Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Diplomová práce

Porovnání ASP.NET Core MVC a frameworku Blazor

Bc. Lukáš Veteška

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Lukáš Veteška

Systémové inženýrství a informatika
Informatika

Název práce

Porovnání ASP.NET Core MVC a frameworku Blazor

Název anglicky

ASP.NET Core MVC vs Blazor framework

Cíle práce

Diplomová práce je tematicky zaměřena na problematiku vývoje webových aplikací na platformě .NET Core. Hlavním cílem práce je porovnání frameworku Blazor a ASP.NET Core MVC pro tvorbu webových aplikací se zaměřením na online zpravodajství. Dílčí cíle práce jsou:
- Vytvoření webových aplikací pro porovnání
- Stanovení vhodných kritérií pro porovnání
- Realizace experimentu pro porovnání obou technologií

Metodika

Metodika řešení diplomové práce je založena na analytickém a syntetickém přístupu. Bude provedena analýza odborných informačních zdrojů, jejíž souhrn bude uveden v teoretické části práce. Na základě syntézy zjištěných poznatků bude zpracován popis současného stavu, nástrojů a technologií pro vývoj webových aplikací na platformě Blazor a ASP.NET Core.

Dále budou vytvořeny dvě reálné webové aplikace založené na frameworku Blazor a ASP.NET Core MVC. Budou zvoleny vhodné metriky měření a na základě vyhodnocení provedeného experimentu porovnány obě technologií. Na základě výsledků teoretické a praktické části budou shrnuty poznatky a formulovány závěry.

Doporučený rozsah práce

60 – 80 stran

Klíčová slova

ASP.NET Core, C#, Blazor, framework, MVC, SPA, web

Doporučené zdroje informací

Blazor Revealed: Building Web Applications in .NET
Exploring Blazor: Creating Hosted, Server-side, and Client-side Applications with C#
Pro ASP.NET Core 3: Develop Cloud-Ready Web Applications Using MVC 3, Blazor, and Razor Pages

---

Předběžný termín obhajoby

2020/21 LS – PEF

Vedoucí práce

Ing. Jan Masner, Ph.D.

Garantující pracoviště

Katedra informačních technologií

Elektronicky schváleno dne 29. 7. 2020

Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 21. 10. 2020

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 21. 03. 2021

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Porovnání ASP.NET Core MVC a frameworku Blazor" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 22. března 2021 _____

Poděkování

Rád bych poděkoval Ing. Janu Masnerovi, Ph.D. za odborné vedení diplomové práce, pevné nervy a cenné rady při psaní práce. Dále tátovi za plnou podporu při studiu, jeho přítelkyni za zastoupení menzy v době koronakrize, strýci Milanovi a tetě Sonje za odborný jazyk, ale také zbytku rodiny, která mě podporovala a motivovala. Poděkování patří blízkým přátelům zejména J. Šipkovi, Pavlovi a Kristýně za velkou psychickou podporu, ale i ostatním kamarádům, kteří mi byli oporou při studiu. Velké díky patří předsedkyni iZUN.eu Nikole Řehákové za jazykovou korekturu a studentské organizaci iZUN.eu, ve které jsem byl součástí po celou dobu studia a umožnila mi realizovat jak tvorbu nového webu, tak experimentu v diplomové práci. Díky patří tvůrcům Carter Bayesovi a Craigu Thomasovi za tvorbu HIMYM série, která mi během psaní práce v době koronakrize dodávala nadšení a zápal.

Porovnání ASP.NET Core MVC a frameworku Blazor

Abstrakt

Diplomová práce se zaměřuje na porovnání dvou různých technologií – ASP.NET Core MVC a frameworku Blazor. Teoretická část se zabývá historií technologií pro vytváření intuitivních webových aplikací a jejich následným srovnáním. Následuje srovnání současných programovacích jazyků pro realizaci webových aplikací či prezentací. V dalších částech práce dochází k popisu rozdílu MVC a frameworku Blazor. V čem konkrétně nová technologie Blazor přináší užitek a v čem spočívá jeho nevýhoda. Zároveň jsou popsány metody, jakými způsoby lze srovnávat dvě odlišné technologie.

Praktická část se nejdříve zabývá předpoklady pro implementaci obou technologií. Po definování předpokladů dochází k implementaci. V další fázi jsou zvoleny vhodné srovnávací metody a dochází k porovnání technologií. Praktickou část uzavírá zhodnocení náročnosti implementace, požadavků na provoz, případných bezpečnostních rizik a dosažených výsledků.

Klíčová slova: ASP.NET Core, C#, Blazor, framework, MVC, SPA, web

ASP.NET Core MVC vs Blazor framework

Abstract

The diploma thesis focuses on comparison of two different technologies – ASP.NET Core MVC & Blazor framework. Theoretical part is focused on introducing history of creating intuitive web apps followed by comparison. Next step is comparison of current programming languages for realization of web apps / pages. Next parts describe difference between MVC & Blazor Framework, what exactly brings new technology Blazor versus MVC and what is the technology disadvantage. There are also described methods – how to compare different technologies.

Practical part deals first with assumptions for implementation of MVC & Blazor. After assumptions are defined, implantation follows. The next stage occurs selection of appropriate methods for technology comparison. Practical part is closed by evaluation of implementation difficulty, system requirements, security vulnerabilities and achieved results.

Keywords: ASP.NET Core, C#, Blazor, framework, MVC, SPA, web

Obsah

1 Úvod **12**

2 Cíl práce a metodika **13**

2.1 Cíl práce 13

2.2 Metodika 13

3 Teoretická východiska **14**

3.1 Webové technologie 14

3.1.1 Client-side 14

3.1.2 Server-side 19

3.1.3 Protokoly a komunikace 20

3.1.4 Datové formáty 24

3.2 Nástroje pro testování rychlosti a výkonu webových stránek 25

3.2.1 GTmetrix 25

3.2.2 WebPage Test 26

3.2.3 Google PageSpeed Insight 26

3.2.4 Google Chrome DevTools 26

3.3 ASP.NET Core MVC 26

3.3.1 MVC 28

3.3.2 Routing 29

3.3.3 Model binding 30

3.3.4 Validace modelu 30

3.3.5 Dependency Injection 31

3.3.6 Filtry 31

3.3.7 Oblasti (Areas) 32

3.3.8 Web API 32

3.3.9 Testovatelnost 32

3.3.10 Razor view engine 32

3.3.11 Tag helpers 33

3.4 Blazor 33

3.4.1 Komponenty 34

3.4.2 Client-side (WebAssembly) 36

3.4.3 Server-side (SignalR) 37

3.4.4 Routing 38

3.4.5 JavaScript Interop 39

3.4.6 Event handling 39

3.4.7 Data binding 40

3.4.8 Layouts 40

3.5 Obdobná řešení 40

3.5.1 Srovnání Symfony, ASP.NET MVC a Node.js ... 40
3.5.2 Srovnání SPA a MPA ... 42
3.5.3 Srovnání Blazor a tradiční webové aplikace ... 44
3.6 Shrnutí ... 46

4 Praktická část ... 47
4.1 Prostředí pro implementaci ... 47
4.2 Implementace ... 48
4.2.1 Společný základ ... 48
4.2.2 ASP.NET Core MVC ... 53
4.2.3 Blazor ... 62
4.3 Porovnání ... 70
4.3.1 Celková velikost a rychlost stránky při prvním načtení ... 70
4.3.2 Celková velikost a rychlost stránky při druhém načtení ... 71
4.3.3 Celková velikost a rychlost stránky při navigaci webem ... 72
4.3.4 Hodnocení PageSpeed Insights ... 73
4.3.5 Počet řádků zdrojového kódu ... 77

5 Výsledky a diskuse ... 79
5.1 Podpora starších prohlížečů ... 79
5.2 SEO ... 80
5.3 Rychlost a velikost ... 80
5.4 Zdrojový kód ... 82
5.5 Srovnání kritérií ... 83
5.6 Možnosti rozšíření ... 84

6 Závěr ... 85

7 Seznam použitých zdrojů ... 87

Seznam obrázků

Obrázek 1: Multiplexing protokolu HTTP (zdroj: [20]) ... 20
Obrázek 2: Klasický (synchronní) vs. AJAX (asynchronní) (zdroj: [25]) ... 23
Obrázek 3: AJAX vs WebSocket (zdroj: [27]) ... 24
Obrázek 4: Porovnání JSON a XML formátu (zdroj: Vlastní) ... 25
Obrázek 5: Životní cyklus požadavku ASP.NET MVC (zdroj: [34]) ... 27
Obrázek 6: Životní cyklus požadavku ASP.NET Core (zdroj: [35]) ... 28
Obrázek 7: Návrhový vzor MVC (zdroj: [37]) ... 29
Obrázek 8: Diagram pipeline filtrů (zdroj: [38]) ... 32
Obrázek 9: Životní cyklus komponenty (zdroj: [41]) ... 36
Obrázek 10: Blazor WebAssembly (zdroj: [40]) ... 36
Obrázek 11: Blazor SignalR (zdroj: [40]) ... 37
Obrázek 12: Tradiční webové aplikace a Single page aplikace (zdroj: [50]) ... 44
Obrázek 13: Základní infrastruktura (zdroj: Vlastní) ... 48
Obrázek 14: Hierarchický diagram iZUN.eu (zdroj: Vlastní) ... 49

Obrázek 15: Databázový diagram článků, hlasování a galerie (zdroj: Vlastní) 50
Obrázek 16: Databázový diagram stránek, uživatelů a kalendáře (zdroj: Vlastní) 51
Obrázek 17: Databázový diagram newsletteru, nastavení a reklamy (zdroj: Vlastní).... 51
Obrázek 18: Infrastruktura MVC aplikace (zdroj: Vlastní)..... 54
Obrázek 19: Infrastruktura Blazor aplikace (zdroj: Vlastní) 63
Obrázek 20: První načtení MVC (zdroj: Vlastní)..... 70
Obrázek 21: První načtení Blazor (zdroj: Vlastní) 71
Obrázek 22: Druhé načtení MVC (zdroj: Vlastní)..... 71
Obrázek 23: Druhé načtení Blazor (zdroj: Vlastní)..... 72
Obrázek 24: Navigace webem MVC (zdroj: Vlastní) 72
Obrázek 25: Navigace webem Blazor (zdroj: Vlastní)..... 73
Obrázek 26: PageSpeed Insights MVC mobil (zdroj: Vlastní)..... 74
Obrázek 27: PageSpeed Insights MVC desktop (zdroj: Vlastní) 75
Obrázek 28: PageSpeed Insights Blazor mobil (zdroj: Vlastní)..... 76
Obrázek 29: PageSpeed Insights Blazor desktop (zdroj: Vlastní)..... 77

Seznam tabulek

Tabulka 1: Počet řádků zdrojového kódu v projektech (zdroj: Vlastní)..... 78
Tabulka 2: Podpora prohlížečů (zdroj: Vlastní) 79
Tabulka 3: Podpora pro roboty (zdroj: Vlastní)..... 80
Tabulka 4: Chrome DevTools výsledky (zdroj: Vlastní)..... 81
Tabulka 5: Google PageSpeed Insights výsledky (zdroj: Vlastní) 82
Tabulka 6: Přehled velikosti zdrojového kódu (zdroj: Vlastní)..... 83
Tabulka 7: Přehled srovnání kritérií (zdroj: Vlastní)..... 83

