Možnosti frameworku Blazor pro tvorbu aplikací z oblasti pojišťovnictví

Bc. Michal Horáček

Diplomová práce
2021

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2020/2021

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Michal Horáček**
Osobní číslo:	**A19355**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Forma studia:	**Kombinovaná**
Téma práce:	**Možnosti frameworku Blazor pro tvorbu aplikací z oblasti pojišťovnictví**
Téma práce anglicky:	**Using the Blazor Framework for Creating Applications in the Insurance Field**

Zásady pro vypracování

1. Popište současný stav vývoje webových aplikací.
2. Zaměřte se na framework Blazor a jeho možnosti, výhody a nevýhody.
3. Zpracujte přehled možností frameworku Blazor pro tvorbu aplikací pro pojišťovnictví a bankovnictví.
4. Navrhněte ukázkovou aplikaci z oblasti pojišťovnictví.
5. Vytvořte vzorová řešení demonstrující klíčové prvky navržené aplikace.
6. Demonstrujte výsledky.

Forma zpracování diplomové práce: **Tištěná/elektronická**

Seznam doporučené literatury:

1. Blazor | Build client web apps with C# | .NET. .NET | Free. Cross-platform. Open Source. [online]. Dostupné z: https://dotnet.microsoft.com/apps/aspnet/web-apps/blazor
2. WebAssembly [online]. [cit. 2019-11-14]. Dostupné z: https://webassembly.org
3. NAGEL, Christian. Professional C# 7 and .NET Core 2.0. 11th edition. Indianapolis: Wrox, a Wiley Brand, 2018. ISBN 978-1119449270.
4. ALBAHARI, Joseph a Ben ALBAHARI. C# 7.0 in a nutshell. 7th edition. Sebastopol: O´Reilly, 2018. ISBN 978-1491987650.
5. V. HAAS, Andreas, Andreas ROSSBERG, Derek L. SCHUFF, et al. Bringing the web up to speed with WebAssembly. HAAS, Andreas, Andreas ROSSBERG, Derek L. SCHUFF, et al. Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation. New York, NY: ACM, 2017, s. 185-200. ISBN 978-1-4503-4988-8.

Vedoucí diplomové práce: **Ing. Erik Král, Ph.D.**
Ústav počítačových a komunikačních systémů

Datum zadání diplomové práce: **15. ledna 2021**
Termín odevzdání diplomové práce: **17. května 2021**

doc. Mgr. Milan Adámek, Ph.D. v.r.
děkan

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN
L.S.

prof. Mgr. Roman Jašek, Ph.D. v.r.
ředitel ústavu

Ve Zlíně dne 15. ledna 2021

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 17. 5. 2021 Bc. Michal Horáček, v. r.

ABSTRAKT

Diplomová práce se zabývá možnostmi frameworku Blazor pro využití v oblasti pojišťovnictví. Cílem této práce je vytvoření ukázkové aplikace z oblasti pojišťovnictví a demonstrování klíčových prvků této aplikace. Teoretická část práce se zabývá současným stavem vývoje webových aplikací, frameworkem Blazor a možnostmi frameworku Blazor pro tvorbu aplikace pro pojišťovnictví a bankovnictví. V praktické části je navržena ukázková aplikace z oblasti pojišťovnictví a je zde popsané založení projektu a klíčové prvky aplikace. Nakonec je vytvořená aplikace demonstrována.

Klíčová slova: Blazor, WebAssembly, webová aplikace, C#

ABSTRACT

The diploma thesis is focused on the possibilities of framework Blazor for use in the field of insurance. The aim is to create a sample application in the field of insurance and demonstrate the key elements of this application. The theoretical part of the thesis deals with the current state of web application development, Blazor framework, and the possibilities of the Blazor framework for creating insurance or bank application. In the practical part of the thesis is designed sample application in the field of insurance and described project creation and key elements of the application. Finally, the created application is demonstrated.

Keywords: Blazor, WebAssembly, web application, C#

Tímto chci poděkovat vedoucímu práce Ing. Et Ing. Eriku Králi, Ph.D. za odborné rady a časté konzultace diplomové práce.

Prohlašuji, že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

V dnešní době se nejvíce používá JavaScript pro client-side část aplikace. Framework Blazor přináší možnost vytvořit většinovou část aplikace prakticky v jednom programovacím jazyce. Toto přináší veliké výhody pro firmy, kdy firmy nepotřebují učit nové programátory dva hlavní programovací jazyky, aby mohli dělat změny v projektech. Toto šetří čas potřebný k naučení technologií projektu nového programátora. Další velkým kladem tohoto frameworku je možnost běhu client-side části aplikace na straně klienta v prohlížeči pomocí WebAssembly. Díky tomu se ušetří zatížení serveru a může vest k rychlejší responzivitě aplikace.

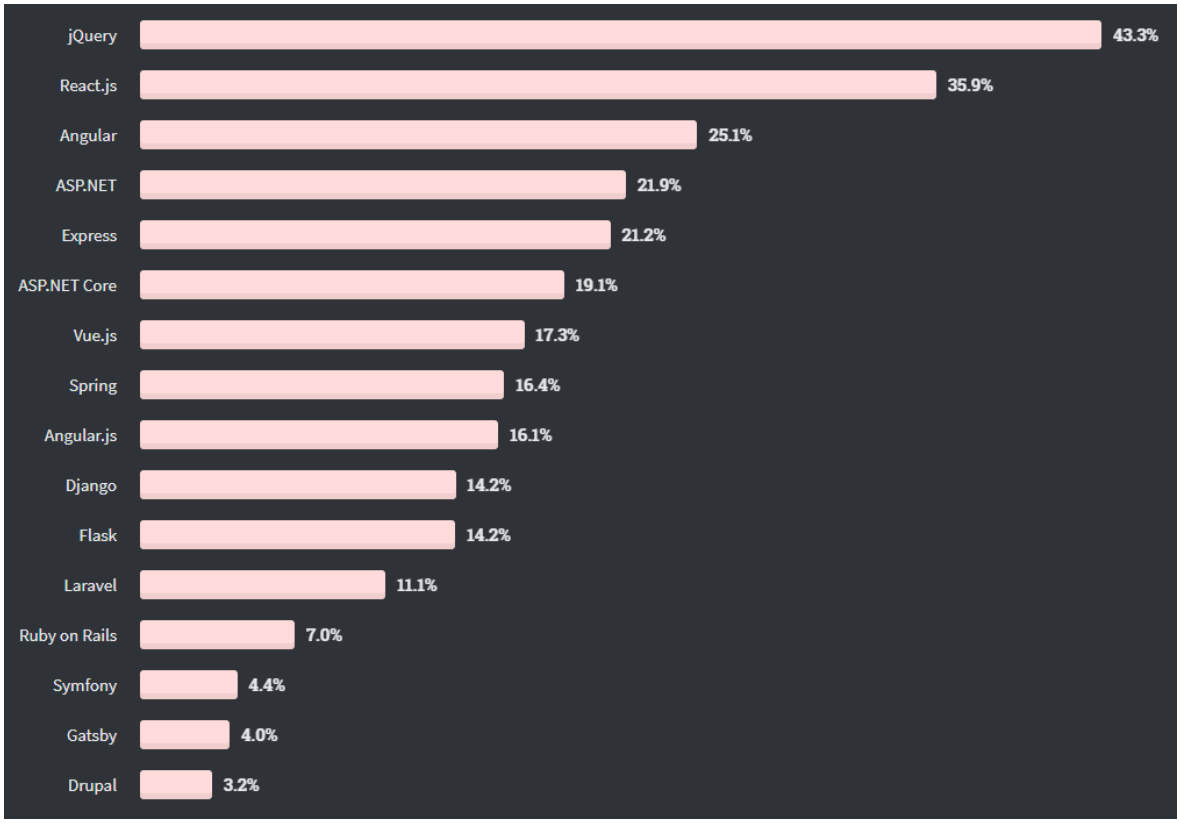
Teoretická část práce se zabývá současným stavem vývoje webových aplikací, popisem frameworku Blazor a možnostmi frameworku Blazor pro tvorbu aplikací pro pojišťovnictví a bankovnictví. V kapitole současného stavu vývoje webových aplikací se rozebírá dotazník webových a databázových technologií z webu StackOverflow za rok 2020. Dále je zde popsáno pět nejpopulárnějších webových technologií a dvě nejpopulárnější databázové technologie. Mezi popsané webové technologie patří jQuery, React.js, Angular, ASP.NET, Express. Popsané databázové technologie jsou MySQL a PostgreSQL.

Praktická část práce se zabývá návrhem aplikace, popisem založení projektu, popisem klíčových částí aplikace a demonstrací vytvořené aplikace. V kapitole návrhu aplikace jsou popsány funkční požadavky, nefunkční požadavky a scénáře použití aplikace. V kapitole klíčové části aplikace jsou popsány využité šablony a nastavení projektu. Dále jsou zde popsány hlavní části aplikace, které demonstrují použitelnost frameworku Blazor pro aplikace v oboru pojišťovnictví a bankovnictví. Mezi klíčové části aplikace patři autorizace uživatele, komponenty, uchování dat skrze stránky, validace vstupů formuláře, práce s databází, vytvoření PDF souboru smlouvy, posílání emailu se smlouvou a využití sdíleného kódu pro Client a Server projekt. U každé klíčové časti aplikace je popsaná použitá technologie s ukázkami částí kódu z aplikace. V kapitole demonstrace aplikace jsou popsané a ukázané jednotlivé stránky aplikace, vygenerovaný soubor se smlouvou a email, který se posílá klientovi.

I. TEORETICKÁ ČÁST

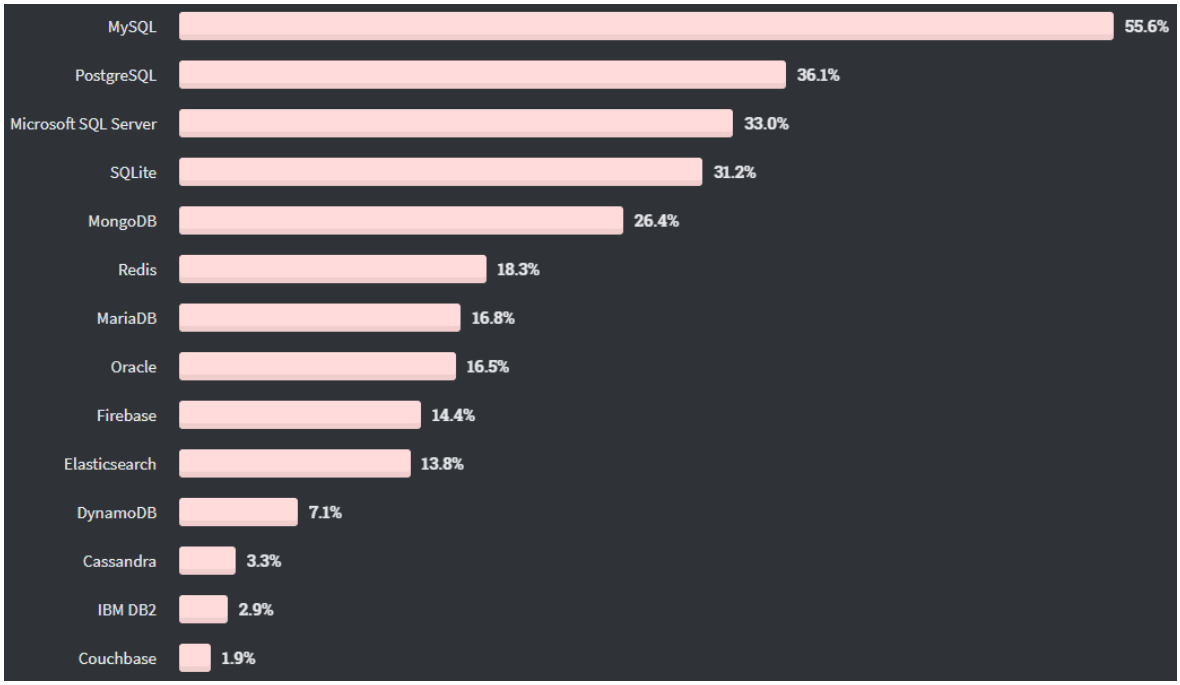
1 SOUČASNÝ STAV VÝVOJE WEBOVÝCH APLIKACÍ

Vývoj webových aplikací je velmi nestálý a co každý rok můžeme vidět nárůst zájmu u jiných nebo nových technologií. Podle dotazníku StackOverflow, kde odpovídalo 42 279 respondentů, stále převládá jQuery za rok 2020 [1]. Podle dotazníků z minulých let postupně používá jQuery méně lidí (pokles 5 % oproti roku 2019), kdežto React.js stoupá na popularitě [1]. Na obrázku číslo 1 můžeme vidět srovnání technologií za rok 2020.



Obrázek 1 Srovnání technologií za rok 2020 [1]

V oblasti databázových technologií stále převládá MySQL s 54 procenty. V této oblasti zůstává rozložení využívání velmi podobné rok po roku. Jediná změna v pořadí oproti roku 2019 je to, že Firebase překonal Elasticsearch [1]. Na obrázku číslo 2 můžeme vidět srovnání databázových technologií za rok 2020.



Obrázek 2 Srovnání databázových technologií za rok 2020 [1]

1.1 jQuery

Knihovna jQuery [19] je malá, rychlá a velmi obsáhlá JavaScript knihovna, která usnadňuje práci s HTML dokumenty. jQuery bere mnoho běžných úloh, které potřebují ke svému vykonání spoustu řádků JavaScript kódu a zaobalí je do metod, které se můžou volat jedním řádkem kódu. jQuery také zjednodušuje hodně složitých věcí z JavaScriptu, jako AJAX volání nebo manipulaci s DOM. Tato knihovna je velmi lehká (normální verze 247 KB, minimalizovaná verze 84 KB [3]) a kompatibilní napříč platformami [2]. jQuery se stále hodně používá. Z jednoho miliónu největších webů využívá jQuery 74.6 % [4]. Hlavním obsahem jQuery knihovny jsou funkce [2]:

- manipulace s HTML/DOM
- manipulace s CSS
- metody obsluh událostí
- efekty a animace
- AJAX volání

- Utility

1.2 React.js

React.js [20] je JavaScript knihovna, která se používá při vývoji webů k vytváření interaktivních prvků na webových stránkách. Před Reactem vývojáři vytvářely složitější znovupoužitelné UI prvky velice obtížně pomocí JavaScriptu a jQuery. To způsobovalo delší vývojový čas potřebný k vytvoření UI prvků a spoustu příležitostí pro udělaní chyb. React nabízí snadný a přehledný způsob vytváření znovupoužitelných UI prvků. React.JS byl vytvořen inženýrem Jordanem Walke ze společnosti Facebook [5], pro zlepšení vývoje uživatelského rozhraní. Tato knihovna se považuje za frontend knihovnu, jelikož se výhradně používá pro programování kódu na straně uživatele. Kromě toho, že React poskytuje znovupoužitelné interaktivní prvky, obsahuje i dvě klíčové funkce, které přidávají jeho přitažlivosti pro JavaScript vývojáře. Tyto dvě funkce jsou: JSX [21] a Virtual DOM [22].

1.2.1 JSX

V srdci každé základní webové stránky jsou HTML dokumenty. Webové prohlížeče čtou tyto dokumenty a zobrazují je v prohlížeči. Během tohoto procesu, prohlížeč vytváří takzvaný DOM. DOM je stromové zobrazení uspořádání webové stránky. Vývojáři můžou přidávat dynamický obsah do webových stránek úpravou modelu DOM pomocí jazyků, jako je například JavaSript. JSX je React rozšíření, které usnadňuje webovým vývojářům úpravy DOM pomocí jednoduchého kódu ve stylu HTML. JSX navíc je kompatibilní se všemi moderními prohlížeči. Kromě usnadnění úpravy DOM JSX vede k výraznému vylepšení výkonu webu. [5]

1.2.2 Virtual DOM

DOM je objektový model dokumentu. Pomocí tohoto modelu se znázorňují strukturované dokumenty (weby) pomocí objektů [23]. Virtual DOM je kopie DOM webu, kterou React.js využívá k zjišťování částí DOM, které jsou potřeba změnit, když dojde k uživatelské události. Po uživatelské události se tedy zjistí, které části je potřeba změnit a ty se aktualizují místo toho, aby se celý DOM aktualizoval. Tento druh selektivní aktualizace DOM vyžaduje méně výpočetního výkonu a kratší dobu načítání. [5]

1.3 Angular

Angular [24] je framework pro vytváření jednostránkových klientských aplikaci pomocí HTML a TypeScript. Angular je psán v jazyku TypeScript. Implementuje základní a volitelné funkce jako sadu TypeScript knihoven, které importuje do svých aplikací. Architektura Angular aplikace se opírá o určité základní prvky. Základními stavebními bloky Angular frameworku jsou Angular komponenty, které jsou organizovány do modulů NgModul. NgModuly shromažďují související kód do funkčních sad. Každá Angular aplikace je definovaná sadou NgModulů. Aplikace má vždy alespoň jeden kořenový modul, který umožňuje bootstrapování [6]. Původní název první verze angularu se jmenovala Angular.js [25] nebo Angular 1.x. Tato verze používala pouze JavaScript a byla oficiálně vydána v roce 2010 Misko Heverym a Adamem Abronsem [7]. Další verze byla kompletně přepsaná stejným týmem, co vytvořil Angular.js. Tato verze se jmenovala Angular 2 a už nabízela více jazyku, jako například TypeScript nebo Dart. Další verze už jenom inkrementovaly tento název o 1 [7]. Aktuální verze Angularu je 11.0.0. [8]

1.4 ASP.NET

ASP.NET [26] je platforma pro vývoj webových aplikací, která poskytuje programovací model, komplexní softwarovou infrastrukturu a různé služby potřebné k vytváření webových aplikací pro počítače a mobilní zařízení. Tato platforma pracuje na vrchní vrstvě HTTP protokolu a používá HTTP příkazy a zásady k vytváření prohlížeč-server oboustranné komunikace a spolupráce. ASP.NET je součástí .Net platformy od společnosti Microsoft. Aplikace ASP.NET jsou kompilované kódy psané pomocí rozšiřitelných a opakovaně použitelných komponent nebo objektů obsažených v .Net frameworku. Tyto kódy mohou využívat celou řadu tříd v rámci .Net frameworku. ASP.NET aplikace lze psát ve čtyřech programovacích jazycích [9]:

- C#
- Visual Basic.Net
- JScript
- J#

ASP.NET se používá k vytváření interaktivních webových aplikací. Skládá se z velkého počtu ovládacích prvků, jako jsou například textové pole, tlačítka a štítky pro sestavování a manipulaci s kódem za účelem vytvoření HTML stránek [9]. Microsoft vydal ASP.NET jako

součást .NET frameworku verze 1.0 během ledna 2002 [10]. ASP.NET je nástupcem technologie Microsoft Active Server Pages (ASP) [27]. V roce 2009 Microsoft vydal nový framework v rámci rodiny ASP.NET a to ASP.NET MVC. Tento framework byl založen na návrhovém vzoru MVC a byl vydaný pod licencí s otevřeným přístupem ke kódu MS-PL [28]. V roce 2016 firma Microsoft vydala nástupce ASP.NET a to ASP.NET Core 1.0 [29].

