor kódu s podporou IntelliSense (např. dokončování slov) a velmi dobře zvládnutou funkci formátování kódu. Součástí Visual Studia je i debugger, který nám nabízí významnou pomoc při hledání chyb v našem kódu. (1) (2)



Obrázek 1 Uživatelské rozhraní Visual Studia (2)

1.2. Microsoft SQL Server Management Studio 18

Microsoft SQL Server Management Studio 18 (dále jen SQL Server) je systém pro tvorbu a správu relačních databází v jazyce T-SQL. Tento jazyk je rozšíření jazyka SQL od společnosti Microsoft. (3)



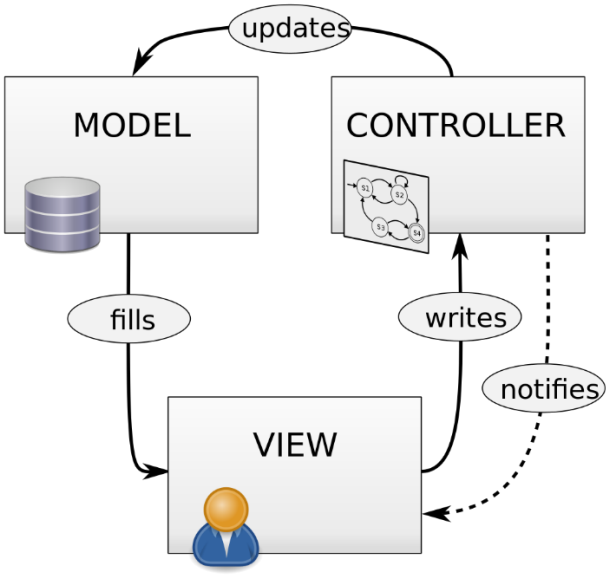
Obrázek 2 Uživatelské rozhraní SQL Serveru (3)

1.3. ASP.NET Core

ASP.NET je jedním z nejvýznamnějších rozšíření .NET frameworku (soubor technologií a softwarových produktů). Je nástupcem technologie ASP (Active Server Pages). Jde o open source framework, který umožňuje vývoj serverových částí dynamických webových aplikací a služeb. ASP.NET je cross-platform (aplikace vyvinuté v ASP.NET mohou být použity na více platformách). Konkrétně byl použit ASP.NET Core, což je nástupce ASP.NET, který je obsažen v novém vydání .NET Core. (4) (5)

1.4. MVC

Model View Controller (dále jen MVC) je softwarová architektura (někdy také chápána jako návrhový vzor), která aplikaci rozděluje do tří navzájem nezávislých komponent (modifikace jedné komponenty má jen minimální vliv na ostatní dvě). Tyto části, jak už název napovídá, se nazývají model, view a controller. Model obsahuje všechnu logiku, informace a data (často prostřednictvím přístupu k databázi), se kterými daná aplikace pracuje. View má na starosti převod obdržených dat do podoby, která je vhodná k prezentaci uživateli. Controller neboli řadič, je řídicí komponenta. Úkolem této komponenty je reagovat na události (nejčastěji pocházející od uživatele, může pocházet ale i z jiných zdrojů). Na základě těchto událostí zajišťuje změny v modelu a view se na základě těchto změn může aktualizovat. ASP.NET podporuje softwarovou architekturu MVC (konkrétně ASP.NET MVC). (6)



Obrázek 3 Diagram MVC (7)

1.5. SignalR

ASP.NET Core SignalR (dále jen SignalR) je open source knihovna, která umožňuje přidávání real-time funkcí do webových aplikací a služeb (zpracovávání funkcí v reálném čase, tak aby byl klientům nabídnut požadovaný obsah téměř okamžitě). Aplikace, které často implementují SignalR jsou např. Aplikace, které využívají různé druhy oznámení, řídicí panely, aplikace pro monitorování, aplikace vyžadující časté aktualizace ze serveru. SignalR může zasílat zprávu současně všem připojeným klientům nebo pouze pár určeným klientům anebo také pouze jednomu klientovi. Připojení zpracovává automaticky. Všechny připojené klienty SignalR rozlišuje podle jejich unikátních id, které jim přiděluje. Pro komunikaci mezi klienty a serverem SignalR používá tzv. rozbočovače (nebo také centrum, hub). Rozbočovač umožňuje klientům volat metody na serveru a naopak (8)

1.5.1. WebSocket

WebSocket je komunikační protokol, který poskytuje plně duplexní komunikaci (obousměrná komunikace, která může probíhat současně). Vytvořený komunikační kanál probíhá přes jedno TCP připojení (TCP je protokol garantující spolehlivé doručování zpráv po počítačové síti ve správném pořadí). WebSocket je ta nejpoužívanější technologie pro přenos zpráv, kterou SignalR používá. (9) (10)