Seznam zdrojových kódů

Zdrojový kód 1: Routing (zdroj: [37])..... 29
Zdrojový kód 2: Model binding (zdroj: [37]) 30
Zdrojový kód 3: Příklad značkovacího jazyku Razor (zdroj: [37])..... 33
Zdrojový kód 4: Příklad tag helperu (zdroj: [37]) 33
Zdrojový kód 5: Dialogová komponenta (zdroj: [40]) 34
Zdrojový kód 6: Použití dialogové komponenty (zdroj: [40])..... 35
Zdrojový kód 7: Blazor routing (zdroj: [43])..... 39
Zdrojový kód 8: Příklad volání JavaScript funkce (zdroj: [44])..... 39
Zdrojový kód 9: Příklad event-handlingu (zdroj: [45]) 40
Zdrojový kód 10: Příklad data bindingu (zdroj: [46]) 40
Zdrojový kód 11: Databázový kontext (zdroj: Vlastní)..... 52
Zdrojový kód 12: DatabaseContext Fluent API (zdroj: Vlastní)..... 52
Zdrojový kód 13: Dynamický titulek (zdroj: Vlastní)..... 55
Zdrojový kód 14: app.js - počasí (zdroj: Vlastní)..... 56
Zdrojový kód 15: Výpis počasí (zdroj: Vlastní) 56
Zdrojový kód 16: View Component - Partners (zdroj: Vlastní) 57
Zdrojový kód 17: View - Partners (zdroj: Vlastní)..... 57
Zdrojový kód 18: MainLayout – postranní panel (zdroj: Vlastní)..... 64
Zdrojový kód 19: MainLayout - Logika počasí (zdroj: Vlastní) 65
Zdrojový kód 20: MainLayout - výpis počasí (zdroj: Vlastní)..... 65
Zdrojový kód 21: Úvodní stránka - Blazor (zdroj: Vlastní) 66
Zdrojový kód 22: Inicializace Blazor aplikace (zdroj: Vlastní) 68
Zdrojový kód 23: JavaScript - WebAssembly podpora (zdroj: Vlastní) 68
Zdrojový kód 24: SEO - Request.Path.Value (zdroj: Vlastní) 69

Seznam použitých zkratek

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
ASP	Active Server Pages
CSS	Cascade StyleSheet
ČZU	Česká Zemědělská Univerzita
DOM	Document Object Model
HTML	HyperText Markup Language
HTTP	HyperText Tarnsfer Protocol
JSON	JavaScript Object Notation
MPA	Multi Page Application
MVC	Model View Controller
REST	REpresentational State Transfer
SEO	Search Engine Optimization
SERP	Search Engine Results Pages
SPA	Single Page Application
TCP/IP	Transmission Control Protocol/Internet Protocol
UDP	User Datagram Protocol
UI	User Interface
URL	Uniform Resource Locator
UX	User eXperience
XML	eXtensible Markup Language
W3C	World Wide Web Consortium

1 Úvod

S příchodem druhého tisíciletí došlo k radikálnímu nástupu využívání webového prostoru. Řada uživatelů a běžná veřejnost opustila tradiční tištěná média, rozhlasové a televizní vysílání a kamenné obchody. Každým rokem přichází technologická inovace v oblasti informačních a komunikačních technologií. Počítače se staly nezbytnou součástí domácností. Nové generace počítačů poskytují obrovské zvýšení výkonu, zároveň se stává dostupnější vysokorychlostní internetové připojení.

Pro tvorbu webových stránek existuje mnoho technologií. Microsoft začal s vývojem pro webový prostor již v roce 2002 se svým frameworkem ASP.NET založeným na platformě .NET Framework, která je určena pro operační systémy Microsoft Windows. Primárně je využíván programovací jazyk C#. Framework je využívaný u velkých firem a státních institucí, protože umožňuje rychlý a bezpečný vývoj webových aplikací. Od roku 2016 je nejvíce využívána modernější verze s označením ASP.NET Core. Vzhledem k předchozí verzi je založená na platformě .NET Core a podporuje další operační systémy jako Linux a MacOS. ASP.NET Core je svým způsobem restart technologie, který vstupuje do generace moderních webových aplikací. Framework představuje inovaci ve sloučení dříve oddělených frameworků: ASP.NET, ASP.NET MVC, ASP.NET Web API a ASP.NET Web Pages.

V posledních letech začala velká část prohlížečů implementovat nový standard nazývající se WebAssembly. Jedná se o technologii umožňující spuštění binárních instrukcí ve virtuálním stroji webového prohlížeče (tzv. sandbox). Využitím WebAssembly je dosaženo rychlejšího provedení operací pomocí kompilace z vysoko-úrovňového programovacího jazyka do binární formy.

Blazor je open-source framework vyvíjený společností Microsoft. Podobně jako ASP.NET Core je založený na platformě .NET Core. Jde o poměrně novou technologii umožňující spuštění ve dvou režimech: client-side a server-side. V režimu client-side využívá webový standard WebAssembly, na rozdíl od režimu server-side pracuje s tenkým JavaScript klientem a přenáší data prostřednictvím SignalR. Hlavní předností je intuitivnost ovládacích prvků, kdy není po akci provedeno znovunačtení stránky a dochází k velké úspoře dat, především na mobilních zařízeních. Uživatelé dále ocení nedoprovázející efekt obnovení stránky – tzv. probliknutí.

Diplomová práce se zabývá porovnáním dvou aktuálních technologií na platformě .NET Core – ASP.NET Core MVC a frameworku Blazor. Součástí práce je vytvoření dvou reálných webových aplikací.

2 Cíl práce a metodika

2.1 Cíl práce

Diplomová práce je tematicky zaměřena na problematiku vývoje webových aplikací na platformě .NET Core.

Hlavním cílem práce je porovnání frameworku Blazor a ASP.NET Core MVC pro tvorbu webových aplikací se zaměřením na online zpravodajství. Dílčí cíle práce jsou:

- Vytvoření webových aplikací pro porovnání.
- Stanovení vhodných kritérií pro porovnání.
- Realizace experimentu pro porovnání obou technologií.

2.2 Metodika

Metodika řešení diplomové práce je založena na analytickém a syntetickém přístupu. Bude provedena analýza odborných informačních zdrojů, jejíž souhrn bude uveden v teoretické části práce. Na základě syntézy zjištěných poznatků bude zpracován popis současného stavu, nástrojů a technologií pro vývoj webových aplikací na platformě Blazor a ASP.NET Core.

Dále budou vytvořeny dvě reálné webové aplikace založené na frameworku Blazor a ASP.NET Core MVC. Budou zvoleny vhodné metriky měření a na základě vyhodnocení provedeného experimentu porovnány obě technologie. Na základě výsledků teoretické a praktické části budou shrnuty poznatky a formulovány závěry.

3 Teoretická východiska

3.1 Webové technologie

Webové aplikace jsou založeny na webových technologiích. Jedná se o obecný pojem a představuje široké spektrum pojmů. Zejména se jedná o protokoly, datové formáty, knihovny či frameworky, případně standardy. Aby webová stránka vzhledově vypadala a fungovala, jsou používané tři klíčové jazyky – značkovací jazyk HTML, stylovací jazyk CSS a programovací jazyk JavaScript. [1]

Webová komunikace probíhá zejména na protokolech TCP/IP a UDP. Pro přenos webových dat je primárně využíván protokol HTTP nebo jeho zabezpečená verze HTTPs. Protokol HTTP(s) je založen na výpočetním modelu client-server s principem požadavek-odpověď. Asynchronní načítání nebo odesílání dat je možné realizovat pomocí AJAX (kapitola 3.1.3) přístupu odesláním požadavku na Web API založené na architektuře REST (kapitola 3.1.3). Výstupem Web API je ve velké míře jeden z datových formátů JSON a XML (kapitola 3.1.4) nebo v menší míře HTML. [1]

Webové technologie se dělí na client-side a server-side. Pro client-side webové aplikace postavené na novém standardu WebAssembly (kapitola 3.1.1.4) lze použít prakticky jakýkoliv server-side jazyk. [2]

Client-side je tzv. označení pro webový prohlížeč uživatele. Navštívením webové stránky uživatel vytváří událost ve webovém prohlížeči, kde vznikne HTTP(s) požadavek, který je odeslán na webový server. Webový server (server-side) zpracovává HTTP(s) požadavek a vrací HTTP(s) odpověď webovému klientovi. Client-side slouží pro zobrazení serverových dat, zatímco server-side obsahuje logiku webové aplikace a transformuje data do tvaru, které může klient přečíst. [3]

3.1.1 Client-side

Webové prohlížeče podporují nativně značkovací jazyky (kapitola 3.1.1.1 - HTML), dále stylovací jazyk (kapitola 3.1.1.2 – CSS) a z programovací jazyků pouze JavaScript (kapitola 3.1.1.3). Další programovací jazyky podporují pomocí uvedeného WebAssembly standardu (kapitola 3.1.1.4). JavaScript byl původně vyvinutý firmou Netscape, který je možné využít i mimo webový prohlížeč jako server-side jazyk, např.: Node.js (kapitola 3.1.2 Server-side). [4]

Z JavaScriptu vychází mnoho programovacích jazyků, které se následně transpilují do JavaScriptu jako výstup. Jeden z nich je např. TypeScript, vyvinutý a udržovaný firmou Microsoft. [4]

3.1.1.1 HTML

První návrh značkovacího jazyku HTML (HyperText Markup Language) pochází z roku 1989. Autory byli Tim Berners-Lee a Rober Caillau, kteří pracovali pro CERN. HTML je standardizovaný značkovací jazyk pro vytváření struktury webových stránek a aplikací. HTML značky reprezentují bloky, formátovaný text, obrázky, odkazy, pole pro formuláře a další jiné struktury. HTML značky je možné doplnit o atributy, které modifikují značku. Atribut může modifikovat základní funkci nebo poskytují funkcionalitu k určitým značkám, které bez nich nemohou správně fungovat. [5] [6]

Roku 1994 byly vyvinuty první webové prohlížeče, které měly své další značky. To mělo za následek jiné zobrazení webové stránky v konkurenčních webových prohlížečích např. v Internet Exploreru a Google Chrome. Z tohoto důvodu bylo potřeba vytvořit soubor definicí, které oddělí logické značky od vizuálních. Výsledkem byl vznik kaskádových stylů CSS (kapitola 3.1.1.2 - CSS). Po vzniku mnoha různých verzí HTML se konsorcium W3C ujalo vývoje. [5]

Když webový prohlížeč vytvoří požadavek prostřednictvím URL adresy pro načtení stránky nebo aplikace, první je vrácen HTML dokument. Dokument může obsahovat odkazy na CSS soubory a JavaScript soubory. [6]

V roce 1999 vznikla verze HTML 4.01, která byla v roce 2014 nahrazena verzí HTML 5. Tato verze lépe odráží potřeby současné doby. Například příchod tagu **<video>** je označováno za konec technologie Flash pro přehrávání videa. Stejně tak nový tag **<audio>**. Došlo k ukončení kosmetických tagů jako **<font>** či ** **. Současná verze HTML je 5.2 z roku 2017. [7]

3.1.1.2 CSS

Kaskádové styly – CSS popisuje zjednodušenou formou visuální formátování modelu pro všechny HTML značky. Jde o kolekci metod upravující grafickou podobu webové stránky např.: barva textu, velikost bloku, ohraničení, vnější a vnitřní odsazení. Protože jsou kaskádové, umožňují na sebe vrstvit definice stylu, ale platí pouze poslední. Syntaxe se skládá ze tří základních prvků: selektor, vlastnost, hodnota. CSS postrádají podporu pro podmínkovou logiku a další vlastnosti programovacích jazyků. První verze byla vydána jako doporučení od

konsorcia W3C v prosinci 1996. CSS 2 staví na původní verzi a bylo vydáno v květnu 1998. Druhá verze přinesla podporu pro specifická media – např. tiskárna. [8] [6]

CSS 3 bylo v červnu 1999 vydáno jako nejnovější verze. Hlavním rozdílem ve vývoji je modulárnost (separátní dokumenty) - umožňuje definovat nové CSS styly v separátním modulu, které jsou později implementované ve webových prohlížečích. S modulárností jsou specifikace jako selektory, hodnoty typů a jiné vyvíjený samostatně a poté vydávané ve formě standardů. [9]

CSS 3. verze přinesla dle [9]:

- Barvy podporují barevné prostory RGBA, HSL, HSLA, přechody.
- Responzivitu v podobě media queries (definování při jaké podmínce platí styly).
- Animace.
- Selektor atributů.
- Funkce *calc().*
- Webfonty.
- A další.