1.5 Express

Express.js [30] je bezplatný open-source webový aplikační framework pro Node.js. Používá se pro rychlé a snadné navrhování a vytváření webových aplikací a aplikačních programovacích rozhraní. Webové aplikace jsou aplikace, které se spouští v prohlížeči. Pro programátory je snadnější navrhování a vytváření webových aplikací v Express.js frameworku, protože je potřeba pouze JavaScript. Express.js se používá jako back-end framework pro Node.js. Pomocí Express.js lze vytvářet jednostránkové, vícestránkové a hybridní webové aplikace. Tento framework je součástí takzvaného MEAN software stacku [11]. MEAN stack je skupina technologií, pomocí kterých lze tvořit moderní aplikace celé vyvíjené v JavaScriptu. Díky této skupině technologií, se z JavaScriptu stal full-stack programovací jazyk. Každé z písmenek ve slově MEAN představuje jednu technologii [12]. Technologie jsou [12]:

- MongoDB – databázový server, který používá JSON pro dotazy a ukládá datové struktury do binárního JSON formátu.
- Express – JavaScript framework využívaný na straně serveru.
- Angular – JavaScript framework využívaný na straně uživatele.
- Node.js – JavasScriptové běhové prostředí.

1.6 MySQL

MySQL [31] je systém správy relačních databázích vyvíjený společností Oracle, který je založen na jazyku strukturovaných dotazů (SQL) [14]. MySQL byl vytvořen švédskou firmou MySQL AB v roce 1995. Původní vývojáři za MySQL byli Michael Widenius, David Axmark a Allan Larsson. Hlavní účel za MySQL bylo vytvoření efektivní a spolehlivé možnosti správy dat pro domácí i profesionální užití. MySQL, který byl původně majetkem MySQL AB, se v roce 2000 stal systémem s otevřeným přístupem ke zdrojovému kódu a začal se řídit podmínkami GPL [13]. Nyní MySQl patří pod firmu Oracle, který ji i spravuje. Primární rozdíl v relačních databázích oproti jiným digitálním uložištím spočívá v tom, jak

jsou data organizována na vrchní úrovni. Databáze MySQL obsahuje záznamy v několika samostatných a vysoce flexibilních tabulkách na rozdíl od jediného všeobjímajícího uložiště nebo skupině polostrukturovaných nebo nestrukturovaných dokumentů. [14]

1.7 PostgreSQL

PostgreSQL [15] je velice výkonný objektově-relační databázový systém, který využívá a rozšiřuje jazyk SQL. PostgreSQL má otevřený přístup ke zdrojovému kódu. Počátek PostgreSQL sahá do roku 1986 jako součást projektu POSTGRES na Kalifornské univerzitě v Berkeley a má více než 30 let aktivního vývoje na jeho jádru. V době psaní práce je aktuální verze 13.2 [32]. PostgreSQL běží na všech moderních operačních systémech, splňuje vlastnosti ACID od roku 2001 a obsahuje spoustu doplňků jako například velmi populární PostGIS. PostgreSQL obsahuje mnoho funkci, jejíž cílem je pomoct vývojářům vytvářet aplikace, správcům chránit integritu dat, vytvářet prostředí odolná vůči chybám a pomáhat spravovat data bez ohledu na to, jak velká či malá je datová sada. Kromě toho, že PostgreSQL je bezplatný a s otevřeným přístupem ke zdrojovému kódu, je také velmi rozšiřitelný. Lze například definovat vlastní datové typy, vytvářet vlastní funkce a také umožňuje psaní kódu v různých programovacích jazyků, aniž by se musela překompilovat databáze. [15]

2 FRAMEWORK BLAZOR

Blazor [33] je framework na vytváření jednostránkových aplikací. Název Blazor vznikl spojením slov prohlížeč (Browser) a Razor (.Net modul vytvářející HTML stránky). Blazor je framework s otevřeným přístupem ke zdrojovému kódu. Zdrojový kód je vlastněn neziskovou organizací .NET Foundation za účelem podpory projektů s otevřeným přístupem kódu, založených na rozhraní .NET. Blazor nepotřebuje instalovat plugin na straně klienta k tomu, aby mohl být spuštěný v prohlížeči. Blazor se vykonává na straně serveru nebo přímo v prohlížeči pomocí WebAssembly [34]. WebAssembly je webový standard, který je podporován všemi hlavními prohlížeči [18].

2.1 Historie

Steve Sanderson z firmy Microsoft začal Blazor jako svůj osobní projekt. V roce 2017 Steve představil na konferenci NDC Oslo svůj tehdy osobní projekt Blazor veřejnosti. První verze byla založena na interpreteru .NET CIL (Common Intermediate Language) běhového prostředí DotNetAnywhere [17]. DotNetAnywhere byl navržen tak, aby byl plně v souladu s .NET běhovým prostředím, což znamená, že může spouštět .NET knihovny a spouštěcí soubory. V dnešní době DotNetAnywhere už není aktivní [16]. Od tohoto prvního dema přidala firma Microsoft Blazor do jejich ASP.NET GitHub organizace jako experimentální projekt. Po přijetí Blazoru ASP.NET tým celý framework přepsal od začátku. ASP.NET tým vyměnil DotNetAnywhere za Mono, který obsahuje mnohem větší a plně funkční nabídku .NET běhového prostředí. Program Mono je součástí firmy Microsoft od roku 2016 a je oficiálním běhovým prostředím pro klientské platformy. Mono je využívané například u Xamarin frameworku a herního vývojového prostředí Unity [17].

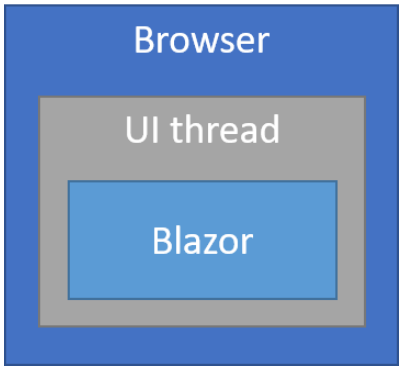
2.2 Modely hostování

Blazor je webový framework navržený ke spuštění na straně klienta (client-side) v prohlížeči na běhovém prostředí .NET založeném na WebAssembly nebo na straně serveru (server-side). Bez ohledu na model hostingu, jsou modely aplikace a komponent stejné.

2.2.1 Blazor WebAssembly

Hlavní model hostingu Blazoru je běh aplikace na straně klienta v prohlížeči přes WebAssembly. Blazor aplikace, její závislosti a .NET běhové prostředí se stahují přímo do prohlížeče. Aplikace se spouští ve vláknu uživatelského rozhraní prohlížeče. Aktualizace

uživatelského rozhraní a zpracování událostí probíhají ve stejném procesu. Prostředky aplikace jsou nasazeny jako statické soubory na webový server nebo na službu, která je schopná poskytovat klientům statický obsah. Na obrázku číslo 3 můžeme vidět grafické zobrazení modelu hostování Blazor WebAssembly.



Obrázek 3 Model hostování Blazor WebAssembly [45]

Pokud je aplikace Blazor WebAssembly vytvořena pro nasazení bez aplikace back-end ASP.NET Core, která poskytuje soubory, tak se tato aplikace nazývá standalone (samostatná) aplikace Blazor WebAssembly. Naopak pokud je aplikace vytvořena pro nasazení s backend aplikací, která poskytuje soubory, tak se tato aplikace nazývá hosted (hostovaná) Blazor WebAssembly aplikace. Hostovaná Blazor WebAssembly aplikace typicky interaguje se serverem přes síť pomocí volání webového API nebo SignalR. Hostovaná Blazor WebAssembly aplikace podporuje Docker kontejnery. Následují klady a zápory Blazor WebAssembly. [45]

Klady:

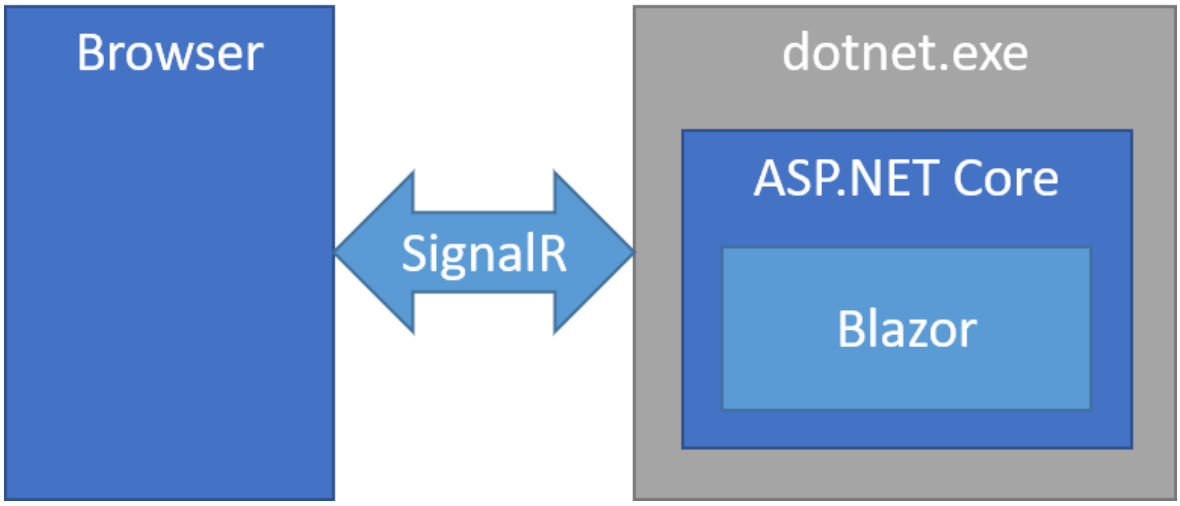
- Neexistuje žádná .NET závislost na straně serveru. Aplikace je plně funkční po stažení do prohlížeče.
- Klientské možnosti a prostředky jsou plně využívány.
- Práce je závislá na klientovi, a ne na serveru.
- Není potřebný ASP.NET Core webový server k hostování aplikace. Jsou možné scénáře bez nasazení serveru, jako například poskytování aplikace z Content Delivery Network (CDN)

Zápory:

- Aplikace je omezena možnostmi prohlížeče.
- Je potřebný výkonný uživatelský hardware a software (například nutná podpora WebAssembly).
- Velikost balíčku potřebný ke stažení do prohlížeče je větší a načítání trvá déle.
- Podpora běhového prostředí .NET a nástrojů je méně obsáhlá. Například existují omezení v podpoře .NET Standardu a ladění.

2.2.2 Blazor Server

U Blazor Server hostujícího modelu se aplikace spouští na straně serveru v aplikaci ASP.NET Core. Aktualizace uživatelského rozhraní, zpracování událostí a volání JavaScriptu jsou zpracovány přes připojení SignalR. Na straně uživatele se pomocí skriptu blazor.server.js navazuje SignalR spojení se serverem. Tento skript je poskytován uživatelské aplikaci z integrovaného prostředku ve sdíleném frameworku ASP.NET Core. Aplikace na straně klienta je zodpovědná za udržování a obnovování stavu aplikace dle potřeby. Na obrázku číslo 4 můžeme vidět grafické zobrazení modelu hostování Blazor Server.



Obrázek 4 Model hostování Blazor Server [45]

Blazor server aplikace se liší od tradičního modelu renderování uživatelského rozhraní v ASP.NET Core aplikacích. Oba modely používají jazyk Razor k popisu obsahu HTML pro vykreslení, ale výrazně se liší v tom, jak se značky vykreslují. U tradičních modelů, se při renderování Razor stránky všechny řádky Razor kódů přetvoří na HTML v textové podobě. Pokud tedy přijde nějaký požadavek na změnu stránky (například zobrazení validace), tak se musí celá stránka překreslit a poslat klientovi. Blazor aplikace se skládá z opakovaně

použitelných prvků uživatelského rozhraní nazývaných komponenty. Při vykreslování komponenty Blazor vytváří stromovou strukturu zahrnutých komponent podobný HTML nebo XML s názvem objektový model dokumentu (DOM). Blazor vyhodnotí tento graf, aby vytvořil binární reprezentaci značek. Z této binární reprezentace může Blazor vytvořit HTML kód komponenty nebo ji může použít k efektivní aktualizaci značek během běžného vykreslování. Při aktualizaci uživatelského rozhraní se vytvoří DOM komponenty a vypočítá se rozdíl mezi předchozí komponentou. Tento rozdíl je nejmenší soubor DOM úprav, potřebných k aktualizaci uživatelského rozhraní na klientovi. Nakonec se tento rozdíl pošle klientovi v binární podobě a aplikuje se prohlížečem. Následují klady a zápory Blazor Server. [45]

Klady:

- Velikost balíčku potřebného ke stažení do prohlížeče je výrazně menší než u Blazor WebAssembly aplikace a aplikace se načítá mnohem rychleji.
- Aplikace plně využívá možnosti serveru včetně použití jakýchkoli rozhraní API kompatibilních s .NET Core.
- Jelikož se na serveru používá .Net Core k běhu aplikace, tak existující nástroje .NET jako například ladění fungují bez problému.
- Blazor Server aplikace běží i na prohlížečích, které nepodporují WebAssembly a i na zařízeních s omezenými prostředky.
- Aplikační .NET/C# kód včetně kódů komponent není poskytováno klientům.

Zápory:

- Obvykle je větší latence. Každá interakce uživatele zahrnuje síťový dotaz.
- Neexistuje žádná podpora offline režimu. Pokud dojde ke ztrátě klientského připojení, aplikace přestane fungovat.
- Škálovatelnost je náročná pro aplikace s mnoha uživateli. Server musí zpracovávat mnoho klientských připojení a stavů klientů.
- Server ASP.NET Core je vyžadovaný k poskytování aplikace. Neexistují scénáře nasazení bez serveru jako například poskytování aplikace ze sítě Content Delivery (CDN)

2.3 Blazor WebAssembly framework

Blazor WebAssembly je SPA framework pro vytváření interaktivních webových aplikací na straně klienta pomocí .Net. Tento framework používá veřejné webové standardy bez pluginů a bez toho, aby překládal kód do jiných jazyků. Blazor WebAssembly funguje ve všech moderních webových a mobilních prohlížečích. Spouštění .NET kódů přímo v prohlížeči je zajištěno pomocí WebAssembly. WebAssembly je kompaktní binární formát optimalizovaný pro rychlé stahování do prohlížeče a maximální rychlost provádění. WebAssembly kód dokáže přistupovat k plné funkcionalitě prohlížeče pomocí Javascript interop. Kód .NET je spuštěn pomocí WebAssembly v sandboxu prohlížeče s ochranou, kterou sandbox poskytuje pspproti škodlivým akcím na klientském počítači. Po sestavení Blazor WebAssembly aplik zelda switch ace a její spuštění v prohlížeči se stane [36]:

- Soubory C# kódů a soubory Razor jsou kompilovány do .NET sestavení.
- Tyto .NET sestavení a .NET běhové prostředí je staženo do prohlížeče.
- Blazor WebAssembly bootstrapuje .NET běhové prostředí a nakonfiguruje toto běhové prostředí k načtení sestavení pro aplikaci. Běhové prostředí Blazor WebAssembly používá JavaScript interop ke zpracování manipulace DOM a volání API prohlížeče.

Velikost publikované aplikace je kritickým faktorem pro její použitelnost. Stahování velké aplikace do prohlížeče může trvat relativně dlouho, což snižuje uživatelské komfort a dojem z aplikace. Blazor WebAssembly optimalizuje tuto velikost následujícími funkcemi [36]:

- Nepoužívaný kód je z aplikace odstraňován při publikování aplikace trimerem IL.
- Odpovědi http jsou komprimovány.
- Modul běhové prostředí .NET a sestavení se ukládají do paměti prohlížeče

2.4 Novinky Blazor

V .NET 6, který se plánuje na listopad 2021, má přijít spoustu novinek. Mezi novinky patří [35]:

- WebAssembly AoT kompilace: Tato kompilace bude umožňovat kompilování .NET kódů přímo do WebAssembly při publikování aplikace, za účelem značného zvýšení výkonu běhu aplikace.
- Hot reload (načtení za běhu): Tato funkcionalita umožní vývojářům dělat změny UI a kódů běžícím aplikacím bez ztráty stavu aplikace.
- Blazor hybridní desktopové aplikace: Kombinace Blazoru a .NET multiplatformního aplikačního UI k vytváření multiplatformních hybridních desktopových aplikací.
- Jedno souborové publikování: Možnost vytváření malých, samostatných a vysoce výkonných aplikací a služeb.
- Micro API: Zjednodušení vytváření API endpointů s daleko méně řádků kódů.
- HTTP/3: Přidání podpory HTTP/3 a QUIC.

3 MOŽNOSTI FRAMEWORKU BLAZOR PRO TVORBU APLIKACÍ PRO POJIŠŤOVNICTNÍ A BANKOVNICTVÍ

Možnosti frameworku budou demonstrovány na aplikaci z oblasti pojišťovnictví, jako zástupce skupiny podnikových aplikací. Výhodou frameworku je možnost psaní kódu aplikace prakticky v jednom jazyce a díky tomu sdílet kód mezi klientskou a serverovou částí aplikace. Tímto se sníží náklady na vytvoření aplikace. V této kapitole jsou popsány možnosti frameworku Blazor pro aplikace z pojišťovnictví a bankovnictví. Je zde popsaná autentifikace uživatele, zabezpečení obsahu, práce s databází a používání gRPC.

3.1 Autentifikace uživatele

Blazor používá existující ASP.NET Core mechanismy autentifikace k ověření identity uživatele aplikace. Přesný mechanismus autentifikace závisí na tom, jak je aplikace hostovaná.

3.1.1 Blazor WebAssembly autentifikace

Ve Blazor WebAssembly aplikacích lze obejít ověřovací kontroly uživatele, protože uživatelé mohou upravit veškerý kód na straně klienta. To stejné lze říct o všech technologiích aplikací na straně klienta, včetně JavaScript SPA frameworků nebo nativních aplikací pro libovolný operační systém.