1.6. SHA-256

SHA-256 (Secure Hash Algorithm) je hašovací funkce vyvinuta ve Spojených Státech Amerických. Tato funkce vytváří ze vstupních dat libovolné délky výstup fixní délky (nazývaný jako otisk, hash, miniatura). Mezi vlastnosti této funkce patří: stejná vstupní data pokaždé vygenerují stejný otisk a pouhá malá změna na vstupu vede k vytvoření zásadně odlišného otisku. Nicméně hlavní vlastností této funkce je fakt, že z otisku je prakticky nemožné získat zpět vstupní data. Z tohoto důvodu je tato funkce velice vhodná pro práci s uživatelskými hesly. Při registraci nového uživatele se nejprve z jeho hesla vygeneruje otisk a ten se uloží do

databáze, poté při opětovném přihlášení se opět ze zadaného hesla vytvoří otisk a pokud se daný otisk shoduje s otiskem uloženým v databázi, přihlášení je úspěšné. Pokud by v databázi byl uložen pouhý znakový řetězec představující uživatelské heslo, pak by v případě úniku dat bylo jeho heslo prozrazeno. Nevýhodou tohoto algoritmu je, že různá vstupní data mohou vygenerovat stejný otisk, z důvodu fixní délky otisku (256 bitů, proto název SHA-256) a tudíž konečného množství možných kombinací. Nicméně toto riziko je značně sníženo vlastností malá změna na vstupu dá velkou změnu na výstupu tzn. nikdo se nepřihlásí pouze s podobným heslem. (11) (12)

1.7. Entity Framework

Tento framework je součástí .NET. Umožňuje nám pracovat s databází ve větší míře abstrakce, protože z entit a atributů nám dokáže vytvořit třídy a vlastnosti, tudíž nám umožňuje pracovat s daty jako s objekty. Vždy, když se zavolá jeho metoda SaveChanges, Entity Framework zkontroluje, jaké změny byli provedeny (jaké objekty byli modifikovány, přidány, smazány) a aktualizuje podle toho databázi.

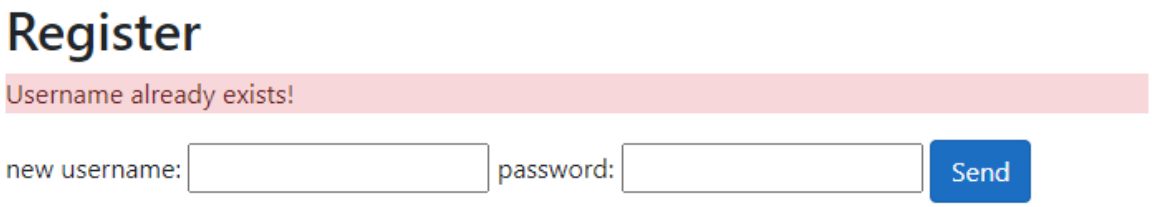
2. Uživatelské rozhraní

2.1. Úvodní strana

Po přístupu k webové aplikaci uživatel pravděpodobně zavítá na úvodní stránku. Tato stránka je velice jednoduchá, kromě vítacího nadpisu se zde nachází pouze tlačítko pro přihlášení existujícího uživatele do aplikace a tlačítko pro registraci nového uživatele. Tyto tlačítka nás po kliknutí přesměrují na příslušnou stránku pomocí správné URL adresy. URL adresa samotné úvodní strany má 2 verze. Buď lze napsat doménovou adresu serveru + /home anebo stačí jen doménová adresa, jelikož ASP.NET uznává /home za úvodní stranu a z důvodu zvýšení intuitivity proto podporuje i druhou variantu.

2.2. Registrace

Při první návštěvě je také velmi pravděpodobné, že uživatel nemá založený žádný účet. Nepřihlášený uživatel nemůže používat většinu služeb, které mu tato aplikace poskytuje, a proto je potřeba vytvořit si nový profil. Na stránku pro registraci se uživatel může dostal buď z úvodní stránky nebo zadáním správné URL adresy, což je doménová adresa + /register. Na této stránce se nachází formulář, obsahující 2 textová pole, jedno pro zadání jména nového uživatele a druhé pro zadání hesla. Po kliknutí na tlačítko “login” aplikace zkontroluje, jestli již v databázi neexistuje uživatel se stejným jménem jako je to, co nový uživatel zadal do prvního textového pole (uživatelská jména totiž jsou a musí být unikátní). Pokud ano, registrace se neuskuteční a uživateli se objeví varování informující ho o nemožnosti použití daného jména. V případě úspěšné registrace se jméno a heslo nového uživatele zapíše do databáze (heslo je samozřejmě převedeno na bezpečně zahashovaný tvar, více v kapitole 1.6). Uživatel může nyní používat všechny služby aplikace.



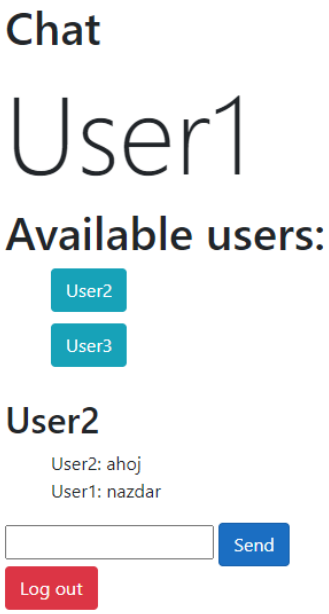
Obrázek 4 Varování při neúspěšné registraci

2.3. Přihlašování

V případě, že uživatel již má vytvořený účet, je potřeba umožnit mu přihlásit se k danému účtu. K tomu slouží přihlašovací stránka s URL adresou doménová adresa + /login. Vzhledově se vcelku podobá předchozí stránce pro registraci, funkcionalitu má ale jinou. Po zadání uživatelského jména a hesla se aplikace pokusí najít v databázi uživatele, který má stejné oba údaje. Pokud nedojde k žádné shodě, proces přihlašování se nezdaří a obdobně jako při nezdařené registraci se uživateli objeví varování o nesprávnosti údaje či obou údajů. V případě, že aplikace v databázi najde uživatele s totožnými údaji, osoba je přihlášena právě jako tento uživatel a od této chvíle je oprávněna používat všechny služby aplikace.