3.1.1.3 JavaScript

JavaScript je standardizovaný dynamický, interpretovaný, programovací jazyk. Standardem pro JavaScript je ECMAScript. JavaScript je možné definovat ve vlastním souboru s příponou *.js* nebo v HTML bloku <*script*>. Mezi nejběžnější úkoly JavaScriptu ve webovém prohlížeči patří: vybrání HTML prvku s následným získáním nebo aktualizací hodnoty, získání nebo odeslání dat přes Web API a validace formulářů. [6] [10]

Od roku 2012 podporují plně všechny moderní webové prohlížeče minimálně ECMAScript verze 5.1. V roce 2016 byla vydána šestá verze, která se oficiálně nazývá ECMAScript2015 (interně ES6). Po vydání ECMAScriptu2015 je v každoročním cyklu vydáván nový standard. [10]

Každý webový prohlížeč podporující JavaScript obsahuje svůj JavaScript engine spouštějící Javascript kód. Kód je zkompilován do binárního kódu, následně je předán interpreteru, který jej převede na strojový kód srozumitelnému pro procesor. [11]

JavaScript je tzv. „first-class functions“ jazyk, který zachází s funkcí stejně jako s proměnnou. Funkce lze tedy předávat pomocí argumentu nebo můžou být vracené jinou funkcí. JavaScript je možné využít i mimo webový prohlížeč v prostředí např.: Node.js, Apache CouchDB a Adobe Acrobat. [10]

Jako v každém jiném programovacím jazyce existuje mnoho knihoven a frameworků pro daný programovací jazyk. Mezi nejpopulárnější knihovnu JavaScriptu patří jQuery. [1]

jQuery

John Resig začal vyvíjet v roce 2005 počáteční verzi jQuery s vydáním následujícího roku. Knihovna měla dvě hlavní propozice. První – poskytnout ergonomickou API pro manipulování stránkou. Svým způsobem jQuery poskytlo velice mocné nástroje pro výběr prvků na stránce. Kromě výběru prvků na základě třídy – *class* a identifikátoru – *id*, poskytlo jQuery výběr přes komplexnější výrazy jako je výběr prvků na základě vztahů s ostatními prvky. [12]

Druhou propozicí byla abstraktnost. Implementace pro specifický webový prohlížeč musí zároveň fungovat ve všech ostatních webových prohlížečích. Zpětně, kdy nebylo jádro Chromium na vrcholu a každý prohlížeč se choval jinak, bylo velice obtížné zajistit kompatibilitu ve všech prohlížečích. [12]

jQuery mělo jednu z největších komunit tvořící ekosystém webů, pluginů a frameworků, které byly založeny právě na této knihovně. V posledních několika letech se stav jQuery jako jedničky mezi nástroji pro vývoj webů vytratil, protože funkce pro výběr byly implementované do JavaScript API. [12]

JavaScript frameworky

JavaScript frameworky poskytují různé funkce, především strukturu organizující části webu. Výhodou je celková efektivita a rychlejší vývoj za cenu dodržování určitých pravidel. Byť bylo jQuery nejpopulárnější svého času, chyběla mu architektura pro zpracování dat mezi sdílenými komponentami. Prvním populárním frameworkem se stal AngularJS (dále jen Angular). [13]

Angular

Angular přišel na trh v říjnu roku 2010. Je založen na návrhovém vzoru MVC (Model View Controller). Nabízí funkce dle [13]:

- Two-way data binding.
- Dependency injection.
- Routing.
- A další.

Framework pomohl vývojářům vyřešit mnoho problémů, kterým čelili při vývoji webových projektů. Framework se snažil být užitečný, ale s postupným vývojem pociťovali

vývojáři frustraci. To vedlo vývojový tým k redesignu celého frameworku a začali jej nazývat Angular 2. Nový framework byl nekompatibilní s původním a bez podpory možnosti migrace. Tato událost vedla k odlivu mnoha vývojářů. Od této doby zažívá Angular úpadek. [13]

React

Po velké krizi Angularu v květnu 2013 byl na JavaScriptové konferenci oznámen nový framework – React. Vytvořil jej softwarový inženýr Jordan Walke pracující pro Facebook. React byl vytvořen z důvodu velkého množství bugů ve Facebookové aplikaci. [13][14] Vývojáři na konferenci byli ohromeni inovativními funkcemi dle [13]:

- virtualní DOM.
- one-way data flow,
- flux pattern.

Vue.js

Vue.js je další z progresivních JavaScript frameworků. První verze byla vydána v únoru 2014 vývojářem Evan You. Vue.js je střední cesta mezi velmi flexibilním Reactem a umíněným Angularem. Framework je poměrně lehký (v rámci provozu), obklopený ekosystémem spravovaným týmem Vue. [13] [15]

3.1.1.4 WebAssembly

JavaScript jako jediný dostupný jazyk pro webový prohlížeč představoval pro některé vývojáře problém. Začaly se tedy rodit transpilery – neboli překladače jiného jazyku do JavaScriptu. Jeden z nich je TypeScript zmíněný v úvodu kapitoly 3.1.1 – Client-side. [1]

Vznik nového standardu – WebAssembly se stal nutností. Jde o binární formu kódu, který může běžet v moderních webových prohlížečích. Poskytuje nové funkce a především dosahuje vyššího výkonu. WebAssembly není určeno jako psaní kódu, nýbrž je navrhnuto jako efektivní cíl kompilace jakéhokoliv úrovňového programovacího jazyku jako jsou C, C++, C#. Výstupem je optimalizovaný bytecode, který je menší pro stažení vůči JavaScriptu a rychlejší na spuštění. Velkou výhodou je absence nutnosti parserování. [16]

Jde o nový způsob, jakým psát klientské webové aplikace, které nikdy předtím nemohly být napsané. WebAssembly moduly mohou být využívány přes JavaScript. Frameworky v JavaScriptu jako např. React, mohou libovolně volat WebAssembly kód, který nabízí masivní výkon. V současné době WebAssembly neumožňuje manipulaci s DOM prvky a je nutné je

spravovat přes JavaScript. Dále chybí integrovaný garbage-collector, který může být součástí klientské aplikace za cenu zvýšení velikosti klientské aplikace. [16]

Oficiálně bylo WebAssembly oznámeno v roce 2015. Hlavní prohlížeče jako Google Chrome (57+), Edge (16+), Firefox (53+) a Safari (11+) měly částečnou podporu od roku 2017. V roce 2020 přes 90 % uživatelů používají prohlížeč s podporou WebAssembly. Internet Explorer v poslední verzi s označením 11 nepodporuje WebAssembly. [17]

3.1.2 Server-side

Serverová část aplikace je umístěna na tzv. webovém serveru. Webový server je služba zpracovávající komunikaci protokolu HTTP(s) – např.: Internet Information Service, Apache, nginx nebo LiteSpeed. V případě webových stránek se jedná převážně o statický obsah a je odeslán soubor. Při požadavku u webové aplikace např.: PHP soubor je webovým serverem spuštěn tzv. server-side processing. Ten spouští modul zpracovávající PHP kód a následně vrací obsah výstupu. [3]

Serverová aplikace, kde přichází požadavky od uživatelů z prohlížeče, není nijak limitovaná. Může být prakticky napsaná v jakémkoliv jazyce. Mezi nejpopulárnější patří dle [3]:

- PHP
- Python (Django)
- Ruby (Ruby on Rails)
- C# (ASP.NET, kapitola 3.3 - ASP.NET Core)
- Java
- JavaScript (Node.js)
- C, C++
- Případně méně typické – Go, Scala, Perl

Programátoři využívají webové frameworky, které obsahují kolekci funkcí, objektů, pravidel a jiných vlastností navržené pro běžné problémy ve vývoji webových aplikací. Zejména programátor šetří čas a má jednodušší vývoj. [3]

Node.js

Node.js je ne-webové JavaScript prostředí založené na Chrome V8 JavaScript enginu. Prostředí umožňuje spouštět JavaScript jako server-side jazyk. Prostředí vzniklo z jednoho prostého důvodu – vývojáři chtěli psát jak front-end, tak back-end ve stejném programovacím jazyce. [1]

Ruby on Rails

Ruby on Rails neboli jednoduše Rails je open-source web framework. Framework byl vytvořený v programovacím jazyce Ruby roku 2003 Davidem Heinmeier Hanssonem. Oficiálně byl zdrojový kód uvolněn až v červnu 2004. [18] Rails je postaveno na 3 základních principech dle [18]:

- programovací jazyk Ruby,
- MVC architektura,
- štěstí programátora.