Blazor WebAssembly aplikace jsou zabezpečeny stejným způsobem jako SPA aplikace. Existuje několik přístupů k ověřování uživatelů u SPA, ale nejběžnějším a komplexním způsobem je použití implementace založené na protokolu OAuth 2.0 [38] jako je OpenID Connect [39] (OIDC)

Blazor WebAssembly podporuje ověřování a autorizaci aplikací pomocí OIDC prostřednictvím knihovny Microsoft.AspNetCore.Components.WebAssembly.Authentication. Knihovna poskytuje sadu funkcí pro bezproblémové ověřování proti backendu ASP.NET Core. Knihovna integruje ASP.NET Core Identity s podporou autorizace API postavenou na serveru Identity Server. Knihovna se může autentizovat proti jakémukoli poskytovateli identity třetích stran, který podporuje OIDC. Těmto poskytovatelům se říká OpenID Providers (OP). Podpora ověřovaní uživatele v Blazor WebAssembly je postavena na knihovně oidc-client.js, která se používá ke zpracování podrobností ověřovacího protokolu. Jiné možnosti ověřování SPA existují, jako například použití SameSite cookies. Design Blazor WebAssembly je však vyřešen tak, že OAuth a OIDC je nejlepší možnost pro ověřování v aplikacích

Blazor WebAssembly [37]. Autentifikace založená na JSON web tokenech byla vybrána před autentifikací založenou na cookies z funkčních a bezpečnostních důvodů. Tato autentifikace byla vybrána z následujících důvodů [37]:

- Používání protokolu založeného na tokenech nabízí menší plochu útoku, protože tokeny se neodesílají ve všech požadavcích.
- Server endpointy nepotřebují ochranu proti padělání žádostí mezi weby (CSRF), protože tokeny jsou odesílány explicitně. Díky tomu lze hostovat Blazor WebAssembly aplikace zároveň s MVC nebo Razor pages aplikacemi.
- Tokeny mají kratší životnost, standardně jednu hodinu, což omezuje okno útoku. Tokeny lze také kdykoliv odvolat.
- Tokeny s OAuth a OIDC se nespoléhají na to, že se uživatelský agent bude chovat správně, aby zajistil zabezpečení aplikace.
- Samostatné JWT tokeny nabízejí klientovi a serveru záruky ohledně procesu ověřování. Pokud se třetí strana pokusí přepnout token uprostřed procesu ověřování, může klient detekovat přepnutý token a vyhnout se jeho použití.

3.1.2 Blazor Server autentifikace

Blazor server aplikace fungují pomocí připojení v reálném čase, které je vytvořeno pomocí SignalR. Ověření uživatele v aplikacích založených na SignalR je zpracováno při navázání připojení. Ověření může být založeno na cookies nebo jiném nosiči tokenu. Integrovaná služba AuthenticationStateProvider pro Blazor server aplikace získává data stavu ověřování z ASP .NET Core HttpContext.User.

Jedna z možností je autentifikace pomocí cookies. V aplikaci založené na cookies autentifikace umožňuje použití existující identifikace uživatele pro připojení SignalR. Při použití klienta prohlížeče nejsou potřeba žádné další konfigurace. Pokud je uživatel přihlášen v aplikaci, tak připojení SignalR automaticky zdědí toto ověření. Cookies je způsob posílání přístupových tokenů. Posílat tokeny mohou i klienti bez prohlížeče. Při použití .NET klienta lze vlastnosti Cookies nakonfigurovat ve volání .WithUrl tak, aby poskytoval cookie. Použití ověřování pomocí cookies z .NET klienta vyžaduje, aby aplikace poskytovala API k výměně ověřovacích dat za cookies. [46]

Další z možností je autentifikace pomocí nosiče tokenu. Klient může použít přístupový token místo cookies. Server validuje token a použije ho k identifikaci uživatele. Toto ověření

se provádí pouze při navazování připojení. Během životnosti připojení se server automaticky znovu nevaliduje, aby zkontroloval zrušení tokenu. Ve standardních webových API se tokeny posílají v hlavičce HTTP požadavků. SignalR není schopen nastavit tyto hlavičky v prohlížečích při použití některých transportů. Při použití WebSockets a Server-Sent eventů se token přenáší jako parametr řetězce dotazu.

Cookies jsou specifické pro prohlížeče. Odesílání cookies z jiných druhů klientů přidává na složitosti ve srovnání s odesíláním tokenů. Z tohoto důvodu se ověřování pomocí souboru cookies nedoporučuje, pokud jsou použity jiní klienti než klient prohlížeče [46]. Při použití jiných klientů, než klienta prohlížeče je doporučený přístup ověřování pomocí tokenů.

Pokud je v aplikaci nastavené Windows ověřování, SignalR pak toto ověření může použít k vlastnímu zabezpečení. Pokud je ale potřeba odesílat zprávy jednotlivým klientům, pak je potřeba přidání vlastního poskytovatele ID uživatele. Ověřovací systém Windows neposkytuje identifikátor jména. SignalR používá deklaraci identity (claim) k určení uživatelského jména. Windows ověřování je podporováno v Internet Exploreru a Microsoft Edge, ale například v Chromu a Safari není podporováno. Zkoušení Windows ověřování nebo WebSockets selže v těchto prohlížečích. Klient se pokusí přejít zpět na jiné transporty, které by mohly fungovat. [46]

3.2 Zabezpečení obsahu

Cross-Site Scripting (XSS) je chyba zabezpečení, kdy útočník umístí jeden nebo více škodlivých skriptů na straně klienta do vykresleného obsahu aplikace. Tomuto se snaží zabránit zásady zabezpečení obsahu (Content Security Policy). CSP pomáhá chránit před útoky XSS informováním prohlížeče o platných [44]:

- Zdrojích pro načtený obsah, včetně skriptů, šablon stylů a obrázků.
- Akcích provedených stránkou, kde jsou zadané povolené cíle URL.
- Pluginech, které lze načíst.

Pro použití CSP, vývojář musí určit několik směrnic zabezpečení obsahu CSP v jedné nebo více záhlavích Content-Security-Policy nebo značkách <meta>. Při načítání stránky prohlížeč vyhodnocuje zásady. Prohlížeč zkontroluje zdroje stránky a určí, zda splňují požadavky směrnic o zabezpečeném obsahu. Když směrnice pro prostředek nejsou splněny, tak prohlížeč nenačte daný prostředek. Například pokud je ve směrnicích určeno, že nejsou povoleny skripty třetích stran, tak když stránka obsahuje <script> tag s původem třetí strany v atributu

src, tak prohlížeč zabrání načtení tohoto skriptu. CSV je podporováno ve většině moderních desktopových a mobilních prohlížečích, včetně Chrome, Edge, Firefox, Opera a Safari. CSP je doporučováno pro všechny Blazor aplikace.

3.3 Práce s databází

Pro Blazor WebAssembly je možný přístup k IndexedDB nebo Local storage skrze JSInterop, ale nedoporučuje se to, protože uživatelé můžou jakýkoliv klientský kód měnit a vidět. Miscrosoft zatím doporučuje mít Blazor WebAssembly s hostováním ASP .NET Core, kde už je možnost mít bezpečnou komunikaci s databází pomocí EF Core.

Pro Blazor Server se doporučuje Entity Framework Core [43]. Blazor Server je stavový aplikační framework. Aplikace udržuje stále připojení k serveru a stav uživatele je uchováván v paměti serveru. Jedním příkladem stavu uživatele jsou data uchovávaná v instancích závislostí (DI), která jsou vymezena na aplikaci. EF Core spoléhá na DbContext jako prostředek ke konfiguraci přístupu k databázi a jako hlavní prvek práce s databází. EF Core poskytuje rozšíření AddDbContext pro aplikace ASP.NET Core, které ve výchozím nastavení registruje kontext jako scoped službu. V Blazor Server aplikacích může být registrace scoped služeb velmi problematická, protože instance je sdílena napříč všemi komponentami v okruhu uživatele. DbContext není bezpečný pro vlákna a není stavěný na současné použití. Stávající životní běhy nejsou vhodné z těchto důvodů [40]:

- Singleton: sdílí stav napříč všemi uživateli aplikace a vede k nevhodnému souběžnému použití.
- Scoped: představuje podobný problém mezi komponentami pro stejného uživatele.
- Transient: má za následek to, že se vytvoří nová instance při každém požadavku. Jelikož komponenty mohou být dlouhodobé, můžou vést k delší životnosti kontextu, než je zamýšleno.

3.4 Používání gRPC s Blazorem

V Blazor WebAssembly, jako v jiných SPA založených na prohlížeči, je nejčastější výměna dat a spouštění funkcí na straně serveru pomocí JSON-over-HTTP. Klient odešle požadavek HTTP na nějakou předem dohodnutou adresu URL s předem dohodnutou HTTP metodou s daty poslanými pomocí JSON v předem dohodnutém tvaru. Poté server provede operaci a odpoví předem dohodnutým HTTP status kódem a daty v JSON.

Tento způsob většinou funguje dobře ale obsahuje dvě slabiny [41]:

- JSON je velmi podrobný datový formát. Není optimalizovaný pro šířku pásma.
- Neexistuje žádný mechanismus, který by zaručoval, že všechny předem dohodnuté podrobnosti adres URL, metod HTTP a stavových kódů jsou ve skutečnosti konzistentní mezi serverem a klientem.

Mechanismus gRPC [42] je Remote Procedure Call mechanismus původně vytvořený společností Google. Pro účely SPA aplikací to lze považovat jako alternativu JSON-over-HTTP. Přímo opravuje dvě výše zmíněné slabosti [41]:

- Je optimalizovaný pro minimální síťový provoz a odesílá efektivní binárně serializované zprávy.
- Lze při kompilaci garantovat, že server a klient souhlasí o tom, jaké endpointy existují, v jakém tvaru budou data odcházet a přicházet bez toho, aby se musely specifikovat adresy URL a návratové kódy.

Pro vytvoření gRPC služby se napíše soubor s příponou .proto, který je jazykově nezávislým popisem sady služeb RPC a jejich dat. Z tohoto souboru se může vygenerovat silně typované server a klient třídy, aby byla v době kompilace zaručena shoda s protokolem. Pak při běhu gRPC řeší serializaci dat a odesílá, přijímá zprávy v efektivním formátu (defaultně v protobuf).

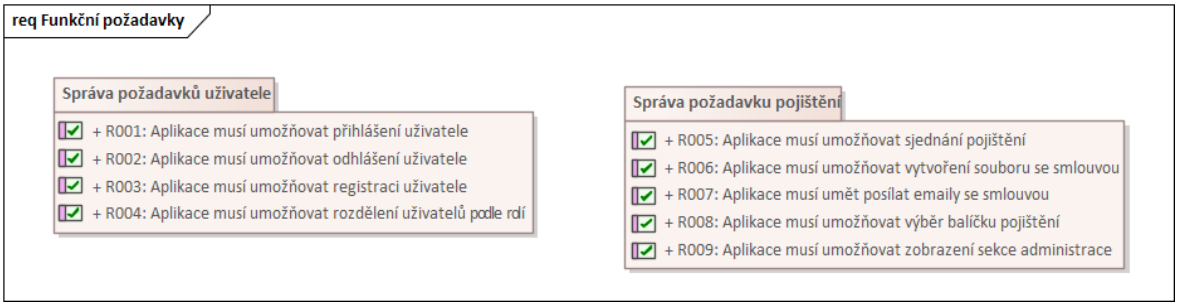
II. PRAKTICKÁ ČÁST

4 NÁVRH APLIKACE

Na začátku byly sestaveny požadavky, podle kterých se dále pracovalo. Požadavky byly inspirovány kalkulačkami pojištění, které jsou volně dostupné na webu. Hlavní funkce aplikace je vytvoření pojištění z údajů, které uživatel zadá do formuláře. Požadavky jsou rozděleny na funkční a nefunkční.

4.1 Funkční požadavky

V této kapitole jsou popsány funkční požadavky aplikace. Na obrázku číslo 5 můžeme vidět funkční požadavky aplikace.



Obrázek 5 Funkční požadavky

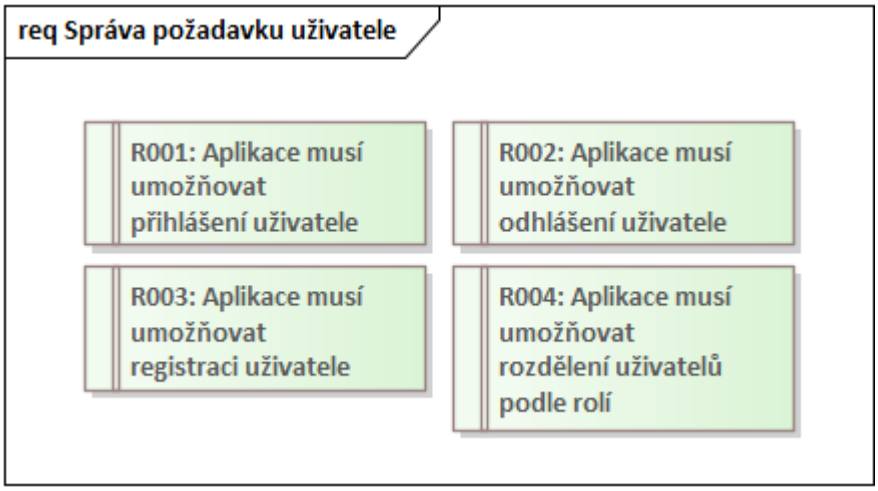
R001: Každý nepřihlášený uživatel se bude moci přihlásit do aplikace pomocí emailu a hesla.

R002: Každý přihlášený uživatel se bude moci odhlásit z aplikace.

R003: Každý neregistrovaný uživatel si bude moci vytvořit účet pomocí registračního formuláře.

R004: Systém musí obsahovat rozdělení uživatelů podle rolí. Výchozí role jsou: User, Admin

Na obrázku číslo 6 můžeme vidět správu požadavků uživatele.



Obrázek 6 Správa požadavků uživatele

R005: Aplikace musí umožňovat sjednání pojištění pro uživatele.

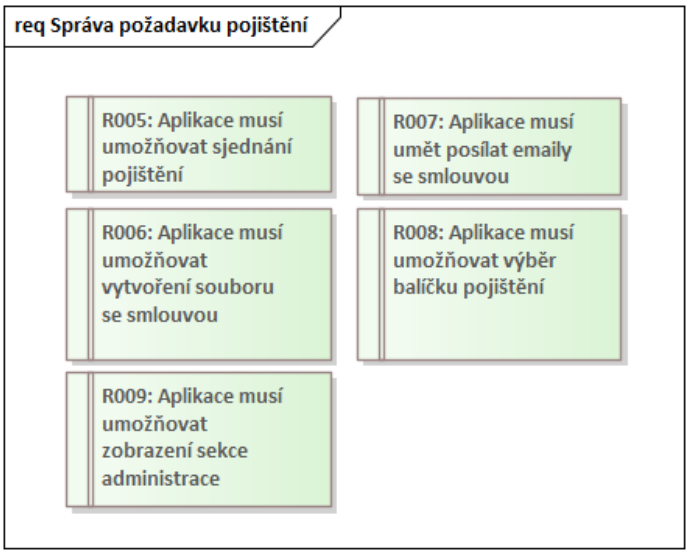
R006: Aplikace musí umožňovat vytvoření souboru se smlouvou.

R007: Aplikace musí umět posílat email se smlouvou na emailovou adresu, kterou zadá uživatel.

R008: Aplikace musí umožňovat uživateli výběr balíčku pojištění. V aplikaci budou vždycky zobrazovány 2 balíčky na pojištění.

R009: Aplikace musí umožňovat zobrazení sekce administrace. Tato sekce bude přístupná pouze uživatelům s roli Administrátor.

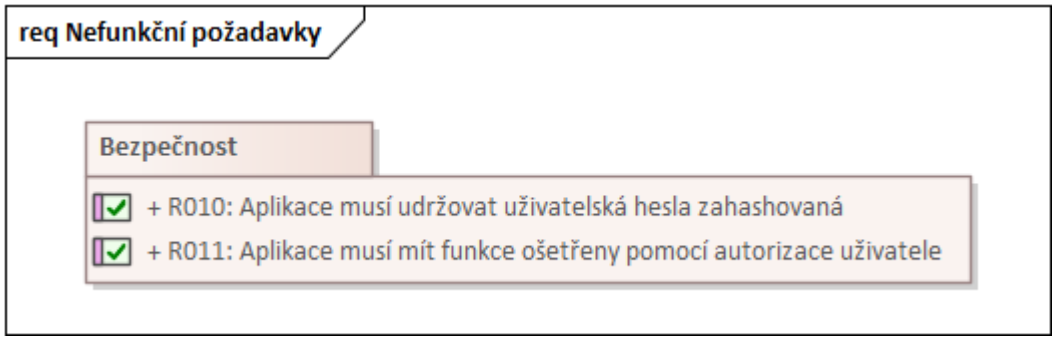
Na obrázku číslo 7 můžeme vidět Správu požadavků pojištění.



Obrázek 7 Správa požadavků pojištění

4.2 Nefunkční požadavky

V této kapitole jsou popsány nefunkční požadavky vytvořené aplikace. Na obrázku číslo 8 můžeme vidět nefunkční požadavky aplikace.



Obrázek 8 Nefunkční požadavky

R010: Aplikace musí ukládat uživatelská hesla do databáze hashovaná.

R011: Funkce systému musejí být ošetřeny podle role uživatele. Sekce administrace musí být přístupná pouze uživatelům s rolí administrátora. Pojištění bude přístupné pro všechny registrované uživatele.

4.3 Scénáře použití

V této kapitole jsou popsány hlavní scénáře použití aplikace.

Scénář: Sjednání pojištění

Tento scénář popisuje případ, kdy uživatel sjednává pojištění v aplikaci.

Jméno scénáře	Sjednání pojištění
Zúčastnění aktéři	Nepřihlášený uživatel
Hlavní scénář	1. Use Case začíná, když uživatel klikne na pojištění vozidla. 2. Uživatel se přihlásí. 3. Přihlášený uživatel vyplní základní údaje. 4. Systém zkontroluje zadané údaje. 5. Systém zobrazí balíčky pojištění 6. Uživatele vybere balíček. 7. Uživatel vyplní dodatečné údaje. 8. Systém sjedná pojištění
Vedlejší scénář	2a. Uživatel zadal špatné údaje. 4a. Uživatel nezadal údaj nebo vyplnil chybně údaj ve formuláři.

Tabulka 1 Scénář – Sjednání pojištění

Alternativní scénář: Sjednání pojištění – Uživatel zadal špatné údaje

Tento alternativní scénář popisuje případ, kdy uživatel zadá špatné údaje do přihlašovacího formuláře, které neodpovídají žádnému již registrovanému účtu.