2.4. Chat

Hlavním účelem této aplikace je posílání zpráv mezi uživateli. Stránka určená k chatu má URL adresu doménová adresa + /chat a neoprávněný uživatel se na ní nedostane. Pro získání oprávnění je potřeba se přihlásit. Chatování v této aplikaci funguje formou Instant Messaging, jelikož zasílání zpráv je téměř okamžité (Pokud jeden uživatel napíše druhému uživateli, který je přihlášený a zrovna má otevřenou stránku konverzace s prvním uživatelem, druhý uživatel uvidí zprávu téměř okamžitě). Na této stránce může uživatel vidět seznam všech ostatních uživatelů, kterým může napsat. Na každého uživatele v seznamu lze kliknout a zobrazit si všechny zprávy, které si daní uživatelé poslali a také textové pole pro zadání a následné zaslání nové zprávy. Také se tu nachází možnost odhlášení. Po kliknutí na odhlašovací odkaz je uživatel odhlášen a jeho oprávnění pro vstup na tuto stránku je odebráno. Pro opětovné získání daného oprávnění je potřeba se znovu přihlásit buď k tomu samému účtu nebo k nějakému jinému.



Obrázek 5 UI pro výběr příjemce a posílání zpráv

3. Metodika systému

3.1. Služby

Ve třídě Startup (třída, jejíž instance se spouští při startu serveru) se nachází definice služeb. Nejprve metodou AddControllersWithViews přidáme do služeb controllery, které potřebujeme ke správnému chodu programu, protože používáme návrhový vzor MVC. Poté zaregistrujeme do služeb SignalR, který je opět potřeba. Poté se do služeb přidají vlastní třídy. V tomto případě jsou 4 – AppConfig, HashAlgorithm (viz. kapitola 4.2.4), CommunicationWithDB a ChatAppDBContext (viz kapitola 4.2.3). Tyto vlastní služby se dělí na 3 druhy: singleton (pouze jedna instance pro všechny HTML požadavky), scoped (pro každý HTML požadavek je instance jiná, ale pro 1 požadavek stejná) a transient (instance je pokaždý jiná). V tomto případě pouze třída AppConfig je služba typu singleton, protože se jedná o třídu nesoucí globální atributy o celé aplikaci. Zbytek jsou služby typu scoped (všechny služby by teoreticky mohli být typu singleton, ale použití typu scoped je výhodnější z důvodu rychlosti, pro každý požadavek se vytvoří nová instance a požadavky se mohou tedy zpracovávat paralelně). (13)

```
services.AddControllersWithViews();
services.AddSignalR();
services.AddSingleton<AppConfig>();
services.AddScoped<CommunicationWithDB>();
services.AddScoped<ChatAppDBContext>();
services.AddScoped<HashAlgorithm>();
```

Obrázek 6 Služby aplikace

3.2. Autentizace a autorizace

Ve třídě Startup se také definuje autentizace a autorizace. Autentizace je zjišťování identity uživatele a autorizace je určování, na co všechno má daný uživatel práva. Z tohoto důvodu musí autentizace vždy předcházet autorizaci (nelze určit uživatelovi práva, když není jasné, kdo ten uživatel vlastně je).

K autentizaci lze použít několik způsobů, ale v této aplikaci se používá postup pomocí tzv. cookies. Cookies jsou malá data, která mají vždy svůj název a hodnotu. Jsou to informace pro prohlížeč a ukládají se na počítači uživatele. Tento program při autentizaci udělí uživateli cookie s názvem user-login a automaticky vygenerovanou hodnotou. Pokud se uživatel pokusí dostat se do části aplikace s omezeným přístupem, aplikace zkontroluje, jestli má uživatel požadované cookies a pustí ho dál pouze, pokud ho má. (14)

```
services.AddAuthentication("CookieAuth")
    .AddCookie("CookieAuth", config =>
    {
        config.Cookie.Name = "user-login";
        config.LoginPath = "/login/authenticate";
    });
```

Obrázek 7 Definice autentizace přes cookie

3.3. Konfigurace serveru

Konfigurace se nachází v JSON souborech. Nejdůležitější nastavení jsou hlavně v souboru appsettings.secret.json. Zde se nachází citlivá data serveru jako je tzv. connection string (textový řetězec představující adresu potřebnou k připojení k databázi) a tzv. salt (náhodný, neměnný znakový řetězec používaný k zesílení uživatelských hesel, více v kapitole 5.6). Tento soubor je potřeba uchovat na bezpečném uložišti.

```
{
  "ConnectionStrings": {
    "ChatAppDB": "Data Source=0.0.0.1;Initial Catalog=ChatAppDatabase"
  },
  "Security": {
    "Salt": "rAnDoM sTrInG"
  }
}
```

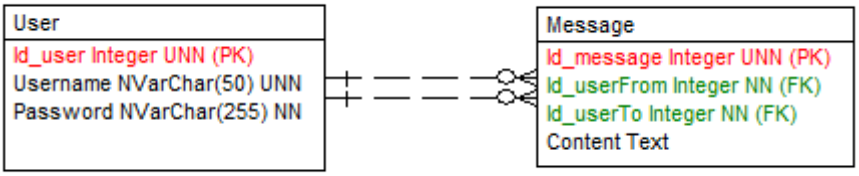
Obrázek 8 Obsah appsettings.secret.json (s falešnými daty)

Dále se tu nachází soubor libman.json, který slouží jako evidence všech použitých externích knihoven. V případě této aplikace je zde zapsaná pouze knihovna SignalR. Soubor appsettings.json, který je vygenerován již při založení nového projektu v ASP.NET ale v tomto případě nebylo potřeba do něj nic přidávat.