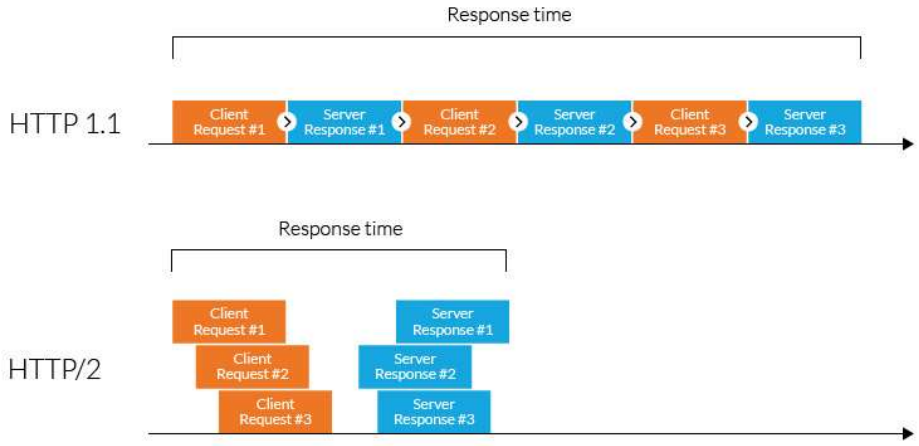
3.1.3 Protokoly a komunikace

HTTP(S)

HTTP neboli HyperText Transfer Protocol je protokol podléhající pod WWW (World Wide Web). Byl vytvořen ve stejné době jako první HTML standard kolem roku 1991. Slouží k přenosu hypertextových dokumentů ve formátů HTML, XML a jiných souborů. Pro komunikaci se využívá port 80 a pro HTTPs port 443. Protokol HTTP sám o sobě není z pohledu bezpečnosti důvěryhodný, proto vznikla verze HTTPs. Jedná se o kombinaci protokolu HTTP společně s protokolem SSL nebo TLS. [19]

V současné době se používá pro přenos souborů, které nesouvisí s webem pomocí rozšíření MIME (Multipurpose Internet Mail Extensions). [19]

Další verze, které byly vyvinuty jsou HTTP 1.1 (1997) a 2.0 (2015). Verze 2.0 přinesla několik vylepšení, zejména Multiplexing zachycený na obrázku č. 1. Funkce Multiplexingu umožňuje vyřizování požadavků paralelně přes jedno TCP připojení. [19] [20]



Obrázek 1: Multiplexing protokolu HTTP (zdroj: [20])

V současné době se standardizuje verze 3.0. Verze 3.0 přinese změnu, kdy nebude komunikovat přes transportní vrstvu TCP/TLS, ale QUIC. Verzi 3.0 si lze již nyní vyzkoušet jako experimentální technologii. [19]

HTTP protokol obsahuje několik dotazovacích metod, které indikují, jaká akce má být na základě požadavku provedena. Názvy dotazovacích metod jsou citlivé na velká písmena. [21] Seznam HTTP metod dle [21]:

- GET – slouží pro získávání dat.
- HEAD – funguje totožně jako GET, ale neobsahuje obsah odpovědi.
- POST – slouží k odeslání dat na server, např. formulář.
- PUT – slouží k nahrazení, případně aktualizaci cílových zdrojů na serveru.
- DELETE – slouží pro smazání dat.
- CONNECT – slouží k navázání spojení.
- OPTIONS – slouží k popisu komunikace.
- TRACE – slouží pro ověření cesty.
- PATCH – slouží k částečné aktualizaci cílových zdrojů.

REST

REST je akronym pro REpresentational State Transfer. Jde o architekturu pro distribuované hypermedia systémy využívající bezstavový protokol, zejména HTTP(s). Architektura je využívaná především ve webových službách např.: ASP.NET Web API nebo ASP.NET Core pro předávání dat mezi klientem a serverem [22]. Aby bylo splněno, že se jedná o RESTful architekturu, musí být splněno 6 vlastností dle [22] [23]:

1. Client-server
    a. Odloučením uživatelského rozhraní od datového uložiště se zvýší portabilita uživatelského rozhraní napříč platformami.
2. Bezstavový
    a. Každý požadavek z klienta na server musí obsahovat všechny potřebné informace k pochopení požadavku.
3. Cache
    a. Data odpovědi na požadavek musí být implicitně nebo explicitně označena jako cachovatelná nebo necachovatelná.
4. Jednotný interface
    a. Využitím obecnosti interface se celková architektura zjednodušuje, lze ji aplikovat na jiných enginech.

5. Vrstvený systém
    a. Vrstvený systém umožňuje vidět pouze architekturu komponenty s kterou reagujeme, nikoliv za ní.
6. Kód na vyžádání (volitelný)
    a. Dovolí klientovi stáhnout a spustit kód ve formě skriptů.

Klíčové slovo pro REST jsou **prostředky**. Jakákoliv informace, např. dokument, obrázek, kolekce, lze označit za prostředek, předně se používají formáty HTML, XML (kapitola 3.1.4), plain text, PDF, JPEG, JSON (kapitola 3.1.4). [22]

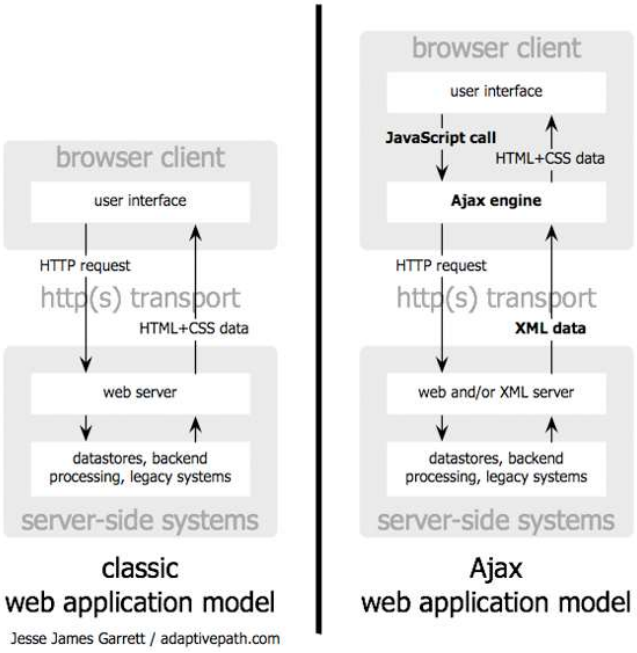
Metody, které lze dle [22] využít pro prostředky, jsou následující:

- GET
- PUT
- POST
- DELETE

AJAX

AJAX (**A**synchronous **J**avaScript **A**nd **X**ML) - nejedná se o technologii, jako spíše o přístup využívající na prvním místě protokol HTTP(s). Termín byl prvně užit Jamesem Garrettem v roce 2005. Zjednodušeně se jedná o využití mnoha existujících technologií, jako je HTML, CSS, JavaScript, XML, DOM (Document Object Model) a v prvé řadě hlavně XMLHttpRequest objektu. Využitím těchto technologií lze docílit aktualizaci UI (User Interface) bez potřeby znovunačtení celé stránky. Obecně tedy AJAX komunikuje se serverem, získá data, a následně aktualizuje část UI bez znovunačtení stránky. Tato technika byla průkopníkem ve zvýšení interaktivity na webu, a obzvláště uživatelské přívětivosti. [24]

Před rokem 2005 bylo velice obtížné zajistit komunikaci mezi klientem a serverem. Vývojáři využívali skryté tagy **<iframe>** k získání dat ze serveru pro klienta. Roku 2005 napsal James Garrett článek „AJAX: a new approach to Web applications“. Klíčem technologie použité v AJAXu byl XMLHttpRequest (XHR), který vynalezl Microsoft a poté se začal užívat v prohlížečích. XHR má schopnost získat data ze serveru a použít je v klientovi bez pomoci jiných technologií. Před XHR museli vývojáři používat technologie jako Java Applets nebo Flash. Rozdíl klasického a AJAX požadavku je možné vidět na obrázku č. 2. [24] [25]



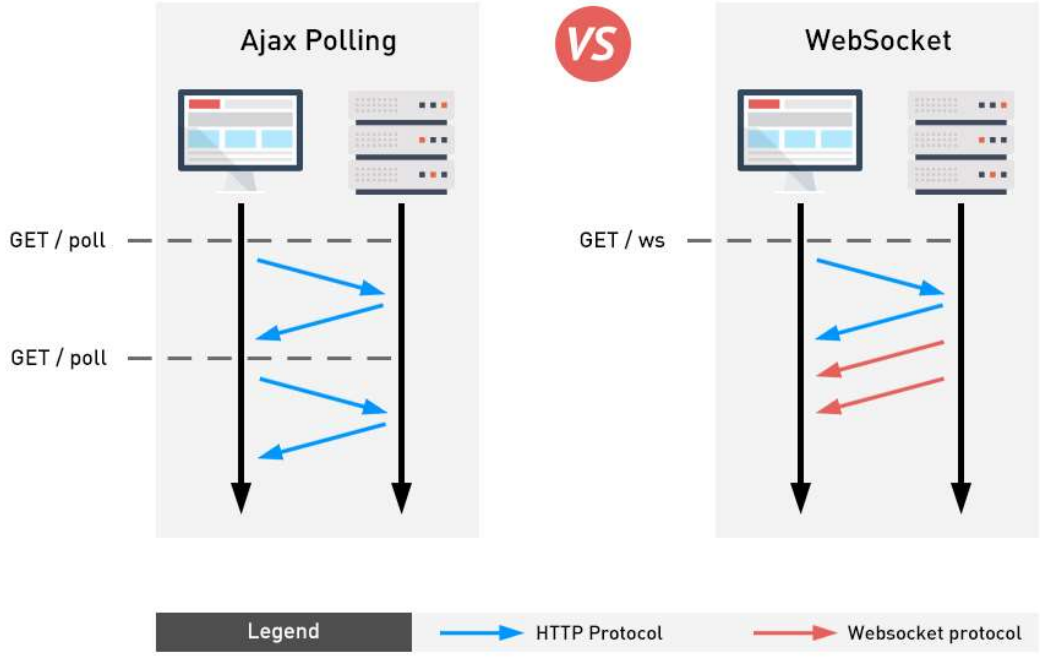
Obrázek 2: Klasický (synchronní) vs. AJAX (asynchronní) (zdroj: [25])

WebSockets

Během poloviny roku 2008 vývojáři Michael Carter a Ian Hickson přišli s nápadem vytvořit nový webový standard pro moderní real-time obousměrnou komunikaci. Zrodil se název WebSocket. Myšlenka doputovala až k W3C HTML standardu. V roce 2010 byl Google Chrome 4 první prohlížeč, který plně podporoval WebSockety. S příchodem roku 2011 vyšel oficiální standard: RFC 6455 – The WebSocket Protocol. V současnosti všechny moderní prohlížeče plně podporují WebSocket, včetně Internet Explorer verze 10. Mobilní prohlížeče pro operační systémy iOS a Android přišly s podporou až v roce 2013. Většina IoT neboli Internet of Things zařízení běží právě na WebSocketu. [26]

WebSocket se skládá z tenké transportní vrstvy postavené na TCP/IP. Princip je poskytnout efektivní a co nejblíže surové TCP komunikační vrstvy pro vývojáře webových aplikací. Je nutné poznamenat, že bylo přidáno pár prvků abstrakce kvůli způsobu fungování webů. [26]

WebSocket podporuje několik subprotokolů jako např. MQTT či WAMP. Příkladem použití WebSocketu může být real-time komunikace. Na obrázku č. 3 je možné vidět srovnání dotazů za pomocí AJAXu a WebSocketu. Při použití WebSocketu je odeslán pouze jeden HTTP dotaz, poté je schopen server odesílat klientovi data bez nutnosti dalších HTTP dotazů jako v případě AJAXu. [26] [27]



Obrázek 3: AJAX vs WebSocket (zdroj: [27])

3.1.4 Datové formáty

Dnes a denně většina webových aplikací spoléhá na jeden z dvou základních standardů přenosu zpráv: JSON a XML. S příchodem AJAXu se ujal vedení standard XML, protože je sám označován, jak již bylo zmíněno v názvu. V současné době je JSON jeden z nejrozšířenějších datových formátů, převzal vedení po XML jen několik let zpět (kolem roku 2013/14). [28]

JSON

V dubnu 2001 přišel Douglas Crockford a Chip Morningstar s prvním JSON (JavaScript Object Notation) záznamem. Crockford a Morgningstar se pokoušeli o vybudování AJAX aplikace dříve, než slovo AJAX vzniklo. Byť se pokoušeli o aplikaci jako takovou, háček spočíval v podpoře prohlížečů. Našli způsob, jakým dostat data do aplikace po načtení stránky, ale nenašli způsob, aby to fungovalo ve všech prohlížečích. [28]

Garrett v termínu AJAX značil „X“ jako XML, poukázal na to, že JSON je zcela přijatelná alternativa. [28]

S popularizací JavaScriptu se v posledním desetitisíciletí stal datový formát JSON vůdcem. V současné době je JSON formát více trendový než jakýkoliv jiný. [28] [29]

XML

XML formát byl vydán v roce 1998 konsorciem W3C. Jedná se o značkovací jazyk podobně jako HTML. XML je stejně jako JSON datový formát, který je lidsky čitelný. [28]

XML proti JSONu podporuje další vlastnosti dle [30] jako např.:

- namespace,
- beztypový,
- podpora komentářů,
- různé znakové sady.