Jméno scénáře	Sjednání pojištění – Uživatel zadal špatné údaje
Alternativní scénář	1. Systém vyhodí chybovou hlášku a informuje uživatele o chybném zadání údajů.

Tabulka 2 Alternativní scénář – Uživatel zadal špatné údaje

Alternativní scénář: Sjednání pojištění – Uživatel nezadal údaj nebo vyplnil chybně údaj ve formuláři

Tento alternativní scénář popisuje případ, kdy uživatel zadal špatně údaj nebo nevyplnil údaj ve formuláři základních informací.

Jméno scénáře	Sjednání pojištění – Uživatel nezadal údaj nebo vyplnil chybně údaj ve formuláři
Alternativní scénář	1. Systém vyhodí chybovou hlášku u údaje a informuje uživatele o chybném zadání údajů nebo o jejich nevyplnění.

Tabulka 3 Alternativní scénář – Uživatel nezadal údaj nebo vyplnil chybně údaj ve formuláři

Scénář: Zobrazení sekce administrace

Tento scénář popisuje případ, kdy uživatel chce zobrazit sekci administrace

Jméno scénáře	Zobrazení sekce administrace
Zúčastnění aktéři	Administrátor
Hlavní scénář	1. Use Case začíná, když uživatel klikne na tlačítko administrace 2. Uživatel se přihlásí. 3. Systém zobrazí sekci administrace
Vedlejší scénář	2a. Uživatel zadal špatné údaje. 2b Uživatel nemá roli administrátor

Tabulka 4 Scénář – Zobrazení sekce administrace

Alternativní scénář: Zobrazení sekce administrace – Uživatel zadal špatné údaje

Tento alternativní scénář popisuje případ, kdy uživatel zadá špatné údaje do přihlašovacího formuláře, které neodpovídají žádnému již registrovanému účtu.

Jméno scénáře	Zobrazení sekce administrace – Uživatel zadal špatné údaje
Alternativní scénář	1. Systém vyhodí chybovou hlášku a informuje uživatele o chybném zadání údajů.

Tabulka 5 Alternativní scénář – Uživatel zadal špatné údaje

Alternativní scénář: Zobrazení sekce administrace – Uživatel nemá roli administrátora

Tento alternativní scénář popisuje případ, kdy uživatel se přihlásil, ale nemá roli administrátora.

Jméno scénáře	Zobrazení sekce administrace – Uživatel nemá roli administrátora
Alternativní scénář	1. Systém vyhodí chybovou hlášku a informuje uživatele o tom, že nemá dostatečná práva k zobrazení sekce administrace

Tabulka 6 Alternativní scénář – Uživatel nemá roli administrátora

Scénář: Registrace uživatele

Tento scénář popisuje funkci pro registraci nového uživatele do aplikace. Neregistrovaný uživatel musí vyplnit všechny údaje v registračním formuláři a vyplněné údaje musí splňovat pravidla údajů.

Jméno scénáře	Registrace uživatele
Zúčastnění aktéři	Neregistrovaný uživatel
Hlavní scénář	1. Use Case začíná, když uživatel klikne na tlačítko „Registrovat“. 2. Uživatel vyplní registrační formulář. 3. Uživatel potvrdí registraci. 4. Systém zkontroluje správnost zadaných údajů. 5. Systém vytvoří nového uživatele.
Vedlejší scénář	2a. Údaje byly zadány chybně. 2b Údaje jsou již použity pro jiný účet.

Tabulka 7 Scénář – Registrace uživatele

Alternativní scénář: Registrace uživatele – Údaje byly zadány chybně.

Tento alternativní scénář popisuje případ, kdy uživatel zadá špatné údaje do registračního formuláře. Například nesplňuje pravidla pro hesla.

Jméno scénáře	Registrace uživatele – Údaje byly zadány chybně
Alternativní scénář	1. Systém vyhodí chybovou hlášku a informuje uživatele o chybném zadání údajů.

Tabulka 8 Alternativní scénář – Údaje byly zadány chybně

Alternativní scénář: Registrace uživatele – Údaje jsou již použity pro jiný účet.

Tento alternativní scénář popisuje případ, kdy uživatel vyplnil údaje, které již někdo používá.

Jméno scénáře	Registrace uživatele – Údaje jsou již použity pro jiný účet
Alternativní scénář	1. Systém vyhodí chybovou hlášku a informuje uživatele o tom, že údaje už někdo používá.

Tabulka 9 Alternativní scénář – Údaje jsou již použity pro jiný účet

Scénář: Přihlášení uživatele

Tento scénář popisuje případ, kdy se chce uživatel přihlásit do aplikace.

Jméno scénáře	Přihlášení uživatele
Zúčastnění aktéři	Neregistrovaný uživatel
Hlavní scénář	1. Use Case začíná, když uživatel klikne na tlačítko „Přihlásit se“. 2. Uživatel vyplní přihlašující formulář 3. Systém přihlásí uživatele do aplikace
Vedlejší scénář	3a. Zadaný účet neexistuje

Tabulka 10 Scénář – Přihlášení uživatele

Alternativní scénář: Přihlášení uživatele – Zadaný účet neexistuje.

Tento alternativní scénář popisuje případ, kdy uživatel vyplnil účet, který neexistuje v aplikaci.

Jméno scénáře	Přihlášení uživatele – Zadaný účet neexistuje
Alternativní scénář	1. Systém vyhodí chybovou hlášku a informuje uživatele o tom, že tento účet neexistuje.

Tabulka 11 Alternativní scénář – Zadaný účet neexistuje

5 KLÍČOVÉ PRVKY APLIKACE

V následujících kapitolách je popsáno založení projektu a klíčové prvky vytvořené aplikace. V kapitolách jsou obsaženy ukázky kódu klíčových prvků aplikace. Klíčové prvky jsou Autorizace uživatele, Komponenty, Uchovávání dat skrze stránky, Validace vstupů formuláře, Práce s databází, Posílání emailu se smlouvou a Využití sdíleného kódu pro Client a Server projekt.

5.1 Založení projektu a výběr šablon

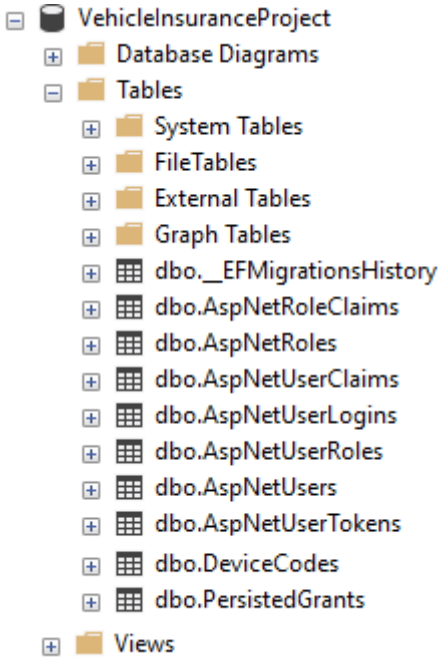
Framework Blazor nabízí dvě hlavní šablony projektu, a to Blazor WebAssembly a Blazor Server. Jelikož autor chtěl ukázat možnosti WebAssembly projektu, tak si vybral šablonu Blazor WebAssembly. U Blazor WebAssembly projektu autor vybral ASP .NET Core hostování. Toto hostování znamená, že se v projektu vytvoří celkem tři podprojekty. Client-side WebAssembly podprojekt, server-side ASP .NET Core podprojekt a SHARED podprojekt. Dále autor při vytváření projektu specifikoval, že chce využít šablonu pro autorizaci uživatele Microsoft identity platform. Před vytvoření projektu bylo potřeba stáhnout .NET SDK balíček a Visual Studio 2019.

5.2 Autorizace uživatele

Tato kapitola se zabývá vytvořením funkce autorizace uživatele a využíváním rolí uživatele k omezení přístupu k jednotlivým částem aplikace. Pro základ autorizace uživatele byla použita šablona Microsoft identity platform. Po vytvoření projektu bylo nejprve potřeba nastavit databázi pro Entity framework. To se provedlo tak, že se nastavil DefaultConnection v souboru appsettings.json v projektu Server.

```
"ConnectionStrings": {
    "DefaultConnection": "Server=DESKTOP-
7EUGMO5;Database=VehicleInsuranceProject;Trusted_Connection=True;MultipleActiveResul
tSets=true"
  },
```

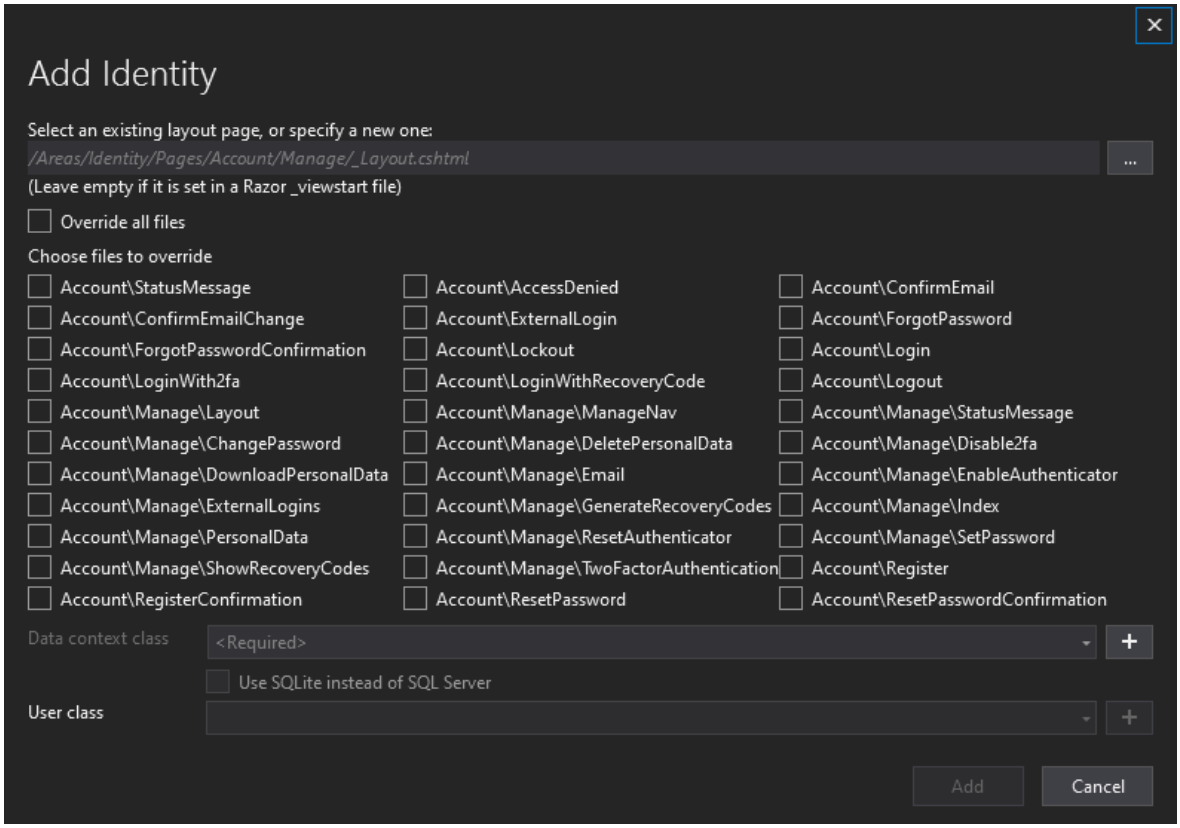
Pro snadné manipulování databáze autor použil program Microsft SQL server management studio. Pro prvotní publikování databáze bylo potřeba do package manager terminálu napsat příkaz Update-Database. Výsledkem tohoto příkazu je databázová struktura, která lze vidět na obrázku 9.



Obrázek 9 Struktura databáze po publikování databáze

Nejdůležitější tabulka pro identifikaci uživatele je AspNetUsers, která obsahuje jednotlivé registrované uživatele. Tato tabulka obsahuje kromě sloupců s informacemi o uživateli, jako například UserName, Phone, PasswordHash a Email i takové sloupce, jako například SecurityStamp a ConcurrencyStamp, které slouží k zabránění nedovoleného přístupu k aplikaci. SecurityStamp je razítko uživatelských údajů, které slouží k odhlášení uživatelů po změně hesla. Pokud je do aplikace přihlášeno více uživatelů pod jedním účtem a jeden účet změní heslo, tak ostatní okamžitě nemají přístup do aplikace, protože nebude souhlasit SecurityStamp. ConcurrencyStamp je razítko, které slouží proti současnému přístupu do databáze.

Pro úpravu jednotlivých endpointů autentizace uživatele je potřeba nad projektem zavolat funkci Scaffolding. To se udělá tak, že se klikne pravým tlačítkem myši na Server projekt, a vybere se Add->New Scaffolded Item->Identity. Visual studio dá programátorovi vybrat, jaké endpointy se mají přidat. Některé endpointy není třeba vůbec přidávat do aplikace, pokud se nechtějí používat. Například endpoint pro potvrzení změny emailu nebo endpoint pro přihlášení pomocí třetích stran. Na obrázku číslo 10 lze vidět nabídku endpointů.



Obrázek 10 Nabídka scaffoldovaných endpointů.

Po vybrání endpointů se vytvoří jednotlivé stránky pod cestou Areas/Identity/Pages.

5.2.1 Nastavení pravidel pro hesla

Nastavení pravidel pro heslo uživatele se nastavuje v souboru Startup.cs v projektu Server. Zde se specifikuje nastavení služby Identity pomocí funkce `AddDefaultIdentity`. U hesla lze nastavit, jestli je vyžadována číslice, znak odlišný od písmenka nebo velké písmeno. Dále zde lze nastavit, jestli je vyžadované potvrzení uživatele pomocí emailu. Ve výchozím stavu jsou tyto pravidla nastaveny na hodnotu true. Autor se rozhodl využít všech těchto pravidel k dosažení co nejbezpečnějšího hesla.

```
services.AddDefaultIdentity<ApplicationUser>(options => {
                options.Password.RequireDigit = true;
                options.Password.RequireNonAlphanumeric = true;
                options.Password.RequireUppercase = true;
                options.SignIn.RequireConfirmedAccount = false;
            })
           .AddRoles<IdentityRole>()
           .AddEntityFrameworkStores<ApplicationDbContext>();
```

5.2.2 Nastavení autorizace uživatele stránky

Pomocí Identity je možné nastavení pravidel přístupu na stránky. Stránku lze nastavit tak, aby byl přístup možný pouze přihlášeným uživatelům pomocí atributu `@attribute [Authorize]`. Tento atribut se doporučuje dávat nad HTML část stránky. Pokud je potřeba omezit přístup na stránku dle role uživatele, tak se do atributu Authorize přidá výčet povolených rolí. Před využíváním atributu Authorize k omezení přístupu uživatelů dle rolí, je nejprve potřeba specifikovat, že se do UserClaims budou ukládat informace o roli uživatele. Toto nastavení se přidá do souboru Startup.cs do funkce konfigurace služeb ConfigureServices.

```
services.AddIdentityServer()
     .AddApiAuthorization<ApplicationUser, ApplicationDbContext>(options => {
                    options.IdentityResources["openid"].UserClaims.Add("role");
                    options.ApiResources.Single().UserClaims.Add("role");
                });
```

Autor omezuje přístup do sekce administrace pouze na uživatele, kteří mají roli Admin. Toto omezení bylo dosaženo pomocí následujícího atributu.

```
@attribut [Authorize(Roles = "Admin")]
```

Pokud je potřeba zobrazit část uživatelského rozhraní stránky pouze uživatelům s roli Admin, tak se využije komponenta autorizace AuthorizeView. Díky této komponentě lze vytvářet dynamický obsah podle přihlášeného uživatele.

```
<AuthorizeView Roles="Admin">
    <p>You can only see this if you're in the Admin role.</p>
</AuthorizeView>
```

5.2.3 Zabezpečení jednotlivých endpointů Server projektu

Pomocí atributu Authorize lze zabezpečit jednotlivé endpointy Server projektu. Atribut Authorize s nastaveným omezením se přidá nad celým controllerem nebo nad jednotlivým endpointem controlleru. Následuje ukázka kódu controlleru sekce administrace z ukázkové aplikace. Tento controller je přístupný pouze pro uživatele, kteří mají roli Admin.

```
[Authorize(Roles = "Admin")]
[ApiController]
[Route("api/[controller]")]
public class AdministrationController : Controller
{
    private readonly ApplicationDbContext _context;
    public IVehicleInsuranceUtils _vehicleUtils { get; set; }
    public AdministrationController(ApplicationDbContext context,
           IVehicleInsuranceUtils vehicleUtils)
    {
        _context = context;
        _vehicleUtils = vehicleUtils;
    }

    [HttpGet("[action]")]
    public IActionResult GetData()
    {
        var model = new AdministrationData();
        model.Contracts = _vehicleUtils.GetContracts(_context.Contract.Include(x =>
       x.Client).Include(x => x.Vehicle).ToList());
        return Ok(model);
    }
}
```

5.3 Komponenty

Komponenty jsou základním stavebním blokem Blazor aplikace. Jsou to nezávislé části uživatelského rozhraní. Komponenty obsahují svoji logiku chování umožňující dynamické chování uživatelského rozhraní. Komponenty mohou být znovu použity, vnořeny a sdíleny mezi projekty a stránky. Komponenty používají Razor syntaxi. Dvě hlavní funkce, které jsou často využívány v komponentách jsou direktivy a atributy direktiv.