4. Struktura systému

4.1. Struktura databáze

Databáze je důležitou součástí zadané aplikace. V tomto případě databáze nemusí být nijak složitá, jediné, co si potřebujeme ukládat jsou informace o uživatelích a o zprávách mezi nimi. Jsou tedy potřeba jen dvě entity (tabulky) – uživatelé a zprávy. Vztah mezi těmito entitami je následující: K uživateli může patřit n zpráv, ale také k němu nemusí patřit žádná, zatímco ke zprávě jsou přidružení právě 2 uživatelé (odesílatel a příjemce). Každý uživatel má několik atributů: Unikátní celočíselný identifikátor, uživatelské jméno, které je pro každého uživatele unikátní, otisk hesla, do kterého se ukládá otisk vygenerovaný hašovacím algoritmem. Každá zpráva má opět několik atributů: Unikátní celočíselný identifikátor, text zprávy a jelikož jde o slabou entitu (entita, která nemůže existovat bez vztahu k silné entitě), musí obsahovat i cizí klíč k silné entitě (k uživateli), jinak by nemohla existovat). V našem případě má cizí klíče dva, 1 odkazující na odesílatele dané zprávy a 1 odkazující na příjemce.



Obrázek 9 Schéma databáze

4.2. Organizace kódu

4.2.1. Controllers

Jelikož se vývoj této aplikace drží návrhového vzoru MVC, je potřeba implementovat tzv. controllery jakožto řídicí komponenty programu. Tyto controllery se nacházejí, jak už název napovídá, ve složce Controllers. Controllery jsou v ASP.NET MVC architektuře od toho, aby se starali o všechny příchozí URL requesty (požadavky). Controller je třída, která obsahuje veřejné metody (tzv. action methods). Tyto metody reagují na příchozí požadavky, požádají služby, popřípadě model, o potřebná data a vrátí příslušnou odpověď. V ASP.NET MVC všechny controllery musí končit slovem „Controller“ a být potomkem třídy Controller z knihovny System.Web.Mvc. V tomto konkrétním projektu máme celkem 4 controllery. **HomeController**, který se stará o požadavky z úvodní stránky (v našem případě tam žádné ani být nemusí). **LoginController**, který se stará o přihlašování existujících uživatelů, o kontrolu správnosti přihlašovacích údajů a o přesměrování do chatovací aplikace v případě správných údajů. **RegisterController**, který je podobný předchozímu controlleru. Tento controller se stará o registraci nových uživatelů. Posledním a nejrozsáhlejším controllerem je **ChatController**. Ten se stará o zobrazování všech uživatelů, kterým lze poslat zprávu, zobrazování konverzace mezi

přihlášeným a navoleným uživatelem, posílání zpráv mezi uživateli a odhlašování uživatele. Tento controller má autorizovaný vstup (více v kapitole 5.7) tj. nemůže s ním komunikovat nepřihlášený uživatel. (15)

4.2.2. Hubs

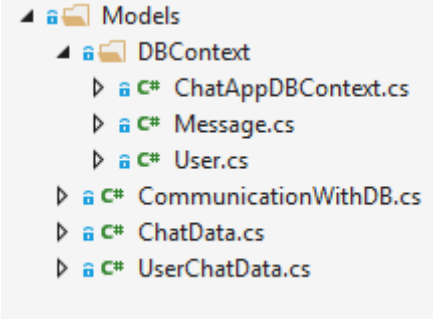
Do složky Hubs patří všechny rozbočovače (anglicky hubs) pro správnou funkci použité knihovny SignalR. Jak už bylo zmíněno v této zprávě (viz. kapitola 1.5), SignalR používá tyto rozbočovače, aby umožnil klientům volat metody na serveru a naopak. V případě zadané aplikace stačí pouze 1 rozbočovač, který se jmenuje ChatHub. Tento rozbočovač klientům nabízí hned několik metod, které mohou kdykoliv zavolat.