XML je využívaný pro datové typy jako např.: vektorový obrázek (.svg), grafy a další. Porovnání JSON a XML zprávy je možné vidět na obrázku č. 4. [30]

JSON

```
{
  "jmeno": "Lukáš",
  "prijmeni": "Veteška",
  "prace": "Diplomová práce",
  "fakulta":
  {
    "id": "1",
    "nazev": "Provozně Ekonomická Fakulta",
    "zkratka": "PEF",
    "skola":
    {
      "nazev": "Česká Zemědělská Univerzita v Praze",
      "mesto": "Praha"
    }
  }
}
```

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
   <fakulta>
      <id>1</id>
      <nazev>Provozně Ekonomická Fakulta</nazev>
      <skola>
         <mesto>Praha</mesto>
         <nazev>Česká Zemědělská Univerzita v Praze</nazev>
      </skola>
      <zkratka>PEF</zkratka>
   </fakulta>
   <jmeno>Lukáš</jmeno>
   <prace>Diplomová práce</prace>
   <prijmeni>Veteška</prijmeni>
</root>
```

Obrázek 4: Porovnání JSON a XML formátu (zdroj: Vlastní)

3.2 Nástroje pro testování rychlosti a výkonu webových stránek

Kapitola popisuje dostupné nástroje, kterými se dají výkonově měřit odlišené webové aplikace. Nejčastějšími parametry jsou odezva, velikost souborů, počet požadavků a doba vykreslení. [31]

3.2.1 GTmetrix

GTmetrix je online aplikace určená pro testování rychlosti webové stránky. Nabízí zdarma pohled na výkon webu. Výstupem testu je výkonnostní skóre reprezentované známkami A až F. Nástroj dále nabízí detaily typu: doba načtení stránky, celková velikost a počet požadavků. [31]

3.2.2 WebPage Test

WebPage Test je další online aplikace určená jako GTmetrix pro testování rychlosti webové stránky. Test nabízí uživateli si vybrat, z které polohy na světě má být proveden. Aplikace dále bere v úvahu typ prohlížeče, zařízení, typ připojení a stav uživatelské cache pro zajištění výsledků. Nabízí mimo jiné rozšířené testování s visuálním porovnáním či sledováním toku požadavků přes internet. [31]

3.2.3 Google PageSpeed Insight

Google PageSpeed Insight je nástroj pro testování výkonu webu. Nástroj je dostupný zdarma. Po vložení URL adresy provede analýzu s výstupem celkového skóre webu v rozmezí 0–100. Podle skóre vyhodnotí, zda je web rychlý, průměrný či pomalý. Výstupem je dále rozlišené skóre pro desktop a mobil. Nejvíce přínosná funkce nástroje jsou rady, kterým směrem optimalizovat pro zvolený parametr. [31]

3.2.4 Google Chrome DevTools

Nástroj Google Chrome DevTools je dle [32] součástí webového prohlížeče Google Chrome. Je možné jej spustit aktivací klávesové zkratky *F12*. DevTools bylo vyvinuto za účelem debuggování (ladění) webové stránky v prohlížeči. Obsahuje tedy dle [32] funkce jako:

- prohlížení zdrojového kódu,
- breakpointy uvnitř zdrojového kódu,
- monitorování požadavků,
- monitorování využití paměti stránkou,
- nástroj na bázi Google PageSpeed Insight – Lighthouse,
- a další.

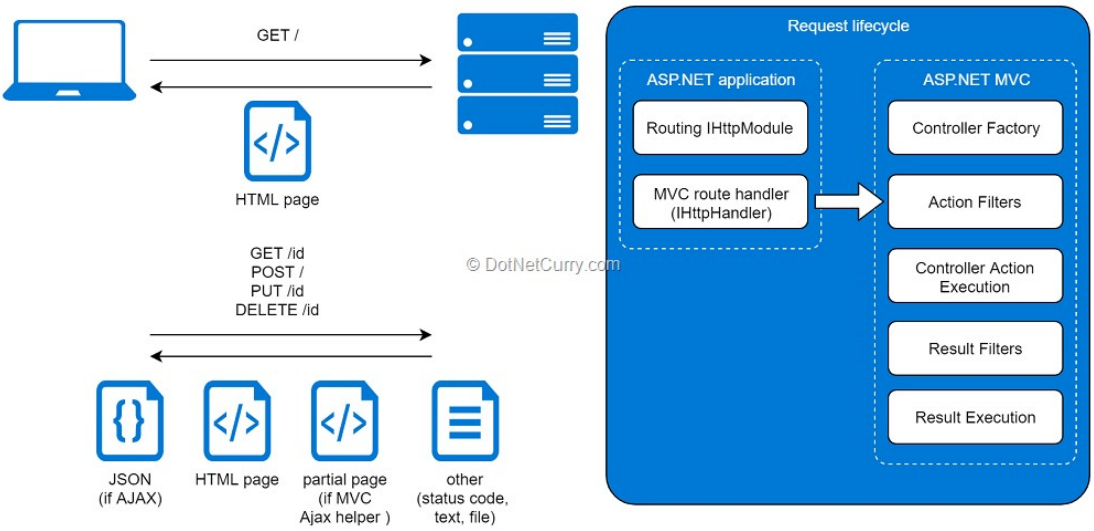
3.3 ASP.NET Core MVC

ASP.NET (Active Server Pages) je webový framework s historií počínající rokem 2002. Do roku 2008 byla éra ASP.NET Web Forms, která připomínala vývojáři spíše Windows Forms. [33]

Novým směrem ASP.NET se v roce 2008 dle [34] stalo ASP.NET MVC (více v kapitole 3.3.1 – MVC). Microsoft tímto reagoval na vývoj prostředí WWW. Bylo to také poprvé, kdy byl framework open-source. Klíčové funkce ASP.NET MVC dle [34]:

- Kompletní kontrola nad generovaným HTML.

- ASPX view engine (později nahrazen Razor enginem).
- Kompletní kontrola nad URL a navigací.
- Podpora existující ASP.NET infrastruktury (Cache, Session, Moduly, Handelry, IIS, …).



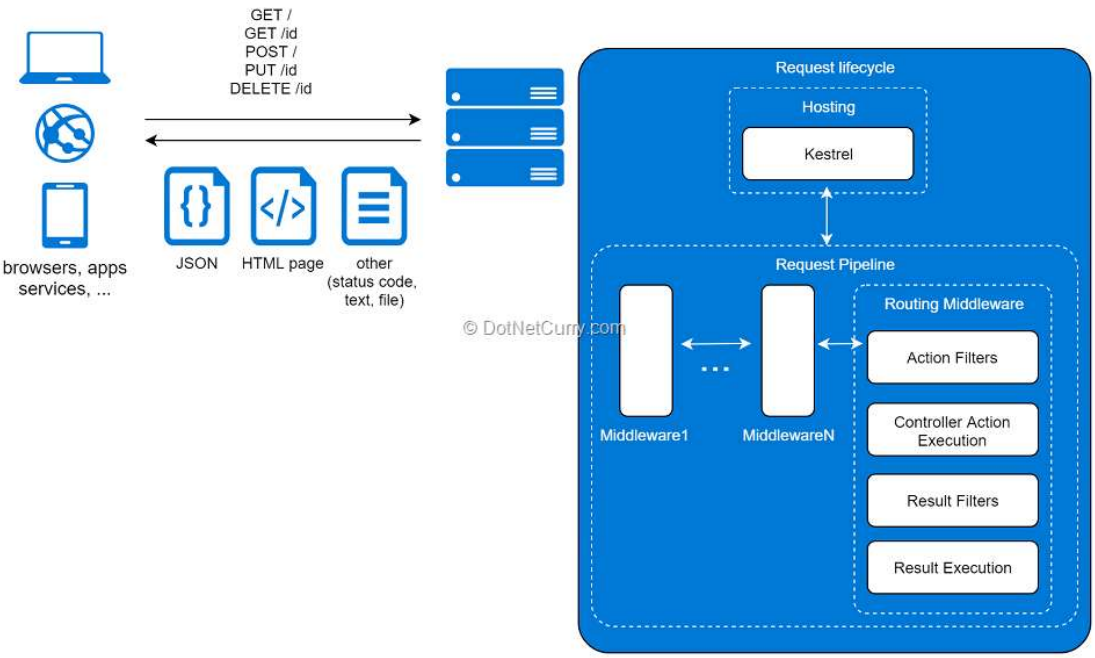
Obrázek 5: Životní cyklus požadavku ASP.NET MVC (zdroj: [34])

Životní cyklus požadavku ASP.NET MVC je zachycen na obrázku č. 5. S příchodem třetí verze si vývojáři a firmy uvědomili, že se MVC uchytilo a zůstane používané. Další verze s pořadovým číslem čtyři v roce 2012 přinesla podporu REST služeb. Poslední verze (MVC 5) poskytla vlastní knihovnu ASP.NET Identity pro autorizaci a autentizaci a další vylepšení. [34]

V době vydání MVC 5 se mohlo zdát, že se framework ustálil na dlouhé období. Vývojový tým ohlásil ASP.NET vNext (interní název), později nazývaný ASP.NET 5. ASP.NET vNext mělo jeden velkolepý cíl – stát se cross-platform. V květnu 2016 došlo k poslednímu přejmenování, a to na jméno – ASP.NET Core. Společně s přejmenováním došlo k ohlášení .NET Core frameworku, ze kterého vychází a je taktéž cross-platform. [35]

Oficiálně bylo ASP.NET Core vydáno v červnu 2016. Vytváření aplikací bylo pocitově velmi podobné původnímu ASP.NET MVC. Funkce, které se změnili dle [35]:

- Cross-platform.
- Nové nástroje pro zjednodušený vývoj.
- Cloud-ready.
- Integrovaná podpora pro NuGet balíčky.
- Dependency Injection.
- Schopnost hostovat v IIS či **vlastním procesu**.
- Sloučení Web API do jednoho projektu.



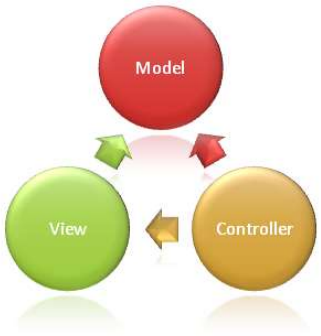
Obrázek 6: Životní cyklus požadavku ASP.NET Core (zdroj: [35])

Na obrázku č. 6 je zobrazen životní cyklus požadavku ASP.NET Core, na kterém je možné vidět vůči obrázku č. 5, že obsahuje vlastní webový server – Kestrel a pipeline je založena na middleware komponentách. Následovala verze ASP.NET Core 2 vydána v dubnu 2017. Nejžhavější novinou byla funkce **Razor Pages**, která znamenala znovuoživení návrhového vzoru Page Controller známý z Web Forms. Mezi další nové funkce přibyla vlastnost provádět práci na pozadí – background processes. Minoritní verze 2.1 přinesla přepsaný SignalR (framework založený na WebSocketu). [35]

23. září 2019 byla vydána verze ASP.NET Core 3, která přinesla server-side framework Blazor (viz kapitola 3.4). Nová verze dále přidala podporu pro gRPC, dále ve výchozím stavu podporu HTTP/2 a mnoho dalších funkcí. [36]

3.3.1 MVC

MVC je návrhový vzor, který dělí aplikaci do 3 skupin: Models (objekty s daty předávané mezi Views a Controllers), Views (stránky, výstup HTML) a Controllers (třídy zpracovávající požadavek). Tento návrhový vzor odděluje jednotlivé vrstvy, výsledkem je přehlednější architektura. V tomto návrhovém vzoru je uživatelský požadavek (HTTP request) přesměrován do Controlleru, který zpracuje požadavek na danou uživatelskou akci. Controller je místo, kde probíhá business logika aplikace. Výstupem Controlleru je Model s daty, který je předáván do View, kde probíhá zobrazení dat. [37]



Obrázek 7: Návrhový vzor MVC (zdroj: [37])

Model reprezentuje stav aplikace a jakoukoliv logiku či operaci, která by měla být provedena. View část se stará o prezentaci obsahu uživatelského rozhraní. Využíván je Razor engine, který transformuje .NET kód do HTML značek. Je doporučováno, aby View část obsahovala minimum aplikační logiky, pouze logiku k prezentování obsahu. Controller zastupuje část, která zpracovává uživatelskou interakci. Pracuje s modelem a poté vybírá, který View použije pro vykreslení. V návrhovém vzoru MVC je Controller vstupním bodem. [37]

3.3.2 Routing

ASP.NET Core MVC je postaveno na ASP.NET Core routingu. Jedná se o ohromně mocný nástroj k mapování URL. Umožňuje dosáhnout srozumitelnému a snadno vyhledávajícímu URL. Zvolení URL vzoru napomáhá výborně pro SEO a generování odkazů bez potřeby znalosti organizace souborů na webovém serveru. [37]