5.3.1 Komponenta ProductTypeComponent

Autor používá komponenty například k vytvoření přepínače pojištění. Tento přepínač je tvořen ze tří tlačítek. Každé tlačítko má svůj vlastní obrázek a název. Po kliknutí na jedno z tlačítek je vybrána výsledná možnost, přiřadí se css třída s výrazněním a hodnota se propíše zpět do modelu. Aktuální vybraná hodnota se zjišťuje z parametru model. Tento parametr je typu enum InsuranceType. U tohoto enumu jsou specifikovány atributy Description pro pozdější překlad hodnoty enumu do češtiny. Následuje ukázka kódu enumu typu pojištění s atributy description.

```
public enum InsuranceType
{
        /// <summary>
        /// Povinné ručení
        /// </summary>
        [Description("Povinné ručení")]
        LiabilityInsurance = 0,
        /// <summary>
        /// Havarijní pojištění
        /// </summary>
        [Description("Havarijní pojištění")]
        CollisonDamageInsurance = 1,
        /// <summary>
        /// Povinne ručení + Havarijní pojištění
        /// </summary>
        [Description("Povinne ručení + Havarijní pojištění")]
        VehicleInsurance = 2
}
```

Specifikování parametrů komponent je obsaženo v bloku @code, kde se píše C# kód. Pro specifikování proměnné jako parametru komponenty slouží atribut [Parameter]. Propisování změn komponenty do modelu je dosaženo pomocí třídy EventCallback. U této třídy se specifikuje vracející typ na enum InsuranceType a poté se nastaví na jednotlivé kliknutí tlačítek funkce InvokeAsync s vybraným typem pojištění. Specifické styly komponent se dávají do bloku <style>. Následuje ukázka komponenty ProductTypeComponent.

```
<div>
    <button type="button" class="btn button-box @(model == InsuranceType.Liability-
    Insurance ? "btn-warning" : "btn-light")"
    @onclick="@(() => OnClick.InvokeAsync(InsuranceType.LiabilityInsurance))">
        <img src="icons/car-pov.png" alt="Image" style="width: 70px;height:70px;">
        Povinné ručení
    </button>
    <button type="button" class="btn button-box @(model == InsuranceType.CollisonDa
mageInsurance ? "btn-warning" : "btn-light")"
    @onclick="@(() => OnClick.InvokeAsync(InsuranceType.CollisonDamageInsurance))">
        <img src="icons/car-hav.png" alt="Image" style="width: 70px;height:70px;">
        Havarijní pojištění
    </button>
    <button type="button" class="btn button-box @(model == InsuranceType.VehicleIn-
surance ? "btn-warning" : "btn-light")"
    @onclick="@(() => OnClick.InvokeAsync(InsuranceType.VehicleInsurance))">
        <img src="icons/car-povhav.png" alt="Image"
            style="width: 70px;height:70px;">
        Komplexní autopojištění
    </button>
</div>
```

```
@code {
    [Parameter]
    public InsuranceType model { get; set; }

    [Parameter]
    public EventCallback<InsuranceType> OnClick { get; set; }
}

<style>
    .button-box {
        width: 150px;
        height: 150px;
        border-color: #d3d3d3 !important;
    }
</style>
```

Zavolání komponenty na stránce je dosaženo vložením objektu komponenty do HTML části stránky. U komponenty je potřeba specifikovat návratovou funkci OnClick. V této funkci se zapíše vybraný typ pojištění do modelu stránky.

```
<ProductTypeComponent model="SimulationInput.InsuranceType" OnClick="ClickHandler">

</ProductTypeComponent>

public void ClickHandler(InsuranceType clickedResult)
{
    SimulationInput.InsuranceType = clickedResult;
}
```

Na obrázku číslo 11 můžeme vidět výslednou podobu komponenty.

Obrázek 11 Komponenta ProductTypeComponent

5.3.2 Komponenta s našeptáváním aut

Framework Blazor zatím neobsahuje komponentu pro vstup uživatele s našeptáváním hodnot, a proto pro získání značky a modelu vozidla klienta je využita komponenta Blazored.Typeahead. U této komponenty je potřeba specifikovat proměnnou, na kterou se váže tato komponenta v parametru @bind-Value. Dále je potřeba specifikovat název funkce pro hledání hodnot v našeptávači v parametru SearchMethod a zástupnou hodnotu vstupu Placeholder.

V této komponentě lze specifikovat celkem pět šablon a z toho dvě šablony je nutné specifikovat. Mezi povinné šablony patří šablona výsledku ResultTemplate a šablona vybraného prvku SelectedTemplate. U obou těchto šablon je nutné specifikovat parametr Context. Mezi nepovinné šablony patří šablona nenalezené hodnoty NotFoundTemplate, šablona nesplněné minimální délky hodnoty HelpTemplate a šablona zápatí našeptávače hodnot FooterTemplate. Následuje ukázka kódu využití komponenty Typeahead pro získání značky a modelu.

```
<div class="input-block">
     <div class="inline-title">Značka</div>
          <div class="inline-input">
               <div class="input-width inline">
                    <BlazoredTypeahead SearchMethod="SearchManufacturers"
                                       @bind-Value="SimulationInput.Vehicle.Brand"
                                       Placeholder="Zadejte název značky">
                        <SelectedTemplate Context="brand">
                            @brand
                        </SelectedTemplate>
                        <ResultTemplate Context="brand">
                            @brand
                        </ResultTemplate>
                    </BlazoredTypeahead>
                </div>
                <div class="inline">
                  <ValidationMessage For="@(() => SimulationInput.Vehicle.Brand)" />
                </div>
            </div>
        </div>
<div class="input-block">
     <div class="inline-title">Model</div>
          <div class="inline-input">
               <div class="input-width inline">
                    <BlazoredTypeahead Disabled="SimulationInput.Vehicle.IsBrand-
NullOrEmpty()"
                                       SearchMethod="SearchModels"
                                       @bind-Value="SimulationInput.Vehicle.Model"
                                       Placeholder="Zadejte název modelu">
                        <SelectedTemplate Context="model">
                            @model
                        </SelectedTemplate>
                        <ResultTemplate Context="model">
                            @model
                        </ResultTemplate>
                    </BlazoredTypeahead>
                </div>
                <div class="inline">
                  <ValidationMessage For="@(() => SimulationInput.Vehicle.Model)" />
                </div>
            </div>
</div>
```

Funkce pro získání napovídajících značek aut je vytvořena pomocí Linq knihovny a Contains funkce. U vstupu modelu auta je specifikovaný parametr Disabled, který znemožňuje vyplnění tohoto vstupu, když nebude vyplněná značka vozidla. Následuje ukázka vyhledávacích funkcí pro získání značky a modelu vozidla.

```
private async Task<IEnumerable<string>> SearchManufacturers(string searchText)
{
    return await Task.FromResult(Manufacturer.Where(x => x.ToLower().Contains(sear-
chText.ToLower())).ToList());
}

private async Task<IEnumerable<string>> SearchModels(string searchText)
{
    if (searchText != null && searchText != "")
    {
        if (Models.ContainsKey(SimulationInput.Vehicle.Brand))
        {
            return await Task.FromResult(Models[SimulationInput.Vehi-
cle.Brand].Where(x => x.ToLower().Contains(searchText.ToLower())).ToList());
        }
    }
    return await Task.FromResult(new List<string>());
}
```

5.4 Uchování dat skrze stránky

Pro uchování dat mezi jednotlivými stránkami se autor rozhodl použít Session Storage. Session Storage je prohlížečové uložiště, kde si klienti mohou ukládat data. Díky Session Storage jsou data ukládány v rámci jedné relace prohlížeče. Jelikož Blazor zatím nepodporuje přímou manipulaci se Session Storage, autor využil balíčku `Blazored.SessionStorage`, který je s otevřeným přístupem ke kódu. Pro použití Session Storage na stránce je potřeba nejprve přidat závislost na službu `ISyncSessionStorageService`. Přidání závislostí na stránku se dělá pomocí funkčního slova *@inject*. Přidání závislosti na službu `ISyncSessionStorageService` vypadá následovně:

```
@inject Blazored.SessionStorage.ISyncSessionStorageService sessionStorage
```

Po přidání závislosti lze ukládat nebo načítat model stránky z uložiště Session Storage. To se provádí pomocí funkcí GetItem a SetItem. U funkce SetItem se specifikuje proměnná, která se má uložit a název, pod kterým to bude uloženo. Následuje ukázka uložení modelu SimulationInput do proměnné vehiclemodel v Session Storage.

```
sessionStorage.SetItem("vehiclemodel", SimulationInput);
```

U funkce GetItem se specifikuje proměnná Session Storage, ze které se má model načíst a typ proměnné. Načtení modelu se většinou provádí po přechodu na stránku. To se docílí přepsáním funkce OnInitializedAsync klíčovým slovem override. Následuje ukázka načtení modelu ze Session Storage druhého kroku pojištění auta aplikace:

```
protected override async Task OnInitializedAsync()
{
    SimulationInput = sessionStorage.GetItem<SimulationInput>("vehiclemodel");
    var response =
    await Http.PostAsJsonAsync("api/VehicleInsurance/SimulateAll", SimulationInput);
    SimulationInput.SimulationAllResult =
       await response.Content.ReadFromJsonAsync<SimulationAllResult>();
}
```

5.5 Validace vstupů formuláře

Validaci vstupů formuláře lze udělat v Blazoru více způsoby. Přímo Blazor podporuje validaci formuláře EditForm pomocí knihovny DataAnnotations. Autor se ale rozhodl použít validaci Fluent, jelikož lze použít jak pro validaci v Client projektu, tak i pro validaci Server projektu jednotlivých endpointů. To znamená, že se ušetří mnoho řádků kódu psaní definic validací. Dále se předejde chybám při změně definic validací. Jelikož jsou definice validací na jednom místě pro oba projekty, nenastane to, že se například zapomene upravit validace jednoho z projektů. Definice modelu je umístěna v Shared projektu, který je sdílený pro projekty Client a Server. Validátor třídy musí dědit od třídy `AbstractValidator`, který má v parametru typ validujícího modelu. Jednotlivá pravidla se specifikují v konstruktoru třídy. Ve výchozím stavu se pravidla provádí z vrchu dolů a provádí se všechny pravidla, i když některé neprojdou. Toto lze přenastavit pomocí validátoru Cascade. Následuje ukázka validátoru třídy Vehicle, kde jsou nastaveny pravidla jednotlivých proměnných této třídy.

```
public class VehicleValidator : AbstractValidator<Vehicle>
{
    public VehicleValidator()
    {
        RuleFor(Q => Q.Model)
            .NotEmpty()
               .WithMessage(ValidationErrorMessage.Required);
        RuleFor(Q => Q.Brand)
            .NotEmpty()
               .WithMessage(ValidationErrorMessage.Required);
        RuleFor(Q => Q.Performance)
            .NotEqual(0)
               .WithMessage(ValidationErrorMessage.Required);
        RuleFor(Q => Q.Weight)
            .NotEqual(0)
               .WithMessage(ValidationErrorMessage.Required);
        RuleFor(Q => Q.Volume)
            .NotEqual(0)
               .WithMessage(ValidationErrorMessage.Required);
        RuleFor(Q => Q.VIN)
            .NotEmpty()
               .WithMessage(ValidationErrorMessage.Required);
        RuleFor(Q => Q.SPZ)
            .NotEmpty()
               .WithMessage(ValidationErrorMessage.Required);}}
```

Pokud je potřeba validovat složitou třídu skládající se z více tříd, tak se v pravidlech využije funkce SetValidator, která nastaví validátor podtřídy modelu. Jelikož se definice validátoru nastavuje v konstruktoru, tak je možné si poslat v parametrech konstruktoru data, které se využijí v pravidlech. Následuje ukázka validátoru složitější třídy SimulationInput, která obsahuje třídu Vehicle a třídu Client.

```
public class SimulationInputValidator : AbstractValidator<SimulationInput>
{
    public SimulationInputValidator()
    {
        RuleFor(Q => Q.Vehicle).SetValidator(new VehicleValidator());
        RuleFor(Q => Q.Client).SetValidator(new ClientValidator(false));
    }
}
```

Před voláním Fluent validací na formuláři je nejprve potřeba zaregistrovat validátory ve třídě Program.cs projektu Client. Ve třídě Program.cs je asynchronní statická funkce Main, kde se registrují služby. Pomoci metody AddTransient zaregistrujeme validátory do služeb tak, aby se vracela pokaždé nová instance. U této funkce se nejprve specifikuje třída, která se validuje a třída validátoru. Následuje ukázka registrace validátoru pro třídu SimulationInput a ContractInput.

```
builder.Services.AddTransient<IValidator<SimulationInput>,    SimulationInputValida-
tor>();

builder.Services.AddTransient<IValidator<ContractInput>, ContractInputValidator>();
```

Dále je potřeba přidat komponentu FluentValidator do formuláře EditForm. Na formuláři EditForm lze nastavit, co se má stát, pokud uživatel potvrdí formulář. Na komponentě EditForm je atribut OnValidSubmit, ve kterém se specifikuje, jaká funkce se má zavolat, když formulář projde validací. Tato funkce musí existovat v @code bloku stránky. Dále se musí u každého vstupu specifikovat, na jakou proměnnou modelu je navázaný pomocí atributu @bind-Value. Komponenta FluentValidator se postará o to, že se jednotlivé proměnné budou validovat podle pravidel registrovaného Fluent validátoru. Dále FluentValidator zajišťuje správné přiřazení chybových hlášek k jednotlivým vstupům. Pokud je potřeba zobrazovat jednotlivé chybové hlášky někde jinde, nebo je potřeba změnit styl chybových hlášek, tak to lze specifikovat pomocí komponenty ValidationMessage. U této komponenty je potřeba specifikovat v atributu For proměnnou modelu, ke které se tato komponenta vztahuje. Následuje ukázka formuláře prvního kroku pojištění auta.

```
<EditForm Model="@SimulationInput" OnValidSubmit="@HandleValidSubmit">
     <FluentValidator></FluentValidator>
     <ProductTypeComponent model="SimulationInput.InsuranceType" OnClick="ClickHan-
dler">
     </ProductTypeComponent>
     <div class="input-block">
          <div class="inline-title">Frekvence platby</div>
          <div class="inline-input">
               <InputSelect class="form-control input-width" @bind-Value="Simulatio-
nInput.PaymentFrequency">
              @foreach (Enum value in Enum.GetValues(typeof(PaymentFrequencyType)))
              {
                <option value="@value">@GetDescriptionFromEnumValue(value)</option>
              }
              </InputSelect>
          </div>
      </div>
      <div class="input-block">
          <div class="inline-title">Počátek pojištění</div>
          <div class="inline-input">
               <InputDate class="form-control input-width" @bind-Value="Simulatio-
nInput.StartDate" min="@min" max="@max" />
          </div>
      </div>

      <button type="submit">Dále</button>

</EditForm>
```

5.6 Práce s databází

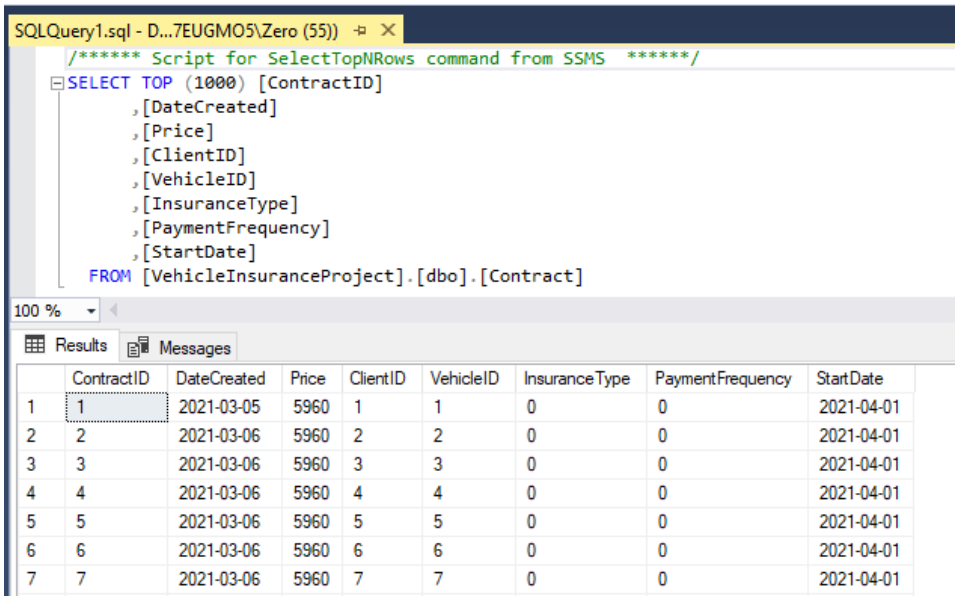
Autor používá doporučený způsob komunikace s databází Entity framework Core. Entity framework Core používá třídu DbContext ke konfiguraci databáze a jako přístup k databázi. Pro Blazor WebAssembly se nedoporučuje implementovat přístup do databáze, protože aplikace běží na straně klienta a může tedy vidět kód aplikace. Autor využívá ASP.NET Core Server projekt k přístupu do databáze. V Server projektu je vytvořena třída InsuranceProjectContext, která dědí od třídy DbContext. Pomocí této třídy lze získávat a přidávat záznamy v databázi. Autor používá tuto třídu k ukládání a získávání informací o sjednaných pojištění. V této třídě je specifikovaný model smlouvy ContractDTO, který obsahuje informace o pojištění.

```
public class InsuranceProjectContext : DbContext
{
   public InsuranceProjectContext(DbContextOptions<InsuranceProjectContext> options)
        :   base(options)
   {
   }

   public DbSet<ContractDTO> Contract { get; set; }
}
```

Třída ContractDTO musí obsahovat označení z DataAnnotations a DataAnnotations.Schema. Pro označení toho, že třída se má vázat na určitou tabulku, se používá atribut Table, kde se v parametru určí název tabulky. Dále se musí u třídy označit proměnná, která odpovídá primárnímu klíči tabulky, pomocí atributu Key. Následuje ukázka třídy ContractDTO, která se páruje na tabulku Contract v databázi.

```
[Table("Contract")]
public class ContractDTO
{
    [Key]
    public int ContractID { get; set; }
    public int Price { get; set; }
    public ClientDTO Client { get; set; }
    public VehicleDTO Vehicle { get; set; }
    public InsuranceType InsuranceType { get; set; }
    public PaymentFrequencyType PaymentFrequency { get; set; }
    public DateTime StartDate { get; set; }
}
```



	ContractID	DateCreated	Price	ClientID	VehicleID	InsuranceType	PaymentFrequency	StartDate
1	1	2021-03-05	5960	1	1	0	0	2021-04-01
2	2	2021-03-06	5960	2	2	0	0	2021-04-01
3	3	2021-03-06	5960	3	3	0	0	2021-04-01
4	4	2021-03-06	5960	4	4	0	0	2021-04-01
5	5	2021-03-06	5960	5	5	0	0	2021-04-01
6	6	2021-03-06	5960	6	6	0	0	2021-04-01
7	7	2021-03-06	5960	7	7	0	0	2021-04-01

Obrázek 12 Tabulka Contract v databázi

Získávání dat v Client projektu se provádí pomocí Get požadavku na Server projekt. Například v ukázkové aplikaci je sekce Administrace, kde se zobrazují informace o sjednaných smlouvách. Ve funkci inicializace stránky OnInitializedAsync se pošle Get požadavek na endpoint api/Administration/GetData. Tento Get požadavek je směrován na funkci GetData controlleru AdministrationController, kde se z kontextu získají data o smlouvách z databáze. Pokud jsou v získávaném modelu další třídy odpovídajícím tabulkám z databáze, tak je

nutné je zahrnout do získávaného modelu pomocí funkce Include. Následuje ukázka funkce pro získání smluv GetData.

```
[HttpGet("[action]")]
public IActionResult GetData()
{
    var model = new AdministrationData();
    model.Contracts = _vehicleUtils.GetContracts(
          _context.Contract.Include(x => x.Client)
                           .Include(x => x.Vehicle).ToList());
    return Ok(model);
}
```