Metodu **SendMessage** volá klient, pokud uživatel na jeho straně posílá zprávu jinému uživateli. Tato metoda po zavolání najde určeného příjemce a na straně jeho klienta zavolá metodu RecieveMessage, která se postará o zobrazení zprávy. Metoda **GetConnectionId** vrací klientovi jeho id, které mu SignalR přidělil. Metoda **GetCOnnectionIdOfUser** vrací klientovi id jiného klienta (volá se, pokud je potřeba vědět id příjemce) a metoda **SetConnectionId** volá klient při startu komunikačního kanálu z důvodu uložení jeho nového id do systému.

```
public class ChatHub : Hub
{
    private readonly AppConfig _appconfig;


    public ChatHub(AppConfig onlineUsers)...

    public async Task SendMessage(string userFrom, string userTo, string message,
                                  string connectionIdFrom, string connectionIdTo)...

    public string GetConnectionId()...

    public string GetConnectionIdOfUser(string username)...

    public void SetConnectionId(string username, string id)...




}
```

Obrázek 10 Tělo rozbočovače

4.2.3. Models



Obrázek 11 Struktura složky Models

V této složce (jak už z názvu vyplývá) se nachází modely, tj. veškerá logika, informace a data aplikace. V tomto případě se zde nachází podsložka **DBContext**, ve které se nachází třídy pro komunikaci s databází. Tato podsložka byla vygenerována automaticky za pomocí Entity Frameworku (více v kapitole 1.7). Dále se tu nachází třída **CommunicationWithDB**, což je ručně vyvinutá nadstavba tříd z podsložky DBContext. Tato třída nabízí metody přímo vyrobené pro potřeby aplikace jako např. vytvořit nového uživatele, najít uživatele, zapsat do databáze novou poslanou zprávu atd. Tyto metody při zavolání udělají to, co mají prostřednictvím instancí tříd z podsložky DBContext a zbytek aplikace, pokud potřebuje přístup k databázi, komunikuje výhradně s touto třídou (prostřednictvím služby).

Kromě těchto tříd jsou tu ještě třídy **ChatData**, jejíž instance se posílá jako informace do view (viz kapitola 4.2.5). View z atributů této instance může vyčíst všechna potřebná informace, která má vypsat na uživatelské rozhraní. Instance třídy ChatData v sobě nese o všech uživatelích a o samotném uživateli. Třída **UserChatData** je potomek třídy ChatData a instance této třídy v sobě nesou informace stejné jako třídy ChatData a k tomu ještě informaci o uživateli, kterému může daný uživatel v danou chvíli poslat zprávu (jehož konverzaci má zrovna otevřenou na uživatelském rozhraní).

4.2.4. Services

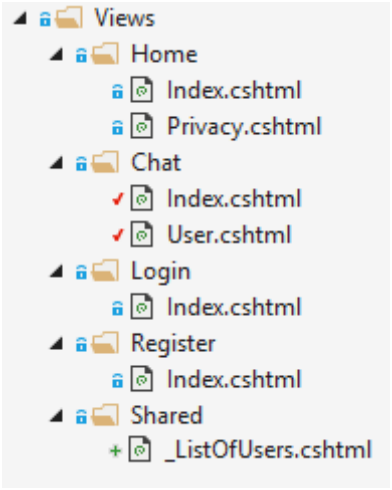
Ve složce Services se nacházejí služby, které má aplikace k dispozici. Konkrétně jde o třídu **AppConfig**, která obsahuje globální informace o celé aplikaci (connection string pro připojení do databáze a seznam všech uživatelů). **UserConfig** je třída, která nese informace o jednom konkrétním uživateli (pro každého uživatele existuje 1 instance třídy UserConfig). **HashAlgorithm** je třída, která se stará o vytváření otisků z uživatelských hesel, které je bezpečné ukládat (více v kapitole 1.6)

4.2.5. Views

Ve složce Views můžeme najít soubory s příponou .cshtml, což jsou předpisy HTML stránek obohacené o Razor (tzv. view engine, který nám umožňuje generovat dynamický obsah na HTML stránce za použití Razor značek a C# kódu, viz obrázek 14). Těmto souborům se říká Razor pages. Ještě, než klient převezme daný soubor pro zobrazení v uživatelském rozhraní, server tento soubor zkompiluje na obyčejný HTML soubor s nadefinovaným obsahem. Používají se k zobrazování chtěných dat uživateli. Pohledy komunikují s controllery, ke kterým jsou přiděleny. V případě ASP.NET MVC se pohledy nacházejí ve složkách nesoucí stejné jméno

jako controller, ke kterým patří (např. view se jménem index ve složce Home bude komunikovat s metodou Index v HomeControlleru). Controller může mít na starosti více než 1 pohled ale pohled patří právě pod 1 controller. (16) (17)

V případě našeho projektu potřebujeme pohledy pro domovskou stránku, přihlašování, registraci, zobrazení uživatelů pro přihlášeného uživatele a zobrazení možnosti napsat jistému uživateli zprávu. Kromě těchto pohledů se tu nacházejí také tzv. částečné pohledy. Tyto pohledy nalezneme v podsložce Shared a jde o takové pohledy, které nejsou dokončené, ale ve výsledném HTML souboru se stanou součástí jiného většího pohledu. To se používá hlavně v případech, kdy potřebujeme stejnou část pohledu ve více pohledech. Tímto se elegantně vyhneme duplicitnímu kódu. V případě této aplikace se tu dá najít jeden částečný pohled se jménem _ListOfUsers.cshtml (je zvykem pojmenovávat částečné pohledy s podtržítkem na začátku). Tento částečný pohled vypisuje seznam všech dostupných uživatelů, který se vyskytuje v obou pohledech z podsložky Chat.