```
[Route("api/[controller]")]
public class ProductsController : Controller
{
    [HttpGet("{id}")]
    public IActionResult GetProduct(int id)
    {
      ...
    }
}
```

Zdrojový kód 1: Routing (zdroj: [37])

Ve zdrojovém kódu č. 1 je možné vidět použití atributu *Route*, kde *[controller]* je klíčové slovo v run-time nahrazené názvem Controlleru (Products). Atribut *HttpGet* značí očekávání *HTTP* požadavku typu *GET* s parametrem *id*, který je možné vidět jako parametr funkce *GetProduct* typu *int*. Při chybě konverze parametru dojde k zahození požadavku. Příkladové volání funkce *GetProduct* pomocí adresy prohlížeče – *https://domena.cz/api/Products/1*. [37]

3.3.3 Model binding

Funkce model binding zajišťuje konverzi vstupních dat z klientského požadavku (formulář, route data, query parametry, HTTP hlavička) do objektů, které může dále zpracovávat Controller. Controller tedy nemusí zjišťovat, co přichází za data, ale jednoduše je má jako parametry metody. [37]

```
public async Task<IActionResult> Login(LoginViewModel model, string returnUrl = null) { ... }
```

Zdrojový kód 2: Model binding (zdroj: [37])

Ve zdrojovém kódu č. 2 je možné vidět definici asynchronní metody Login. Model binding provede konverzi vstupních parametrů z požadavku. Je možné explicitně specifikovat za pomocí speciálních atributů, ze které části je očekávat. Např. kdyby byla definice parametru následující: *[FromBody] LoginViewModel model* – Model binding bude hledat vstupní data pro model *LoginViewModel* pouze v těle požadavku. Dále dostupné jsou dle [37]:

- FromQuery
- FromRoute
- FromForm
- FromHeader

3.3.4 Validace modelu

Validace je další funkcí, kterou ASP.NET Core MVC nabízí. Validace modelu probíhá prostřednictvím dekorace vlastností modelového objektu validačními atributy. Typy atributů dle [37]:

- požadovaný,
- e-mailová adresa,
- telefon,
- datový typ;
    - Heslo
    - Datum
- číselný rozsah,
- délka řetězce,
- porovnání dvou atributů (heslo),
- URL,
- regulární výraz,

- případně vlastní atribut.

V Controlleru je možné provést validaci pomocí vlastnosti *ModelState.IsValid* datového typu *bool*. [37]

3.3.5 Dependency Injection

Dependency Injection dále jen DI je návrhový vzor a technika k získání požadovaných služeb pro třídu. Controllery mohou zažádat o služby pomocí konstruktoru. DI lze použít i ve Views souborech syntaxí *@inject*. [37]

Veškeré služby podléhající DI musí být zaregistrované v metodě *ConfigureServices*. Registraci lze dělit na 3 typy dle [37]:

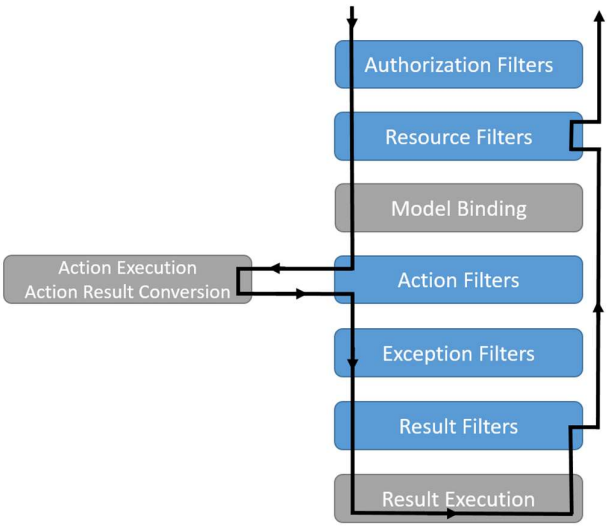
- Transient – služba je vytvořena při každém požadavku z DI. Vhodné pro bezstavové služby.
- Scoped – služba je vytvořena a žije po celou dobu klientského požadavku.
- Singleton – služba je vytvořena 1x po celou dobu a sdílí se mezi klientskými požadavky.

3.3.6 Filtry

Filtry pomáhají vývojářům zapouzdřit jisté obavy, jako jsou chybové hlášky nebo autorizace. Filtry umožňují spustit vlastní kód před a po logické akci metody. Mohou být konfigurovány, aby běžely jen v krajních případech. Lze je aplikovat na metody nebo přímo na Controllery. Nejběžnější filtr je *[Authorize]*, který se aplikuje převážně na celou třídu daného Controlleru. Mezi další nejčastější filtry patří dle [38]:

- Areas – definice oblasti, např. admin.
- AllowAnonymous – při definici *[Authorize]* atributu na Controller je možné povolit i anonymnímu uživateli přístup na specifickou metodou tímto filtrem.

Na obrázku č. 8 lze vidět diagram interakce filtru v pipeline. Na počátku probíhají autorizační filtry, které jsou následované zdrojovými filtry. Třetím krokem je automatický proces Model Binding. Poté dochází ke zpracování před akčním filtrem, vykonání akce a zpracování akčního filtru po vykonání akce. Dalším krokem jsou Exception filtry, kdy dochází k zachycení neošetřených výjimek. Předposledním krokem ve směru dolů jsou výstupové filtry a v závěrečném kroku probíhá zpracování výstupu. [37] [38]



Obrázek 8: Diagram pipeline filtrů (zdroj: [38])

3.3.7 Oblasti (Areas)

Oblasti umožňují rozdělit velkou aplikaci do menších skupin. Například oddělení administrace od návštěvnické části, případně návštěvnickou část rozdělit ještě do menších oddílů. [37]

3.3.8 Web API

Platforma dále nabízí velkou podporu při budování Web API. Služby se mohou rozšířit na velkou škálu klientů, ať už ve webovém prohlížeči či mobilním zařízení. Framework podporuje datové formáty JSON a XML. Pro podporu vlastního datového typu je možné napsat vlastní formatter. [37]

3.3.9 Testovatelnost

Framework je připraven na unit testy (jednotkové testy). Podporuje funkce, které zjednodušují práci při vytváření integračních testů. [37]

3.3.10 Razor view engine

Jak již bylo zmíněno, k vykreslování obsahu je využíván Razor engine. Využívá značkovacího jazyka Razor pro definování views za použití C# kódu. Razor je využíván k vygenerování dynamického obsahu. Jednoduše jím lze spojit server-side kód s client-side kódem. Lze definovat *layouts* (společná šablona), *partial views* (částečný view, který je možné použít ve více views) nebo nahraditelné sekce. [37]

```
<ul>
    @for (int i = 0; i < 5; i++) {
        <li>List item @i</li>
    }
</ul>
```

Zdrojový kód 3: Příklad značkovacího jazyku Razor (zdroj: [37])

3.3.11 Tag helpers

Tag helpers je skupina tzv. pomocných tagů. Jedná se o server-side kód, který generuje HTML kód v Razor souborech. Existuje několik předdefinovaných tag helperů, které zjednodušují práci při běžných úkonech – např. vytváření formuláře, odkazů. [37]

```
<p>
    Thank you for confirming your email.
    Please <a asp-controller="Account" asp-action="Login">Click here to Log in</a>.
</p>
```

Zdrojový kód 4: Příklad tag helperu (zdroj: [37])

Ve zdrojovém kódu č. 4 je možné vidět tag *<a>* obsahující vlastní atribut *asp-controller* a *asp-action*. Výstupem po vygenerování je atribut *href* s odkazem na Controller *Account* s akcí *Login*. [37]

3.4 Blazor

Blazor je dle [39] framework pro vytváření interaktivních webových UI na platformě .NET. Využívá tedy programovací jazyk C# místo tradičního JavaScriptu. Jedná se o alternativu známých izomorfních frameworků typu Angular, React a Vue.js. Umožňuje sdílet logiku mezi client-side a server-side napsanou v .NET. Blazor podporuje velké spektrum webových prohlížečů, včetně mobilních. Několik výhod Blazor frameworku dle [39]:

- Psaní kódu v C# místo JavaScriptu.
- Využití ekosystému .NET s existujícími .NET knihovnami.
- Sdílení aplikační logiky s client-side.
- Rychlost .NET a bezpečnost.
- Produktivní vývoj pomocí Visual Studio IDE.

Blazor byl původně experiment zaměstnance Microsoftu Steva Sandersona. Cílem bylo zjistit, zda je možné využít programovací jazyk C# pro vývoj interaktivního client-side UI. [39]

Framework lze využít jakožto server-side verzi nebo client-side. Server-side verze byla vydána společně s ASP.NET Core 3 v 23. září 2019. Client-side neboli WebAssembly verze byla vydána až v květnu 2020. [36] [40]

3.4.1 Komponenty

Blazor aplikace jsou založené na tzv. komponentách. Komponenta je prvek UI jako např. stránka, dialog nebo formulář. Komponenty jsou třídy překládané do .NET assembly s vlastnostmi dle [40]:

- Flexibilní UI.
- Zpracovává uživatelské akce a události.
- Lze dědit a znovu použít.
- Mohou být sdílená pomocí Razor knihovny a distribuována NuGet balíčky.

Komponenty jsou ve většině případů psané formou značkovacího jazyka Razor. Příklad dialogové komponenty je zobrazen ve zdrojovém kódu č. 5. [40]