Pro přidání záznamů do databáze se také používá kontext. Nejprve se přidá záznam do kontextu a potom pomocí funkce SaveChanges se aktualizuje databáze. Následuje ukázka kódu funkce uložení smlouvy do databáze, kde se nejprve vstup z Client aplikace namapuje na databázový model, a ten se pak vloží pomocí kontextu do databáze. Kontext databáze se dostane do třídy VehicleInsuranceService pomocí vkládání závislostí (dependency injection).

```
private void SaveContractToDB(ContractInput contractInput)
{
    var contract = _vehicleUtils.MapContractInputToContractDTO(contractInput);
    _context.Contract.Add(contract);
    _context.SaveChanges();
}
```

5.7 Vytvoření PDF souboru smlouvy

Vytvoření PDF v ASP .NET Core projektu jde mnoha způsoby. Existuje spousta placených i neplacených knihoven, které vytváří soubory typu PDF pomocí HTML. Autor se rozhodl použít knihovnu DinkToPdf, protože má obsáhlou dokumentace a je jednoduchá na implementování. Nejprve bylo potřeba nainstalovat NUGET balíček DinkToPdf do Server projektu. Dále bylo potřeba zaregistrovat službu SynchronizedConverter ve funkci ConfigureServices v souboru Startup.cs. Tato služba se zaregistruje s rozsahem Singleton. Singleton vrací pokaždé stejnou instanci služby.

```
services.AddSingleton(
     typeof(IConverter),
     new SynchronizedConverter(new PdfTools()));
```

Vytvořila se nová třída PDFGenerator a její interface IPDFGenerator. Jelikož autor potřeboval vytvářet dokumenty pouze v Server projektu, tak umístil tyto třídy do Server projektu. Kdyby bylo potřeba sdílet tuto službu i s Client projektem, tak by se umístila v Shared projektu. Dalším krokem bylo registrování těchto tříd jako službu s rozsahem Singleton.

```
services.AddSingleton<IPDFGenerator, PDFGenerator>();
```

Ve třídě PDFGenerator se vytvořila funkce CreateContractPDF, do které vstupuje třída ContractInput obsahující informace o smlouvě. Tato funkce vrací soubor ve formátu PDF jako byte pole. V této funkci bylo potřeba nejprve specifikovat nastavení požadovaného PDF souboru pomocí ObjectSettings a GlobalSettings. Lze například nastavit, jestli má být PDF barevné, velikost stránky, orientace stránky, kódování stránky. Hlavní tři nastavení, které se musí vyplnit jsou:

- HtmlContent – Obsah PDF souboru v HTML.
- UserStyleSheet – Cesta k souboru se styly CSS, které jsou použité na stránce.
- DocumentTitle – Název výsledného PDF souboru

Následuje ukázka funkce CreateContractPDF, která z HTML kódu vytvoří PDF soubor smlouvy v ukázkové aplikaci.

```
public class PDFGenerator : IPDFGenerator
{
     private IConverter _converter;
     public PDFGenerator(IConverter converter)
     {
         _converter = converter;
     }
     public byte[] CreateContractPDF(ContractInput contractInput)
     {
         var globalSettings = new GlobalSettings
         {
             ColorMode = ColorMode.Color,
             Orientation = Orientation.Portrait,
             PaperSize = PaperKind.A4,
             Margins = new MarginSettings { Top = 10 },
             DocumentTitle = "Contract"
         };

         var objectSettings = new ObjectSettings
         {
             PagesCount = true,
             HtmlContent = TemplateGenerator.GetHtml(contractInput),
             WebSettings = { DefaultEncoding = "utf-8", UserStyleSheet = Path.Com-
bine(Directory.GetCurrentDirectory(), "Services", "PDFGenerator", "Assets", "PDFSty-
les.css") },
         HeaderSettings = { FontName = "Arial", FontSize = 9, Right = "Strana [page]
z [toPage]", Line = true },
         FooterSettings = { FontName = "Arial", FontSize = 9, Line = true, Center =
"" }
         };

         var pdf = new HtmlToPdfDocument()
         {
             GlobalSettings = globalSettings,
             Objects = { objectSettings }
         };

         return _converter.Convert(pdf);}}
```

5.8 Posílání emailu se smlouvou

Pro posílání emailu využil autor systémové knihovny System.Net.Mail. Nejprve vytvořil novou třídu EmailSenderService a interface IEmailSenderService. V této třídě vytvořil autor funkci SendEmail, která slouží k odesílání emailů. V této funkci se vytvoří Smtp klient s hostem smtp.gmail.com. Emaily se posílají přes emailovou adresu insuranceproject.school@gmail.com, kterou autor vytvořil pro tento projekt. Dále se specifikuje používání SSL klientem Smtp k zašifrování spojení. Následuje ukázka funkce SendEmail autorovy ukázkové aplikace.

```
public void SendEmail(MailMessage input)
{
     try
     {
          using(MailMessage mail = input)
          {

              mail.IsBodyHtml = true;

              using(SmtpClient smtp = new SmtpClient("smtp.gmail.com", 587))
              {
                  smtp.Credentials = new System.Net.NetworkCredential("insurancepro-
ject.school@gmail.com", "Aa123456.");
                  smtp.EnableSsl = true;
                  smtp.Send(mail);
              }
           }
      }
      catch(Exception ex)
      {

      }
}
```

Dále byla vytvořena funkce, která vytvoří emailovou zprávu a zavolá funkci SendEmail. Tato funkce se jmenuje SendContractEmail a volá se při uzavření smlouvy klientem. V této funkci se nejprve nastaví adresa odesilatele v proměnné From, poté se specifikuje předmět emailu v proměnné Subject a na konec se nastaví tělo emailu. Tělo emailu se složí z informací od klienta a předpřipraveného HTML kódu. Pro získání českého překladu enum proměnných používá autor description atribut z knihovny Systém.ComponentModel. Jelikož systémové knihovny neobsahují funkci pro jednoduché vytažení description atributu z proměnné, tak se autor rozhodl využít knihovny EnumsNET. Pomocí této knihovny lze získat description atribut enum proměnné funkcí AsString. U této funkce je nutné specifikovat typ hodnoty, kterou je potřeba získat. To se specifikuje v parametru této funkce. Následuje ukázka získání description atributu pomocí funkce AsString.

```
((InsuranceType)contractInput.SimulationInput.InsuranceType).AsString(EnumFor-
mat.Description)
```

Dále je nutné specifikovat, na kterou emailovou adresu se pošle email. To se provede přidáním klientské adresy do kolekce adres To. Příloha emailu se přidá do kolekce příloh Attachments. V rámci autorovy demonstrující aplikace se do přílohy dává PDF soubor smlouvy pojištění. Následuje ukázka funkce SendContractEmail autorovy demonstrující aplikace.

```
private void SendContractEmail(ContractInput contractInput)
{
     var email = new MailMessage()
     {
          From = new MailAddress("insuranceproject.school@gmail.com"),
          Subject = "Contract",
          Body = "<h1>Děkujeme vám za vytvoření smlouvy u náší pojišťovny</h1>" +
                "<div>Klient: <b>"+ contractInput.SimulationInput.Client.FirstName +
" " + contractInput.SimulationInput.Client.FirstName + "</b></div>" +
                "<div>Type pojištění: <b>" + ((InsuranceType)contractInput.Simulati-
onInput.InsuranceType).AsString(EnumFormat.Description) + "</b></div>" +
                "<div>Produkt: <b>" + contractInput.SimulationInput.SelectedTa-
riff.Name + "</b></div>" +
                "<div>Cena: <b>" + contractInput.SimulationInput.SelectedTa-
riff.Price + " Kč</b></div>"
     };
     email.To.Add(contractInput.SimulationInput.Client.Email);
     email.Attachments.Add(new Attachment(new MemoryStream(_pdfGenerator.CreateCon-
tractPDF(contractInput)), "Smlouva.pdf"));
     _emailSender.SendEmail(email);
}
```

5.9 Využití sdíleného kódu pro Client a Server projekt

Obrovská výhoda tohoto Blazor projektu je možnost sdílení kódu mezi projekty. Tímto sdílením kódu se šetří čas potřebný k napsání kódu. Dále se eliminují chyby způsobené změnou kódu jen v jednom projektu. Autor využívá například sdílení kódu pro modely a pro funkci vypočítávající cenu balíčků pojištění. Tato funkce se používá v Server projektu k vypočítání cen balíčku po přechodu na třetí krok pojištění a v Client projektu k přepočítání cen balíčků po změně frekvence placení. Díky této funkci, není potřeba znovunačtení stránky pro přepočet cen balíčků pojištění. Tato funkce se tedy implementovala do projektu Shared, který je sdílený mezi projekty. Následuje ukázka funkce GetProducts třídy ProductGenerator, která z informací od uživatele stanoví cenu balíčků pojištění.

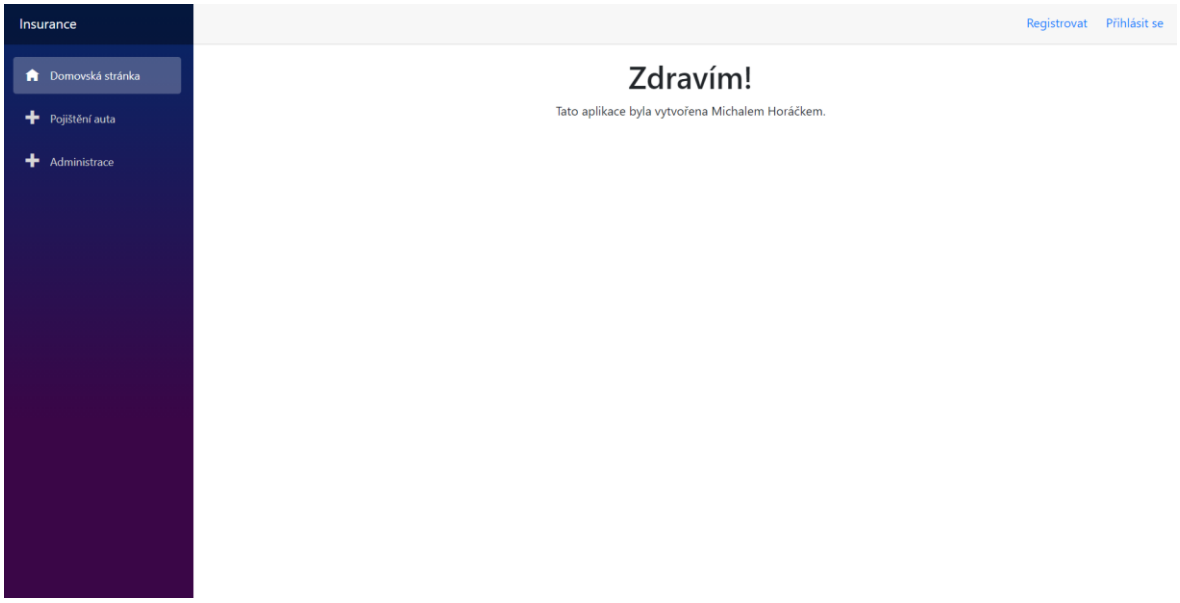
```
public static List<Tariff> GetProducts(SimulationInput input)
{
    var tariffs = new List<Tariff>();
    switch (input.InsuranceType)
    {
        case InsuranceType.LiabilityInsurance:
             tariffs = GetPovSimulationResults(input);
             break;
        case InsuranceType.CollisonDamageInsurance:
             tariffs = GetHavSimulationResults(input);
             break;
        case InsuranceType.VehicleInsurance:
             tariffs = GetPovHavSimulationResults(input);
             break;
        default:
             break;
    }
    if (input.PaymentFrequency == PaymentFrequencyType.Yearly)
    {
        tariffs.ForEach(e => e.Price = (int)(e.Price * 0.97));
    }
        // payment frequency change
        tariffs.ForEach(e => e.Price = GetTariffPriceByPaymentFrequency(input.Pay-
mentFrequency, e.Price));
        return tariffs;
}
```

6 DEMONSTRACE APLIKACE

Tato kapitola popisuje autorem vytvořenou aplikaci ve frameworku Blazor. Aplikace byla vytvořena za účelem demonstrace možností frameworku Blazor.

6.1 Domovská stránka

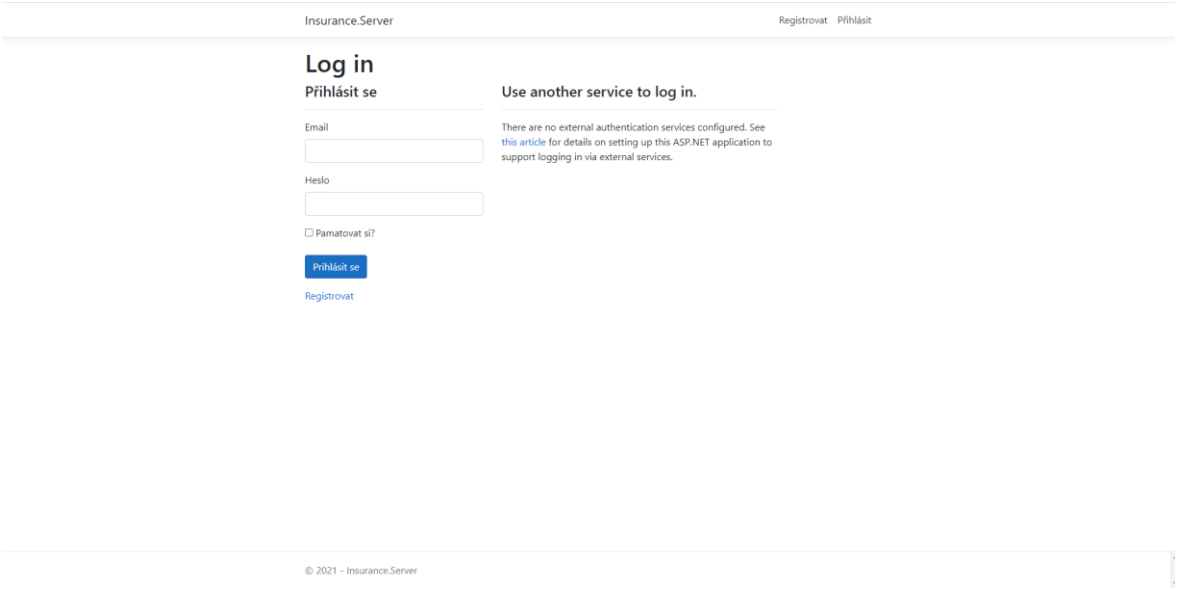
Po načtení webové stránky se uživatel dostane na domovskou stránku. V pravém horním rohu se nachází odkaz na přihlášení uživatele a odkaz na registraci uživatele. V levé části aplikace se nachází menu aplikace. Menu obsahuje odkazy na stránky Domovská stránka, Pojištění auta a Administrace. V prostřední části je specifikovaný autor aplikace a dále pokud je uživatel přihlášený, tak vidí text specifický podle role uživatele. Na obrázku číslo 13 můžeme vidět, jak domovská stránka vypadá.



Obrázek 13 Domovská stránka

6.2 Přihlášení uživatele

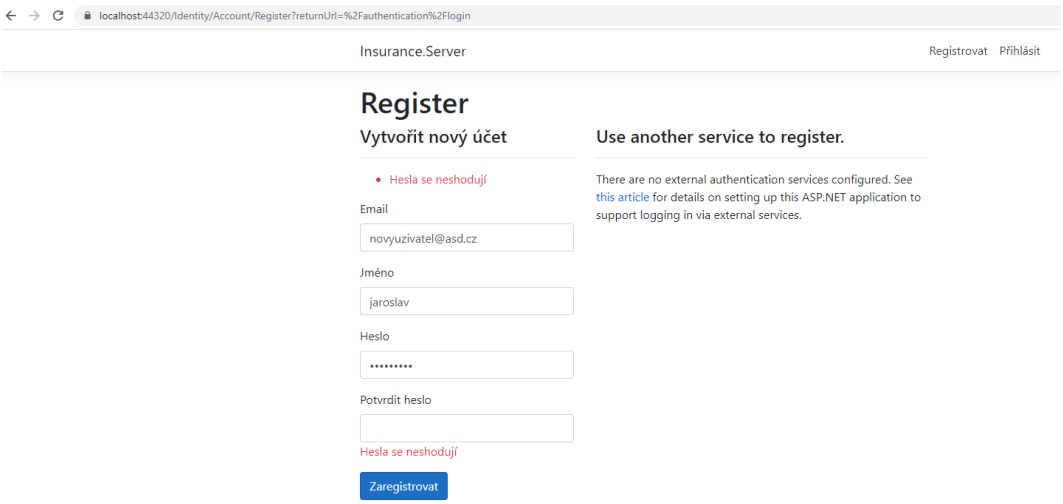
Tato stránka slouží k přihlášení uživatele do aplikace. Pro přihlášení uživatele je potřeba vyplnit políčka Email a Heslo. Dále se zde nachází checkbox Pamatovat si, kterým se nastavuje, jestli se má přihlášení ukládat do Session storage nebo do permanentní Cookie. Když se tento checkbox zatrhne, tak si prohlížeč bude pamatovat přihlášeného uživatele i po vypnutí prohlížeče. Nakonec se zde nachází odkaz na registraci nového uživatele a odkaz na domovskou stránku. Na obrázku číslo 14 může vidět stránku přihlášení uživatele.



Obrázek 14 Stránka přihlášení uživatele

6.3 Registrace uživatele

Tato stránka slouží k registraci uživatele v aplikaci. Po uživateli je požadováno vyplnění informací o novém účtu. Musí se vyplnit email, jméno uživatele a heslo. Heslo se musí potvrdit, aby nedošlo k vyplnění nechtěného hesla. Registrace uživatele se potvrzuje tlačítkem Zaregistrovat. Pokud některé z údajů neodpovídají nastaveným pravidlům registrace, tak se pod daným vstupem zobrazí chybová hláška a dále se zobrazí v souhrnu chybových hlášek nad formulářem. Mezi pravidla registrace patří například shoda hesla ve vstupu Heslo a Potvrdit heslo. Na obrázku 15 můžeme vidět zobrazení chybových hlášek na stránce registrace.