Obrázek 12 Struktura složky Views

```
@model ChatAppCoreMVC.Models.ChatData

<h1>Available users:</h1>

<ul>
    @foreach (string name in Model.Usernames)
    {
        <li>
            <a class="btn btn-info" href="@Url.Action("User", new { nameTo = name})">@name</a>
        </li>
    }
</ul>

<script src="~/js/signalr/dist/browser/signalr.js"></script>
```

Obrázek 13 Tělo částečného pohledu _ListOfUsers.cshtml

V obrázku č.14 je zobrazen částečný pohled _ListOfUsers.cshtml. Zde můžete vidět, že se opravdu podobá normálním HTML souborům. Je obohacený o Razor značky (symbol @) a o C# kód. Zavináč nám v tomto kontextu říká, kde začíná C# kód, který při kompilaci zmizí a pouze upraví nově vzniklý HTML soubor.

Na prvním řádku je vidět nastavování modelu. Tímto tomuto pohledu říkáme, jaký typ bude příchozí objekt, jenž je možno dále použít. Dále je vidět nečíslovaný list, jehož počet položek definuje cyklus foreach (počet položek odpovídá počtu prvků v množině Model.Usernames). Každý prvek se nahraje do proměnné name a s tou se pracuje v rámci jedné položky. Konkrétně se použije pro text odkazu a jako parametr při vyvolání HTML požadavku, která zavolá akční metodu s názvem User v controlleru ChatController (název kontroleru není potřeba udávat, protože cesta k němu je relativní a tento částečný pohled bude součástí pohledů z podsložky Chat).

4.2.6. App Settings

Zde se nachází JSON soubory obsahující nastavení a konfiguraci serveru (více v kapitole 3.3)

4.2.7. Program.cs

Třída program je třída vygenerovaná automaticky již při založení projektu v ASP.NET. Tato třída obsahuje statickou metodu Main, která je ta první metoda, která se volá při startu serveru. (18)

4.2.8. Startup.cs

Třída startup obsahuje metody Configure a ConfigureServices a dohromady definuje služby, autorizaci, autentizaci atd. (více v kapitole 5.7)

5. Vývoj aplikace

5.1. Databáze

Jako první krok pro vývoj zadané chatovací aplikace bylo vytvořit relační databázi (databází se začínat nemusí, ale je to jeden z rozumných začátků). Pro tuto aplikaci byl zvolen relační druh databáze. K vývoji databáze byl použit SQL server, přes který se dá připojit na požadovaný databázový server a založit tam novou databázi. Databáze byla vytvořena podle návrhu (viz kapitola 4.1). Po vytvoření databáze je vhodné přes SQL příkazy vložit do nové databáze první testovací data, které tam nemusí zůstat natrvalo. Poté je potřeba ve vlastnostech databáze zjistit její connection string a uložit si ho na pozdější použití.

5.2. Entity framework

Po založení nového projektu ASP.NET MVC ve Visual Studiu je potřeba propojit databázi s aplikačním modelem, abychom mohli s databází jednoduše komunikovat. K tomu se v .NET projektech v drtivé většině používá Entity Framework (více v kapitole 1.7). K vytvoření takového propojení můžeme použít 2 způsoby. Buď tzv. model-first (vytvoříme model a necháme podle něj Entity Framework vygenerovat databázi) anebo tzv. database-first (vytvoříme databázi a necháme podle ní Entity Framework vygenerovat model). V případě této aplikace byl použit přístup database-first, jelikož databáze již byla vytvořena v předchozím kroku a není proto potřeba ji vytvářet znovu ve formě modelu, stačí si model nechat vygenerovat.

5.3. Controllery

Nově založený projekt v ASP.NET již obsahuje HomeController (controller řídící úvodní stránku). V našem případě nebylo potřeba ho nijak modifikovat. Nové controllery, které postupně v průběhu vývoje přibyly jsou LoginController, RegisterController a ten nejrozsáhlejší ChatController (více v kapitole 4.2.1).

```
[Route("chat")]
[Authorize]

public class ChatController : Controller
{
    private readonly AppConfig _onlineUsers;
    private readonly CommunicationWithDB _communicationWithDB;

    public ChatController(AppConfig onlineUsers, CommunicationWithDB db)...

    [Route("")]
    [HttpGet]

    public IActionResult Index()...

    [Route("user/{nameTo}")]
    [HttpGet]

    public new IActionResult User(string nameTo)...

    [HttpPost]

    public IActionResult SendMessageToUser(string nameTo)...

    [Route("logout")]
    [HttpGet]

    public IActionResult Logout()...
```

Obrázek 14 Tělo třídy ChatController

V obrázku č.15 je vidět tělo třídy ChatController. Nahoře je vidět relativní URL adresa k danému controlleru a fakt, že pouze autorizovaný uživatel může funkce tohoto controlleru použít. Dále vidíme atributy, do kterých se nahrávají instance potřebných služeb (získané z parametrů z konstruktoru). Metoda index vrací pohled domovské stránky daného uživatele. Metoda User vrací pohled stránky určené pro komunikaci s jistým uživatelem (určený podle uživatelského jména poslané v atributu nameTo). Metoda SendMessageToUser se volá, když uživatel posílá zprávu a má za úkol informovat model o potřebné modifikaci databáze. Poslední metoda Logout obstarává odhlášení uživatele (odstranění autorizačního cookies). Některé metody jsou typu HttpGet a některé HttpPost. Rozdíl mezi těmito dvěma typy je takový, že GET metoda dostává parametr požadavku v URL adrese, zatímco POST metoda dostává parametr přímo jako součást požadavku. Oba tyto typy se používají k posílání dat od klienta k serveru (na které je většinou následně odpovězeno). U GET metod je také vidět jejich relativní URL adresa, která navazuje na adresu controlleru. (19)

5.4. Views

Při vytváření pohledů je potřeba je správně pojmenovat a uložit je do správné podsložky. Podle jejich názvu a umístění totiž daný pohled ví, ke kterému controlleru patří a naopak, přičemž k jednomu controlleru může patřit více pohledů, ale jeden pohled patří právě k jednomu controlleru (více v kapitole 4.2.5).