```
<div>
    <h1>@Title</h1>

    @ChildContent

    <button @onclick="OnYes">Yes!</button>
</div>

@code {
    [Parameter]
    public string Title { get; set; }

    [Parameter]
    public RenderFragment ChildContent { get; set; }

    private void OnYes()
    {
        Console.WriteLine("Write to the console in C#! 'Yes' button was selected.");
    }
}
```

Zdrojový kód 5: Dialogová komponenta (zdroj: [40])

Vlastnosti *Title* a *ChildContent* obsahují atribut *Parameter*, který slouží jako atribut HTML značky. *RenderFragment* ukládá obsah značky do vlastnosti zobrazené na zdrojovém kódu č. 6. Pro uložení obsahu musí nést vlastnost specifický název *ChildContent*. Ve zdrojovém kódu č. 6 je možné vidět použití dialogové komponenty. [40]

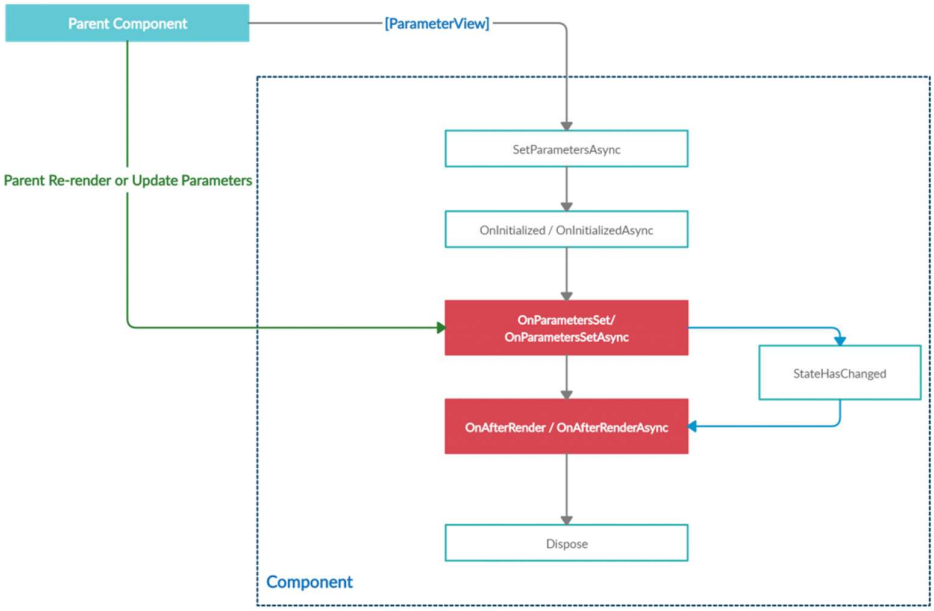
```
@page "/"

<h1>Hello, world!</h1>

Welcome to your new app.

<Dialog Title="Blazor">
    Do you want to <i>learn more</i> about Blazor?
</Dialog>
```

Zdrojový kód 6: Použití dialogové komponenty (zdroj: [40])

Životní cyklus komponenty

Blazor poskytuje synchronní a asynchronní verze metod životního cyklu popsaného na obrázku č. 9. Následující list metod dle [41] je volán při inicializaci komponenty a při renderování komponenty:

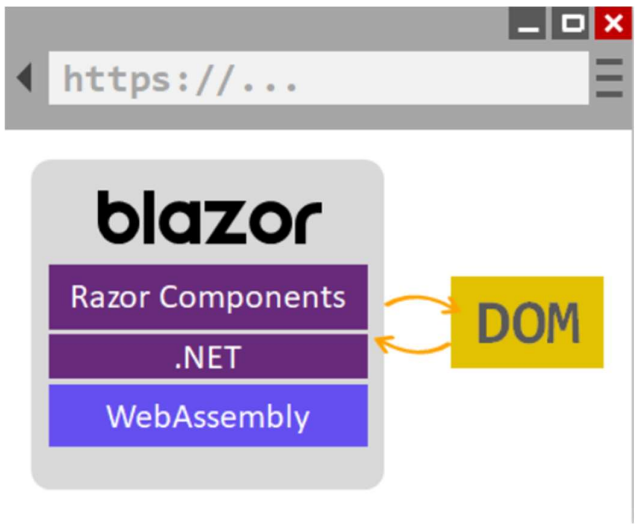
- SetParametersAsync
  - První metoda, která je zavolána po vytvoření komponenty. Přiřadí do vlastností hodnoty označené atributem *Parameter*, případně *CascadingParameter*.
- OnInitialized(Async)
  - Metoda, která je volána po inicializaci komponenty. Vhodné pro získání dat pomocí služby.
- OnParametersSet(Async)
  - Metoda je volána pokaždé, když získá nové hodnoty parametrů od rodičovské komponenty.
- OnAfterRender(Async)
  - Metoda je volána po dokončení renderu. Metoda obsahuje parametr *firstRender*, kdy je možné zaznamenat první vykreslení.
- ShouldRender
  - Metoda vrací hodnotu typu *bool*. Je volána před vykreslením a rozhoduje o tom, zda má být komponenta překreslena.
- StateHasChanged
  - Speciální metoda, která nepatří mezi metody životního cyklu. Její účel slouží pro signalizaci frameworku, aby vykreslil znovu komponentu.



Obrázek 9: Životní cyklus komponenty (zdroj: [41])

3.4.2 Client-side (WebAssembly)

Klientská verze Blazor frameworku funguje na technologii WebAssembly uvnitř webového prohlížeče. Aplikace i potřebné soubory pro spuštění jsou stáhnuty do webového prohlížeče při načtení stránky. Aplikace je spuštěna přímo ve webovém prohlížeči na UI vlákně. Aktualizace UI včetně zpracování událostí probíhá ve stejném procesu. Diagram fungování klientské verze je možné vidět na obrázku č. 10. [42]



Obrázek 10: Blazor WebAssembly (zdroj: [40])

WebAssembly verze nabízí dle [42] několik výhod:

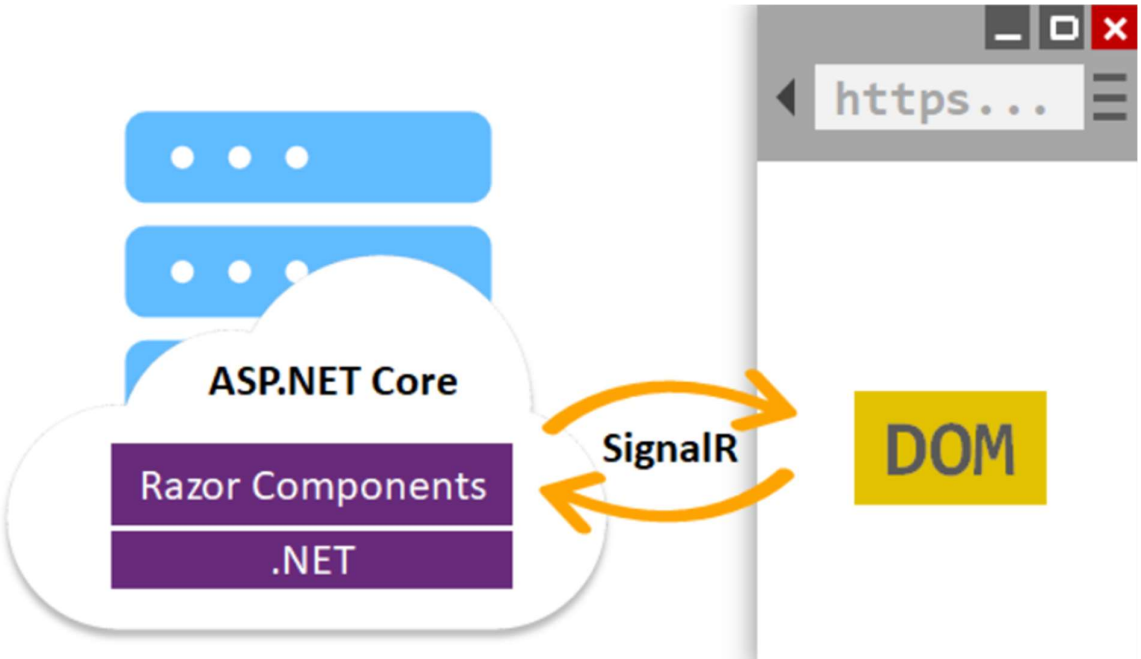
- Aplikace nevyžaduje přítomnost server-side. Funguje nezávisle po stažení do webového prohlížeče.
- Využívá zdroje klienta a jsou plně dostupné možnosti jeho systému.
- Práce je vykonávána mimo server.
- Pro deploy lze využít například CDN služby.

WebAssembly má dle [42] i své nevýhody:

- Aplikace je omezena na schopnosti klientského stroje.
- Klient musí podporovat WebAssembly.
- Aplikace má větší velikost a musí se načíst.
- Omezenost .NET runtime a menší podpora nástrojů.
- V současné době omezený debugging.

3.4.3 Server-side (SignalR)

Serverová verze Blazor frameworku funguje na bázi, kdy je veškerá logika zpracována uvnitř ASP.NET Core aplikace na serveru. Aktualizace UI, zpracování událostí a volání JavaScriptu je prováděno pomocí tenkého klienta ve webovém prohlížeči komunikujícího prostřednictvím SignalR konexe. Diagram fungování serverové verze je možné vidět na obrázku č. 11. [42]



Obrázek 11: Blazor SignalR (zdroj: [40])

Server-side verze dle [42] nabízí několik výhod:

- Velikost aplikace pro klienta je značně menší vzhledem k WebAssembly. Je stažený jen tenký klient. Aplikace je načtena mnohem rychleji.
- Aplikace využívá zdroje serveru a jeho dostupné možnosti systému.
- Plná podpora .NET nástrojů jako např. debuggování.
- Vysoká podpora malých klientů. Prohlížeče nemusí podporovat WebAssembly.
- Zdrojový kód Blazor aplikace není odeslán do klienta.

Nevýhody server-side verze dle [42]:

- Větší latence. Každá uživatelská interakce vyžaduje požadavek prostřednictvím sítě.
- Žádná offline podpora. Při ztrátě připojení dojde k zastavení aplikace.
- Při větším počtu uživatelů může dojít k využití veškerých zdrojů serveru.
- Je vyžadována přítomnost ASP.NET Core serveru.

3.4.4 Routing

Server-side verze routingu je integrovaná v sadě ASP.NET Core Endpoint Routing. Aplikace se za pomocí funkce *MapBlazorHub* nakonfiguruje pro přijetí příchozích požadavků od interaktivních komponent. Volání funkce probíhá v konfigurační sekci *Startup.Configure*. Mezi nejběžnější konfiguraci patří přesměrovat veškeré požadavky do Razor stránky, která se chová jako hostitel pro server-side část aplikace. Tato stránka má většinou název *_Host.cshtml*. Stejně jako v předešlém kroku se aplikuje nastavení pro mapování tzv. fallback route. Pokud žádné URL neodpovídá, je zvolena fallback route pro zpracování požadavku prostřednictvím *MapFallbackToPage(„/_Host“)*. [43]

Routing pro Blazor obecně obstarává komponenta *Router* uvnitř souboru *App.razor*. Při kompilaci každé komponenty obsahující klíčové slovo *@page* dochází k přidání atributu *RouteAttribute*. *RouteView* komponenta uvnitř *Router* komponenty obstarává část, kdy jsou předávána data z *Routeru* do cílové komponenty, dále je možné zvolit výchozí layout pomocí atributu *DefaultLayout*. Příklad routingu je možné vidět ve zdrojovém kódu č. 7. [43]