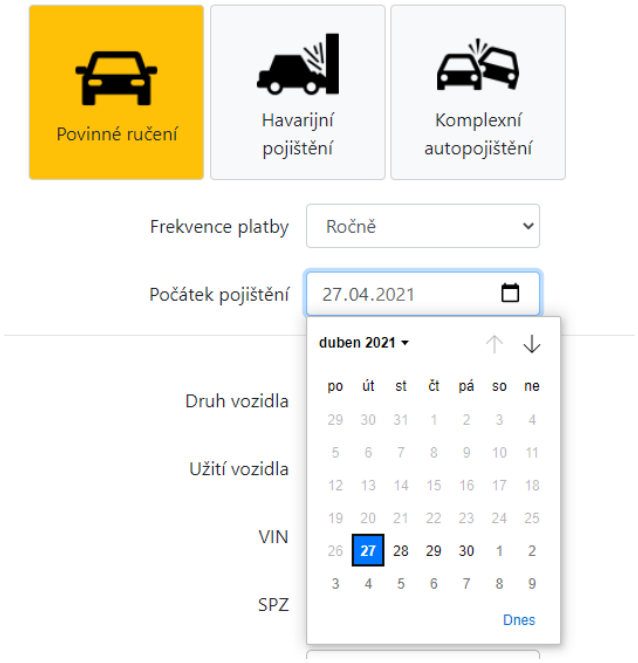
Obrázek 15 Chybové hlášky formuláře registrace

6.4 Pojištění auta

Pojištění auta se skládá ze čtyř stránek, kde uživatel vyplňuje informace, a nakonec sjedná smlouvu. Tyto stránky jsou Základní informace, Nabídka balíčků, Dodatečné informace klienta, Děkovná stránka.

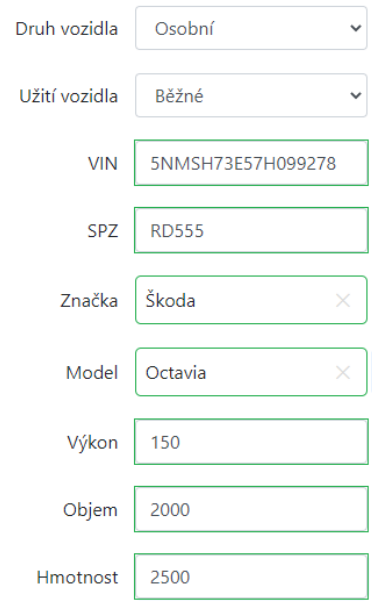
6.4.1 Základní informace

Tato stránka slouží k získání základních informací o uživateli, vozidla a pojištění. V první části stránky se získávají informace o typu pojištění, frekvence platby a počátku pojištění. Typ pojištění se získává z výběru hodnot, kde každý prvek je tvořen obrázkem a názvem typu pojištění. Tento výběr hodnot slouží k demonstraci komponent. Po kliknutí na tlačítko z výběru hodnot se žlutě označí. Počátek pojištění se zadává pomocí vyskakovacího kalendáře. Na obrázku 16 můžeme vidět první část stránky.



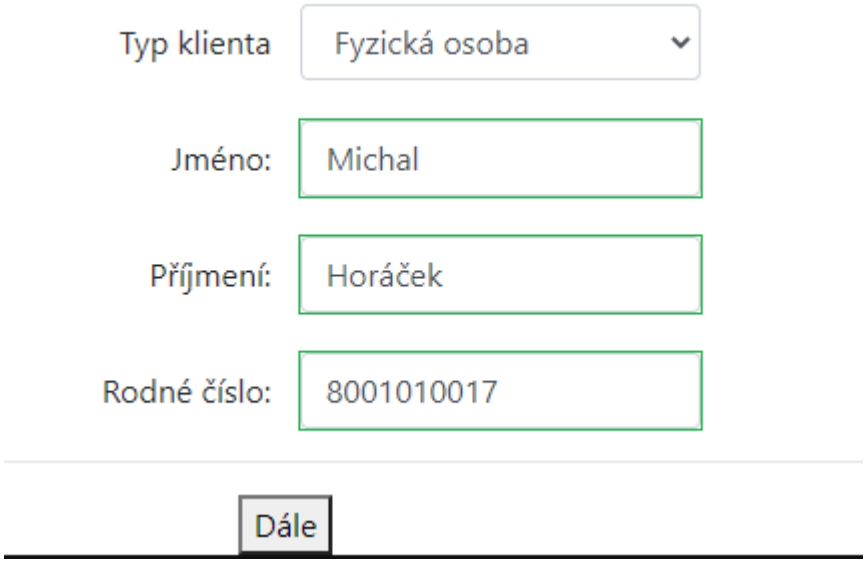
Obrázek 16 Základní informace – typ pojištění a frekvence platby

V další části se získávají informace o vozidle. Mezi tyto informace patří druh vozidla, užití vozidla, VIN, SPZ, značka, model, výkon, objem motoru, hmotnost. Druh vozidla a užití vozidla se získává pomocí výběru hodnot. Značka a model vozidla se získává pomocí textového vstupu s vyhledáváním hodnot. Model vozidla se dá vybrat až po zvolení značky vozidla. Zbytek informací se získává pomocí textových vstupů. Na obrázku 17 lze vidět získávané informace o autě.



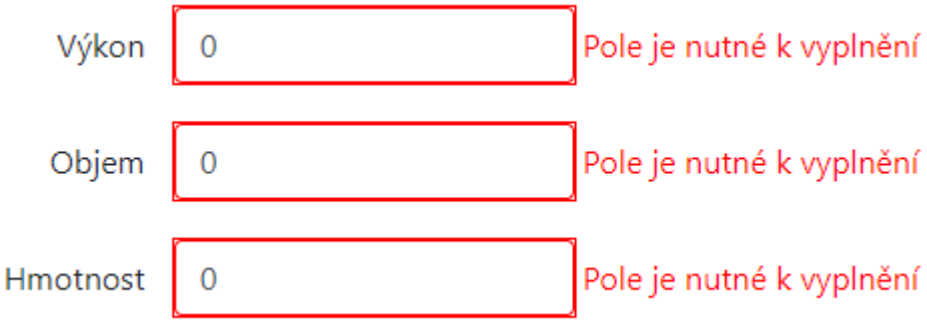
Obrázek 17 Základní informace – informace o vozidle

V poslední části se vyplňují informace o klientovi. Prvně se vyplní typ klienta. Podle typu klienta jsou požadovány různé údaje. Je na výběr z fyzické osoby, právnické osoby a OSVČ. U fyzické osoby se vyplňuje jméno, příjmení a rodné číslo. U právnické osoby se vyplňuje název společnosti a IČO. U OSVČ se vyplňuje jméno, příjmení, rodné číslo, název společnosti, IČO. Na obrázku číslo 18 můžeme vidět získávané informace klienta.



Obrázek 18 Základní informace – informace o klientovi

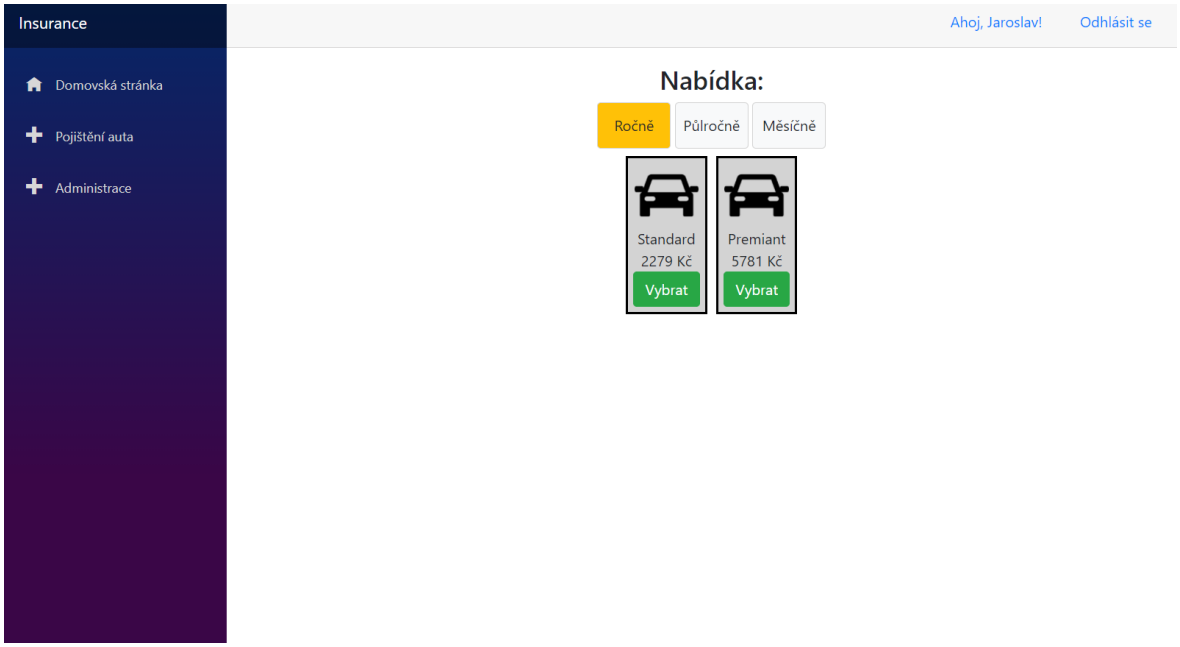
Tento formulář se validuje pomocí Fluent validací. Pokud nějaké pole formuláře neprojde přes stanovené Fluent validace, tak se u pole zobrazí červeně chybová hláška. Formulář se potvrdí, dokud se nevyplní všechny požadované informace ve správném tvaru. Pomocí tlačítkem Dále se přejde na další krok. Na obrázku číslo 19 můžeme vidět validaci polí.



Obrázek 19 Základní informace – validace polí

6.4.2 Nabídka

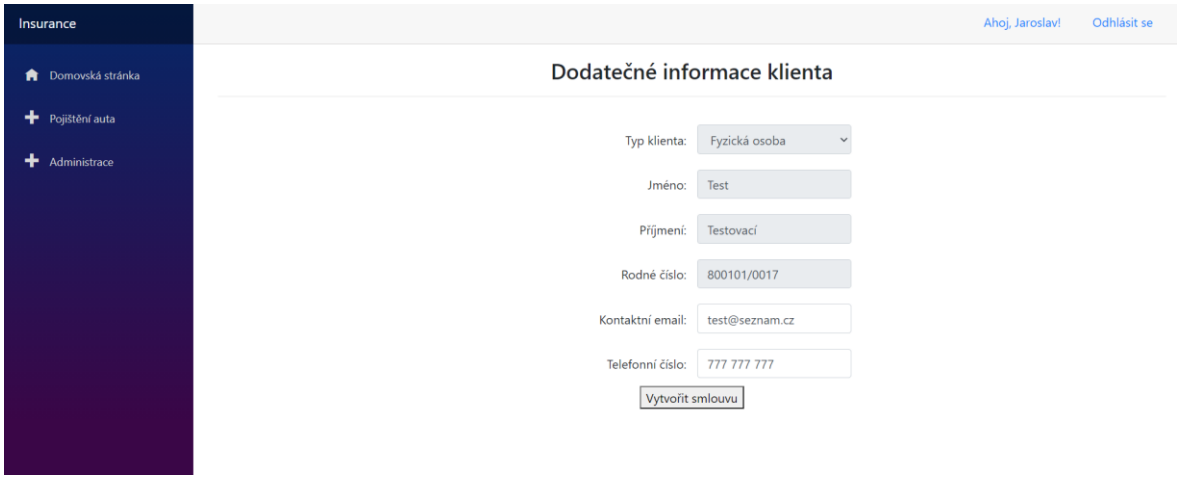
Na stránce nabídka uživatel může vidět vypočítanou cenu jednotlivých balíčků. Dále si zde může změnit frekvenci placení. Pokud uživatel si změní frekvenci placení, tak se přepočítají jednotlivé ceny balíčku dle zvolené frekvence. Přepočítání nové ceny balíčků po změně frekvence placení se provádí v prohlížeči uživatele, a proto není potřeba znovunačtení stránky. Po kliknutí na tlačítko Vybrat v balíčku se přejde na další stránku pojištění. Na obrázku číslo 20 můžeme vidět stránku s nabídkou balíčku pojištění.



Obrázek 20 Nabídka balíčku pojištění

6.4.3 Dodatečné informace klienta

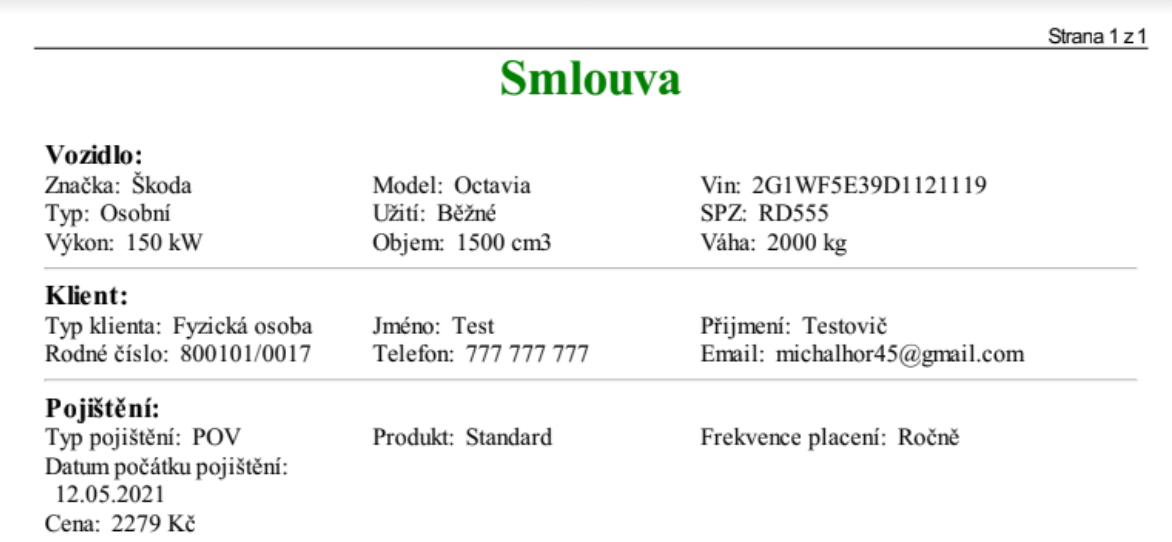
Tato stránka slouží k získání dodatečných údajů, které nejsou cenotvorné. Tyto údaje jsou potřebné pro kontaktování klienta po sjednání smlouvy. Na této stránce se propisují údaje klienta z prvního kroku, ale nejdou změnit. Jsou zde pouze pro informování uživatele. Po uživateli se požaduje vyplnění emailu a telefonního čísla. Po kliknutí na tlačítko Vytvořit smlouvu se vytvoří v aplikaci smlouva, vytvoří se PDF soubor se smlouvou a odešle se email o vytvoření smlouvy. Na obrázku číslo 21 můžeme vidět stránku s dodatečnými informacemi klienta.



Obrázek 21 Dodatečné informace klienta

6.4.4 PDF soubor se smlouvou

Vytvořený soubor se smlouvou je typu PDF. Obsahuje všechny informace o smlouvě, které uživatel vyplnil a vybrané pojištění. Smlouva je rozdělena do tří částí. První část obsahuje veškeré informace o vozidle, které uživatel vyplnil na stránce Základní informace. Druhá část obsahuje veškeré informace o klientovi, které uživatel vyplnil na stránce Základní informace a Dodatečné informace. Poslední část obsahuje informace o sjednaném pojištění. Mezi tyto informace patří vybraný typ pojištění, vybraný produkt, frekvence placení, datum počátku pojištění a výsledná cena pojištění. Na obrázku číslo 22 můžeme vidět výslednou smlouvu, která se posílá klientovi na email.



Strana 1 z 1

Smlouva

Vozidlo:

Značka: Škoda	Model: Octavia	Vin: 2G1WF5E39D1121119
Typ: Osobní	Užití: Běžné	SPZ: RD555
Výkon: 150 kW	Objem: 1500 cm3	Váha: 2000 kg

Klient:

Typ klienta: Fyzická osoba	Jméno: Test	Příjmení: Testovič
Rodné číslo: 800101/0017	Telefon: 777 777 777	Email: michalhor45@gmail.com

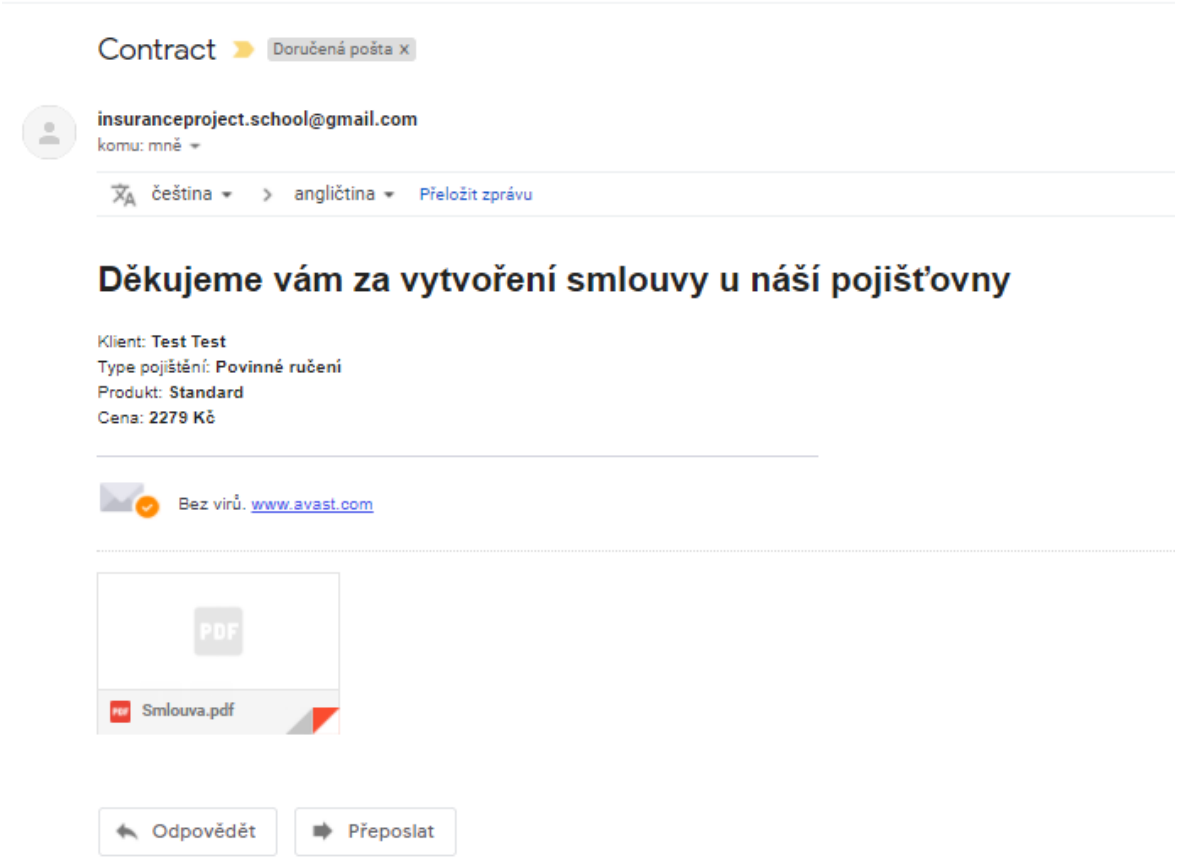
Pojištění:

Typ pojištění: POV	Produkt: Standard	Frekvence placení: Ročně
Datum počátku pojištění: 12.05.2021		
Cena: 2279 Kč		

Obrázek 22 Smlouva

6.4.5 Email poslaný klientovy

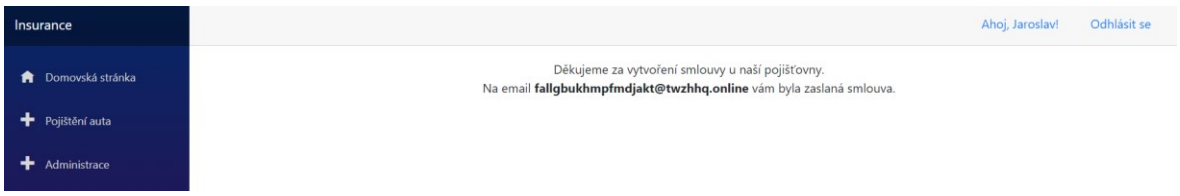
Po vytvoření smlouvy se klientovi odešle email se základními informacemi a přiloženou smlouvou. V emailu je poděkování za vytvoření smlouvy a základní informace o pojištění. Mezi tyto informace patří název klienta, typ pojištění, vybraný produkt a cena pojištění. Název klienta je rozdílný podle typu klienta. U fyzické osoby a OSVČ se zobrazuje jméno a příjmení. U právnické osoby se zobrazuje název společnosti. V příloze je obsažený soubor s názvem Smlouva.pdf, který obsahuje vytvořenou smlouvu. Na obrázku číslo 23 můžeme vidět ukázkový email, který se posílá klientovi.