5.5. SignalR

Pro správný provoz zadané chatovací aplikace potřebujeme SignalR pro posílání zpráv. Když jeden uživatel pošle zprávu druhému uživateli, tak právě tomu druhému uživateli se zpráva musí zobrazit ve chvíli kdy přijde a bez obnovení stránky (pouze pokud je dotyčný uživatel online a má otevřenou stránku konverzace s uživatelem, který mu právě zprávu poslal). K tomu použijeme tzv. hub (rozbočovač) (v našem případě ChatHub). Tento rozbočovač se nachází na serverové části aplikace a čeká až klient pošle dotaz. V našem případě čeká až některý klient pošle zprávu jinému klientovi. V případě, kdy tato situace nastane, ChatHub příjme informaci o odeslání zprávy a danou zprávu odešle jak příjemci, tak i zpět odesílateli. Oba klienti na druhé straně neustále čekají na příchozí zprávu od hubu, která jim oznámí, co za zprávu jim přišlo. Kromě toho nabízí klientům možnost zjistit jejich id, které je potřeba pro rozeznání uživatelů. Zpráva se nezobrazí uživateli, pokud daná zpráva nebyla určena pro něj anebo pokud nemá uživatel otevřeného právě toho uživatele, který mu zprávu poslal. Další záležitosti jako např. zápis zprávy do databáze atd. signalR neřeší (maximálně předá informaci jiné službě), ten je pouze pro real-time zobrazení chtěných informací. (více v kapitole 4.2.2)

5.6. Bezpečnostní opatření

Tato webová aplikace ukládá osobní data o uživatelích (uživatelské jméno a heslo) a proto je potřeba implementovat ochranu osobních dat. V žádném případě nesmíme ukládat hesla v normálním tvaru tak jak jsou, v případě nějakého útoku, či nechtěnému úniku dat by každý hned věděl hesla uživatelů. Proto musíme ukládat pouze otisky hesel (více v kapitole 1.6). Poté musíme ještě implementovat tzv. salt, protože některá jednoduchá hesla se dají podle jejich otisků na internetu dohledat. Salt je pouze textový řetězec náhodných ale neměnných znaků, který je uložený na bezpečném místě v konfiguraci aplikace a při práci s hesly se přidá před či za heslo a dohromady se zahashuje. Tímto způsobem se z hesla stane neobvyklý řetězec znaků, který už podle otisku dohledat nejde.

5.7. Autentizace a autorizace

Jak už bylo zmíněno, funkce ChatControlleru klient nemůže používat, pokud není jeho uživatel autorizovaný. Nepřihlášený uživatel není autorizovaný (nemá autorizační cookies). Autentizace tudíž musí proběhnout ve chvíli, kdy se uživatel úspěšně přihlásí či registruje do aplikace. K tomu se v ASP.NET používají tzv. claimy (potvrzení). Ve chvíli, kdy dojde k přihlášení uživatele, vytvoří se jeho identita a ta se uloží jako taková identita, která je autorizovaná pro vstup do ChatControlleru. Ve chvíli, kdy se takto ověří, o jakého uživatele se jedná, uživatel dostane autorizační cookies (vzhledem k tomu, že do ChatControlleru může jakýkoliv přihlášený uživatel, není potřeba příliš řešit autorizaci). Když je potřeba později zjistit údaje o přihlášeném uživateli, stačí se na ně dotázat (viz. obrázek 16).

```
var userClaims = new List<Claim>()
{
    new Claim(ClaimTypes.Name, username)
};
var userIdentity = new ClaimsIdentity(userClaims, "user identity");
var userPrincipal = new ClaimsPrincipal(new[] { userIdentity });
HttpContext.SignInAsync(userPrincipal);
```

```
string username = HttpContext.User.Identities.ToArray()[0].Claims.ToArray()[0].Value;
```

Obrázek 15 Proces autentizace a dotaz na údaj o přihlášeném uživateli

Závěr

Aplikace, o které tato práce pojednává, splňuje všechny požadavky, které byli obsaženy v zadání (viz. úvod). Tato aplikace umožňuje jakémukoliv uživateli založit svůj účet, přihlásit se ke svému účtu, odhlásit se ze svého účtu, a hlavně umožňuje uživateli poslat textovou zprávu jakémukoliv jinému uživateli, co má v tomto systému založený účet. Posílání zpráv funguje téměř okamžitě (v případě že příjemce má otevřenou stránku konverzace s odesílatelem) a jiný uživatelé než příjemce a odesílatel se k obsahu jejich zpráv nedostane. Aplikace obsahuje intuitivní uživatelské rozhraní. Po důsledném otestování praktické části této práce bylo ověřeno, že nic neovlivňuje chod aplikace žádným nežádoucím způsobem. Aplikace sice nebyla nasazena do ostrého provozu, ale s případným nasazením v budoucnu by neměla žádný problém. V této práci existují místa pro případné vylepšení v budoucnu jako např. implementování skupinové konverzace, možnost přidat si přítele (aby se uživatelovi nezobrazovali všichni ostatní uživatelé, pouze jeho přátelé), posílání obrázků jako součást zprávy. I přes tyto možná zlepšení tato práce splňuje všechny požadavky ze zadání, a proto se podle mého názoru dá požadovat za úspěšnou. Existuje více způsobů, jak vyvinout takovou aplikaci, ale použití frameworku ASP.NET bylo v zadání, a navíc jde o jednu z validních možností pro vývoj takového typu webové aplikace. Tato práce mi byla přínosná, protože jsem se v průběhu naučil spoustu věcí o mém oboru (zejména v oblasti komunikace přes internet, vývoje webových aplikací v ASP.NET a práce s relačními SQL databázemi), které jsou v dnešní době uznávané jako velmi cenné. Také jsem díky této práci lépe připravený na budoucí práce většího rozsahu jako je bakalářská práce a diplomová práce.