Obrázek 23 Vytvořený email

6.4.6 Děkovná stránka

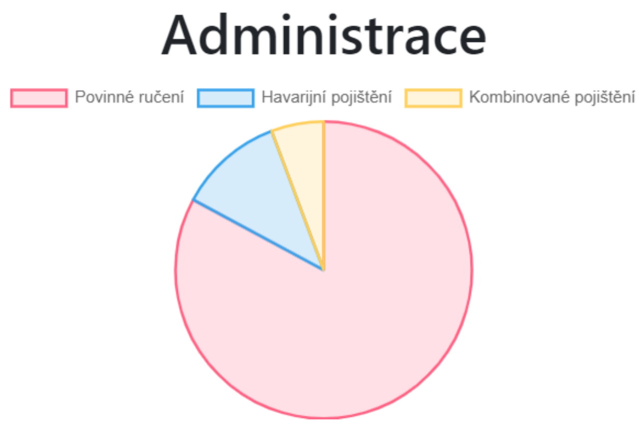
Po vytvoření smlouvy se přejde na děkovnou stránku. Na této stránce je poděkování za vytvoření smlouvy a připomínka, že na zadaný email byla zaslaná smlouva. Na obrázku číslo 24 můžeme vidět děkovnou stránku po vytvoření smlouvy.



Obrázek 24 Děkovná stránka

6.5 Administrace

Tato stránka slouží pro informování administrátorů o sjednaných smlouvách v aplikaci. Tato stránka je přístupná pouze uživatelům s rolí administrátor. Ve vrchní části stránky je graf sjednaných pojištění. Tento graf znázorňuje množství sjednaných typu pojištění. Mezi typy pojištění patří Povinné ručení, Havarijní pojištění a Kombinované pojištění. Na obrázku číslo 25 můžeme vidět graf sjednaných pojištění.



Obrázek 25 Administrace – graf sjednaných pojištění

Ve spodní části stránky je tabulka sjednaných pojištění. Tato tabulka zobrazuje všechny sjednané pojištění. Tuto tabulku lze seřadit podle libovolného sloupce a je stránkovaná po deseti záznamech. Sloupce v tabulce jsou Datum, Klient, Typ pojištění a Cena. Na obrázku číslo 26 lze vidět tabulku sjednaných pojištění.

Datum	Klient	Typ pojištění	Cena
01.04.2021 0:00:00	asd asd	Povinné ručení	5960 Kč
01.04.2021 0:00:00	asd asd	Povinné ručení	5960 Kč
01.04.2021 0:00:00	asd asd	Povinné ručení	5960 Kč
01.04.2021 0:00:00	asd asd	Povinné ručení	5960 Kč
01.04.2021 0:00:00	asd asd	Povinné ručení	5960 Kč
01.04.2021 0:00:00	asd asd	Povinné ručení	5960 Kč
01.04.2021 0:00:00	asd asd	Povinné ručení	5960 Kč
01.04.2021 0:00:00	asd asd	Povinné ručení	5960 Kč
01.04.2021 0:00:00	asd asd	Povinné ručení	5960 Kč
01.04.2021 0:00:00	asd asd	Povinné ručení	5960 Kč

First Prev 1 2 3 4 Next Last — 1 - 10 of 35 items

Obrázek 26 Administrace – tabulka sjednaných pojištění

ZÁVĚR

Výstupem praktické části diplomové práce je aplikace z oblasti pojišťovnictví demonstrující možnosti frameworku Blazor a popis klíčových částí této aplikace s ukázkami kódů. Čtenář by po přečtení práce měl být seznámený s frameworkem Blazor a měl by být schopen vytvořit aplikaci v tomto frameworku.

V teoretické části diplomové práce byl popsaný současný stav vývoje webových aplikací včetně popisů nejpoužívanějších webových a databázových technologií. Dále byl popsán framework Blazor a jeho možnosti hostování. Větší pozornost byla věnována šabloně Blazor WebAssembly. Zbytek teoretické části je věnován možnostmi frameworku Blazor pro tvorbu aplikací z oblasti pojišťovnictví a bankovnictví.

V praktické části diplomové práce byl vytvořen návrh ukázkové aplikace z oblasti pojišťovnictví. V návrhu jsou popsány scénáře užití, funkční a nefunkční požadavky. Dále bylo popsáno založení projektu, vybrané nastavení projektu a šablony. V další kapitole byly popsány klíčové prvky aplikace s jednotlivými ukázkami kódu. Mezi klíčové prvky aplikace autor zahrnul autorizaci uživatele, uchování dat skrze stránky, validaci vstupu formuláře, práce s databází, vytvoření PDF souboru smlouvy, posílání emailu se smlouvou a využití sdíleného kódu pro Client a Server projekt. Poslední kapitola se věnuje demonstraci vytvořené aplikace. Jsou zde popsány jednotlivé stránky aplikace.

Autor při vytváření ukázkové aplikace narazil na několik problémů. Například napovídání kódu IntelliSense nefungovalo ve všech případech. Při vytváření komponent se autorovi párkrát stalo to, že IntelliSense neviděl na vytvořenou komponentu a nedoplňoval tak její název. Tohle autor vyřešil restartováním Visual Studia. Dále autor narazil na limitace frameworku Blazor pro vytváření dynamických vstupů ve formuláři. Například pomocí systémových komponent nelze vytvořit vstup ve formuláři s našeptáváním hodnot. Tento nedostatek ale lze vyřešit napsáním vlastní komponenty nebo použitím knihoven třetích stran. Tento framework je poměrně nový a každou novou aktualizací jeho autoři přidávají nové funkcionality.

Pro aplikace z oblasti bankovnictví autor nedoporučuje Blazor WebAssembly z důvodu toho, že je tento framework relativně nový. Do budoucna má tento framework velký potenciál. Při vytváření ukázkové aplikace autor došel k závěru, že tento framework může konkurovat dnešním frontendovým frameworkům.

SEZNAM POUŽITÉ LITERATURY

[1] 2020 Denveloper survey. Stackoverflow [online]. 2020 [cit. 2021-02-06]. Dostupné z: https://insights.stackoverflow.com/survey/2020#technology-databases-all-respondents4

[2] *JQuery Introduction* [online]. Refsnes Data [cit. 2021-02-06]. Dostupné z: https://www.w3schools.com/jquery/jquery_intro.asp

[3] BYNENS, Mathias. JQuery file size. *Mathiasbynens* [online]. [cit. 2021-02-06]. Dostupné z: https://mathiasbynens.be/demo/jquery-size

[4] *JQuery Usage Statistics* [online]. BuiltWith® Pty [cit. 2021-02-06]. Dostupné z: https://trends.builtwith.com/javascript/jQuery

[5] MORRIS, Scott. TECH 101: WHAT IS REACT JS? *Skillcrush* [online]. c2012-2021 [cit. 2021-02-10]. Dostupné z: https://skillcrush.com/blog/what-is-react-js/#used

[6] *Introduction to Angular concepts* [online]. c2010-2020 [cit. 2021-02-12]. Dostupné z: https://angular.io/guide/architecture

[7] *The Good and the Bad of Angular Development* [online]. c2021, 25 Mar 2020 [cit. 2021-02-14]. Dostupné z: https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-angular-development/

[8] TECHSON, Mark. *Version 11 of Angular Now Available* [online]. 2020, 12. prosince 2020 [cit. 2021-02-14]. Dostupné z: https://blog.angular.io/version-11-of-angular-now-available-74721b7952f7

[9] *ASP.NET - Introduction* [online]. c2021 [cit. 2021-02-14]. Dostupné z: https://www.tutorialspoint.com/asp.net/asp.net_introduction.htm

[10] GARCIA, Daniel Jimenez. The History of ASP.NET – Part I. *Dotnetcurry* [online]. c2007-2021, 26. 4. 2019 [cit. 2021-02-14]. Dostupné z: https://www.dotnetcurry.com/aspnet/1492/aspnet-history-part-1

[11] What is Express.js? *Besanttechnologies* [online]. Chennai, c2021 [cit. 2021-02-19]. Dostupné z: https://www.besanttechnologies.com/what-is-expressjs

[12] YEGULALP, Serdar. What is the MEAN stack? JavaScript web applications. *Infoworld* [online]. London, c2021, 30 března 2020 [cit. 2021-02-19]. Dostupné z: https://www.infoworld.com/article/3319786/what-is-the-mean-stack-javascript-web-applications.html

[13] RIEUF, Emmanuelle. History of MySQL. *Data Science Central* [online]. c2021, 16. Prosince 2016 [cit. 2021-02-21]. Dostupné z: https://www.datasciencecentral.com/profiles/blogs/history-of-mysql

[14] What is MySQL? Everything You Need to Know. *Talend* [online]. © 2021 [cit. 2021-02-21]. Dostupné z: https://www.talend.com/resources/what-is-mysql/

[15] What is PostgreSQL? *PostgreSQL: The World's Most Advanced Open Source Relational Database* [online]. c1996-2021 [cit. 2021-02-22]. Dostupné z: https://www.postgresql.org/about/

[16] WARREN, Matt. DotNetAnywhere: An Alternative .NET Runtime. *Https://mattwarren.org/* [online]. 19. Říjen 2017 [cit. 2021-02-27]. Dostupné z: https://mattwarren.org/2017/10/19/DotNetAnywhere-an-Alternative-.NET-Runtime/

[17] SAINTY, Chriss. What is Blazor and why is it so exciting?: BLAZOR. *Chrissainty* [online]. c2019, 24 Březen 2018 [cit. 2021-02-27]. Dostupné z: https://chrissainty.com/what-is-blazor-and-why-is-it-so-exciting/

[18] MORRIS, Peter. What is Blazor? *Blazor University* [online]. Morris, c2019-2021 [cit. 2021-02-27]. Dostupné z: https://blazor-university.com/overview/what-is-blazor/

[19] *What is jQuery?* [online]. c2021 [cit. 2021-04-12]. Dostupné z: https://jquery.com/

[20] React: A JavaScript library for building user interfaces. *Reactjs* [online]. c2021 [cit. 2021-04-12]. Dostupné z: https://reactjs.org/

[21] *Introducing JSX* [online]. c2021 [cit. 2021-04-12]. Dostupné z: https://reactjs.org/docs/introducing-jsx.html

[22] *Virtual DOM and Internals* [online]. c2021 [cit. 2021-04-12]. Dostupné z: https://reactjs.org/docs/faq-internals.html

[23] FREED, Tony. Https://medium.com/@tony_freed/what-is-virtual-dom-c0ec6d6a925c. *Medium* [online]. 11. června 2016 [cit. 2021-04-12]. Dostupné z: https://medium.com/@tony_freed/what-is-virtual-dom-c0ec6d6a925c

[24] *The modern web developer's platform* [online]. c2010-2021 [cit. 2021-04-12]. Dostupné z: https://angular.io/

[25] *AngularJS* [online]. c2010-2018 [cit. 2021-04-12]. Dostupné z: https://angularjs.org/

[26] ASP.NET: Free. Cross-platform. Open source. A framework for building web apps and services with .NET and C#. *Microsoft* [online]. c2021 [cit. 2021-04-12]. Dostupné z: https://dotnet.microsoft.com/apps/aspnet

[27] ASP Overview. *Microsoft* [online]. c2021, 16. 6. 2017 [cit. 2021-04-12]. Dostupné z: https://docs.microsoft.com/en-us/previous-versions/iis/6.0-sdk/ms524929(v=vs.90)

[28] GARCIA, Daniel Jimenez. The History of ASP.NET – Part II (Covers ASP.NET MVC). *Dotnetcurry* [online]. c2007-2021, 27. 4. 2019 [cit. 2021-04-12]. Dostupné z: https://www.dotnetcurry.com/aspnet/1493/aspnet-history-part-2-mvc

[29] FRITZ, Jeffrey. *Announcing ASP.NET Core 1.0* [online]. c2021, 27. června 2016 [cit. 2021-04-12]. Dostupné z: https://devblogs.microsoft.com/aspnet/announcing-asp-net-core-1-0/

[30] *Express: Fast, unopinionated, minimalist web framework for Node.js* [online]. c2017 [cit. 2021-04-12]. Dostupné z: https://expressjs.com/

[31] *Mysql* [online]. c2021 [cit. 2021-04-12]. Dostupné z: https://www.mysql.com/

[32] PostgreSQL 13.2, 12.6, 11.11, 10.16, 9.6.21, and 9.5.25 Released! *Postgresql* [online]. c1996-2021, 11. 2. 2021 [cit. 2021-04-12]. Dostupné z: https://www.postgresql.org/about/news/postgresql-132-126-1111-1016-9621-and-9525-released-2165/

[33] Blazor: Build client web apps with C#. *Microsoft* [online]. c2021 [cit. 2021-04-12]. Dostupné z: https://dotnet.microsoft.com/apps/aspnet/web-apps/blazor

[34] *WebAssembly* [online]. [cit. 2021-04-12]. Dostupné z: https://webassembly.org/

[35] SHIRHATTI, Sourabh. *ASP.NET Core updates in .NET 6 Preview 1* [online]. c2021, 17. února 2021 [cit. 2021-04-12]. Dostupné z: https://devblogs.microsoft.com/aspnet/asp-net-core-updates-in-net-6-preview-1/

[36] Introduction to ASP.NET Core Blazor. *Microsoft* [online]. c2021, 25. 9. 2020 [cit. 2021-04-12]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-5.0

[37] *Secure ASP.NET Core Blazor WebAssembly* [online]. c2021, 27. 10. 2020 [cit. 2021-04-12]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/blazor/security/webassembly/?view=aspnetcore-5.0

[38] *OAuth 2.0* [online]. [cit. 2021-04-12]. Dostupné z: https://oauth.net/2/

[39] *Welcome to OpenID Connect* [online]. c2021 [cit. 2021-04-12]. Dostupné z: https://openid.net/connect/

[40] *ASP.NET Core Blazor Server with Entity Framework Core (EFCore)* [online]. c2021, 14. 8. 2020 [cit. 2021-04-13]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/blazor/blazor-server-ef-core?view=aspnetcore-5.0

[41] *Using gRPC-Web with Blazor WebAssembly* [online]. c2021, 15. ledna 2020 [cit. 2021-04-13]. Dostupné z: https://blog.stevensanderson.com/2020/01/15/2020-01-15-grpc-web-in-blazor-webassembly/

[42] *GRPC: A high performance, open source universal RPC framework* [online]. c2021 [cit. 2021-04-13]. Dostupné z: https://grpc.io/

[43] *Entity Framework documentation* [online]. c2021 [cit. 2021-04-13]. Dostupné z: https://docs.microsoft.com/en-us/ef/

[44] *Enforce a Content Security Policy for ASP.NET Core Blazor* [online]. c2021, 19. 5. 2020 [cit. 2021-04-13]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/blazor/security/content-security-policy?view=aspnetcore-5.0

[45] ASP.NET Core Blazor hosting models. *Microsoft* [online]. c2021, 12. 07. 2020 [cit. 2021-5-7]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/blazor/hosting-models?view=aspnetcore-5.0

[46] Authentication and authorization in ASP.NET Core SignalR. *Microsoft* [online]. c2021, 12. 05. 2019 [cit. 2021-5-13]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/signalr/authn-and-authz?view=aspnetcore-5.0

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

PDF	Portable Document Format
DOM	Document Object Model
JSX	JavaScript Xml
UI	User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
MVC	Model View Controller
ASP	Microsoft Active Server Pages
MS-PL	Microsoft Public License
MEAN	MongoDB Express Angular Node.js
JSON	JavaScript Object Notation
SQL	Structured Query Language
ACI	Application Centric Infrastructure
CIL	Common Intermediate Language
XML	Extensible markup language
API	Application Programming Interface
CDN	Content Delivery Network
SPA	Single Page Aplication
AoT	Ahead of Time
OP	OpenID Providers
OIDC	OpenID Connect
JWT	JSON Web Token
CSFR	Cross-Site Request Forgery
XSS	Cross-Site Scripting

CSP	Content Security Policy
EF Core	Entity Framework Core
URL	Uniform Resource Locator
RPC	Remote Procedure Call
SMTP	Simple Mail Transfer Protocol
SSL	Secure Sockets Layer
SPZ	Státní poznávací značka
IČO	Identifikační číslo osoby
OSVČ	Osoba samostatně výdělečně činná
JS	JavaScript

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: CD

Přiložené CD obsahuje:

- Diplomovou práci ve formátu docx: fulltext.docx
- Diplomovou práci ve formátu pdf: fulltext.pdf
- Zdrojové kódy: příloha.zip