Seznam obrázků

Použitá literatura

1. **Zeil, Steven J.** Integrated Development Environments. *Old Dominion University.* [Online] 14. 9 2017. [Citace: 14. 8 2020.] https://www.cs.odu.edu/~zeil/cs350/f17/Public/IDEs/index.html.

2. **Microsoft.** Visual Studio 2019. *Microsoft.* [Online] [Citace: 14. 8 2020.] https://visualstudio.microsoft.com/cs/vs/.

3. **Ghanayem, Mark.** Download SQL Server Management Studio (SSMS). *Microsoft Docs.* [Online] 22. 7 2020. [Citace: 14. 8 2020.] https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15.

4. **Microsoft.** What is ASP.NET? *Microsoft .NET.* [Online] [Citace: 14. 8 2020.] https://dotnet.microsoft.com/learn/aspnet/what-is-aspnet.

5. —. What is ASP.NET Core? *Microsoft .NET.* [Online] [Citace: 14. 8 2020.] https://dotnet.microsoft.com/learn/aspnet/what-is-aspnet-core.

6. **Adamek, Petr.** Model-view-controller. *Wikipedie.* [Online] 14. 9 2005. [Citace: 14. 8 2020.] https://cs.wikipedia.org/wiki/Model-view-controller.

7. **Deltacen.** Model-view-controller. *Wikipedia Commons.* [Online] 27. 5 2013. [Citace: 14. 8 2020.] https://commons.wikimedia.org/wiki/File:ModeleMVC.png.

8. **Addie, Scott.** Introduction to ASP.NET Core SignalR. *Microsoft Docs.* [Online] 27. 11 2019. [Citace: 14. 8 2020.] https://docs.microsoft.com/en-gb/aspnet/core/signalr/introduction?view=aspnetcore-3.1.

9. **Malý, Martin.** Web Sockets. *Zdroják.* [Online] 14. 12 2009. [Citace: 15. 8 2020.] https://www.zdrojak.cz/clanky/web-sockets/.

10. **Yale.** Introduction to TCP/IP . *Yale.* [Online] 2. 2 1995. [Citace: 16. 8 2020.] https://web.archive.org/web/20041204044203/http://www.yale.edu/pclt/COMM/TCPIP.HTM.

11. **Svoboda, Josef.** Hašovací funkce. *Wikipedie.* [Online] 26. 2 2008. [Citace: 16. 8 2020.] https://cs.wikipedia.org/wiki/Ha%C5%A1ovac%C3%AD_funkce.

12. **Xorbin.com.** What is SHA-256? *Xorbin.* [Online] [Citace: 16. 8 2020.] https://xorbin.com/tools/sha256-hash-calculator.

13. **akazemis.** AddTransient, AddScoped and AddSingleton Services Differences. *Stack Overflow.* [Online] 1. 7 2016. [Citace: 21. 8 2020.] https://stackoverflow.com/questions/38138100/addtransient-addscoped-and-addsingleton-services-differences.

14. **Microsoft.** Description of Cookies. *Microsoft Support.* [Online] 17. 4 2018. [Citace: 18. 8 2020.] https://support.microsoft.com/cs-cz/help/260971/description-of-cookies.

15. **TutorialsTeacher.** Controller. *TutorialsTeacher.* [Online] [Citace: 17. 8 2020.] https://www.tutorialsteacher.com/mvc/mvc-controller.

16. **FileInfo.** .CSHTML File Extension. *FileInfo.* [Online] 15. 12 2011. [Citace: 17. 8 2020.] https://fileinfo.com/extension/cshtml.

17. **Anderson, Rick, Mullena, Taylor a Vicarel, Dan.** Razor Referenční informace k syntaxi pro ASP.NET Core. *Microsoft Docs.* [Online] 12. 2 2020. [Citace: 17. 8 2020.] https://docs.microsoft.com/cs-cz/aspnet/core/mvc/views/razor?view=aspnetcore-3.1.

18. **TutorialsTeacher.** ASP.NET Core - Program.cs. *TutorialsTeacher.* [Online] [Citace: 17. 8 2020.] https://www.tutorialsteacher.com/core/aspnet-core-program.

19. **TutorialsPoint.** Difference between GET and POST method in HTTP. *TutorialsPoint.* [Online] [Citace: 20. 8 2020.] https://www.tutorialspoint.com/listtutorial/Difference-between-GET-and-POST-method-in-HTTP/3916.