```
<Router AppAssembly="typeof(Startup).Assembly">
    <Found Context="routeData">
        <RouteView RouteData="@routeData" DefaultLayout="@typeof(MainLayout)" />
    </Found>
    <NotFound>
        <p>Sorry, there's nothing at this address.</p>
    </NotFound>
</Router>
```

Zdrojový kód 7: Blazor routing (zdroj: [43])

Routing uvnitř komponenty může obsahovat parametry, které se dosadí do příslušných *public* vlastností s atributem *Parameter* s voláním metody *SetParametersAsync*. [43]

3.4.5 JavaScript Interop

Blazor aplikace využívá pro volání JavaScript kódu z .NET *IJSRuntime* abstrakci. Abstraktní třída se pomocí DI zaregistruje do komponenty. Metoda *InvokeAsync* v prvním parametru přebírá identifikátor volané JavaScript funkce, další parametry jsou argumenty pro volanou JavaScript funkci. [44]

Je nutno poznamenat, že server-side verze může volat JavaScriptové funkce až po navázání připojení, tedy nelze využít v pre-rendering módu JavaScript. Pro detekování využití JavaScriptu se využívá metoda *OnAfterRender*, kdy je možné poprvé volat JavaScript. [44]

```
@inject IJSRuntime JSRuntime

<input @ref="username" />
<button @onclick="SetFocus">Set focus on username</button>

@code {
    private ElementReference username;

    public async Task SetFocus()
    {
        await JSRuntime.InvokeVoidAsync(
            "exampleJsFunctions.focusElement", username);
    }
}
```

Zdrojový kód 8: Příklad volání JavaScript funkce (zdroj: [44])

3.4.6 Event handling

Komponenty poskytují funkce pro zpracování událostí. Nejběžnější událost je při kliknutí na prvek – zápis probíhá ve formě *@on{EVENT}* (pro klik tedy *@onclick*). Příklad zpracování události při kliknutí na tlačítko je možné vidět ve zdrojovém kódu č. 9. [45]

```
<button class="btn btn-primary" @onclick="UpdateHeading">
    Update heading
</button>

@code {
    private void UpdateHeading(MouseEventArgs e)
    {
        ...
    }
}
```

Zdrojový kód 9: Příklad event-handlingu (zdroj: [45])

3.4.7 Data binding

Stejně jako pro zpracování událostí poskytují komponenty funkce pro zpracování data bindingu (datová vazba). Data binding funkce automaticky odkazuje na vlastnost, upravuje nebo vypisuje data přímo z dané vlastnosti. Převážně se využívá pro HTML značku <*input*>. Příklad je možné vidět ve zdrojovém kódu č. 10. [46]

```
<input @bind="CurrentValue" @bind:event="oninput" />

@code {
    private string CurrentValue { get; set; }
}
```

Zdrojový kód 10: Příklad data bindingu (zdroj: [46])

3.4.8 Layouts

Podobně jako ASP.NET Core MVC podporuje Blazor funkci layouts (rozložení). Technicky je layout další komponentou. Liší se pouze v části, kdy dědí (syntaxe *@inherits*) od komponenty *LayoutComponentBase* případně jiného komponenty dědící ze zmíněné komponenty. Výchozí layout je nazýván *MainLayout,* nachází se ve složce *Shared*. Jakákoliv komponenta může využít specifický layout pomocí klíčového slova *@layout nazev*. [47]

3.5 Obdobná řešení

3.5.1 Srovnání Symfony, ASP.NET MVC a Node.js

Xiaoli Mao z Univerzity zemědělství a aplikované vědy státu severní Dakoty napsal vědeckou studii *COMPARISON BETWEEN SYMFONY, ASP.NET MVC, AND NODE.JS EXPRESS FOR WEB DEVELOPMENT* [48]. Ve své studii se zabývá porovnáním frameworku Symfony pro programovací jazyk PHP, ASP.NET MVC a Node.js.

Ve své studii vytvořil webovou stránku nemocnice pro správu schůzek s pacienty. Pro vytvoření webové stránky se věnuje ve své práci výběru vhodné technologie na základě aspektů:

- zkušenosti s vývojem,
- výkon,
- další srovnání.

Zkušenosti s vývojem

Podle slov Xiaoli Mao je pro vývojáře PHP nejjednodušším jazykem. Byť ASP.NET MVC (pozn. autora: nikoliv ASP.NET Core) poskytuje funkce, které jsou velmi pohodlné pro vývoj. Node.js jakožto nejnovější technologie ze všech tří má značně výhod oproti soupeřům, jako je:

- rychlost,
- lehkost (lightweight),
- silné výpočetní schopnosti,
- a další.

Srovnání výkonu

Pro srovnání výkonu využil Xiaoli Mao benchmark test ve třech scénářích, neboť myslel na to, že každý web může mít jiné požadavky. Scénáře jsou:

1. Odeslání textového řetězce na front-end – porovnání I/O operací aplikace.
2. Výpočet Fibonacciho posloupnosti – porovnání výpočtového výkonu frameworku.
3. Výběr položek z databáze – porovnání výkonnosti operací s databází.

Pro porovnání zvolil Xiaoli Mao nástroj AppachBench, který je součástí webového serveru Apache. Autor poznamenává, že před jakýmkoliv testem byl proveden restart počítače a bylo spuštěno pouze vývojové prostředí a prohlížeč Google Chrome.

Výsledky testů byly následující:

1. Odeslání textového řetězce na front-end
    a. ASP.NET byl až 100x rychlejší než Node.js a zároveň 1000x rychlejší než framework Symfony.
2. Výpočet Fibonnaciho posloupnosti
    a. Scénář vypočítává F(10). Vzorec F(n) = F(n-1) + F(n-2).
    b. Při výpočtu mělo ASP.NET nejlepší výsledek a framework Symfony nejhorší.

3. Výběr položek z databáze
    a. Scénář vybírá 15 položek v tabulce o 50 řádcích a 5 sloupcích.
    b. Poslední test dopadl stejně jako v předchozích dvou případech. Symfony framework byl stále nejhorším, dále bylo zjištěno, že při výběru z databáze se výkon propadl z 2,5 dotazů za sekundu na 0,66 dotazů za sekundu.

Další srovnání

Autor jako další srovnání porovnával zabezpečení a podporu. Byť v studii chybí celkové srovnání pro zabezpečení, autor se k němu vrací při hodnocení a udává, že ASP.NET MVC a Symfony mají podobné bezpečnostní politiky. Sám autor doporučuje ASP.NET pro náročnější podniky, zatímco Symfony pro malé podniky, startupy a nezávislé weby. Node.js podporuje zabezpečení v rámci balíčků, na druhé straně se hodí lépe pro webové stránky zaměřené na výpočet či potřebu vysoké rychlosti a výkonu.

V rámci srovnání podpory autor uvádí, že všechny zmíněné technologie mají velkou komunitu vývojářů, kteří poskytnou podporu. Vítězem v podpoře je Node.js z následujících důvodů: open-source, ASP.NET MVC využívají zejména dedikovaní vývojáři. Symfony framework není nejpopulárnější mezi PHP frameworky, jedná se znovu o nejslabší článek v hodnocení.
[48]

3.5.2 Srovnání SPA a MPA

Shanin Tamjidi ze Švédského Blekingova institutu technologie napsal vědeckou studii *Comparison between SPA and MPA* [49]. Ve své práci srovnává SPA (single page application) s MPA (multiple page application). Studie je rozdělena na několik fází:

- Vytvoření dvou webových stránek.
- Nahrání na server.
- Registrace na Google Webmaster.
- Analýza změn.
- Přidání SEO technik.
- Čekání 72 hodin na změnu a následná analýza.

Práce se zabývá otázkami typu „Proč SEO nefunguje pro SPA optimálně?“, tedy je především zaměřená na porovnání SEO u obou technik.

Vytvoření dvou webových stránek

Pointa celé studie byla vytvoření dvou co nejvíce identických webových stránek. MPA využívá HTML a PHP, zatímco SPA využívá pouze JavaScriptu a HTML. Autor dále úmyslně nevyužil indexní (domovskou) stránku, aby zjistil, jak reaguje Google Bot v případě SPA.

Registrace na Google Webmaster

V tomto kroku autor zaregistroval dvě domény. Jedna s MPA webem, druhá s SPA webem. Google Webmaster slouží pro indexaci na vyhledávači Google. Změny mohou zabrat až 72 hodin.

První výzva pro Google Bota nastala v případě SPA a bylo sledováno, zda jej dokáže indexovat.

Přidání SEO technik

V případě MPA webových stránek odesílá název stránky na základě toho, kterou uživatel navštíví (element <*title*>). Dále bylo důležité využití meta tagů, zejména *description, keywords, viewport*.

Důležitá část webu, především pro indexaci je tzv. sitemapa (mapa stránek). Je ve formátu XML a má jasně definovanou strukturu. V případě statických stránek je možné využít nástroje jako https://www.xml-sitemaps.com k vygenerování sitemapy. Soubor je umístěn v kořenové složce webu.

SPA představovalo pro autora větší obtížnost měnit obsah meta tagů a zejména pak tagu <*title*>. Dalším krokem bylo u obou technik využít obecné SEO praktiky, tedy nadpisy v podobě <*h1*> a <*h2*>. Mezi další patří podpora HTTPS, mít u všech elementu <*img*> attribut *alt*, nevyužívat inline css, nemít poškozené odkazy a uvnitř prvního paragrafu (<*p*>) mít klíčové slovo „Angular universal cli“.

Čekání 72 hodin a následná analýza

Zde se opakuje cyklus kontroly na Google Webmaster nástroji a čekání, kdy se změna projeví. V některých případech je možné, že Google Bot nezobrazí správně web, jak by měl. Autorově studii trvalo čas, než se projevily výsledky ve vyhledávači.

Analýza a výsledky

V první analýze autor dosáhl lepších výsledků ve vykreslování u techniky MPA. Dále poznamenává, že se neobešly bez problémů. Po prvním týdnu nástroj Google Webmaster nezaznamenal žádné změny, byť autor již dosáhl vyhledávání na Google. Výsledky SERPu byly takřka identické. Po následujícím týdnu zaznamenal autor na doméně s MPA webem všechny stránky zaindexované. SPA web měl 1 z 5, byť byly všechny dohledatelné vyhledávačem.

Závěrem

Autor se pozastavuje nad výhodami a nevýhodami SPA vůči MPA s otázkou „Stálo to za to?“. Pokud je zvoleno SPA, je nutné, aby veškerá implementace byla provedena správně. Poté je možné získat kladné hodnocení na Google.

Výhody SPA:

- Plynulá a rychlá aplikace.
- DOM v prohlížeči.
- Dobré hodnocení SERPu.

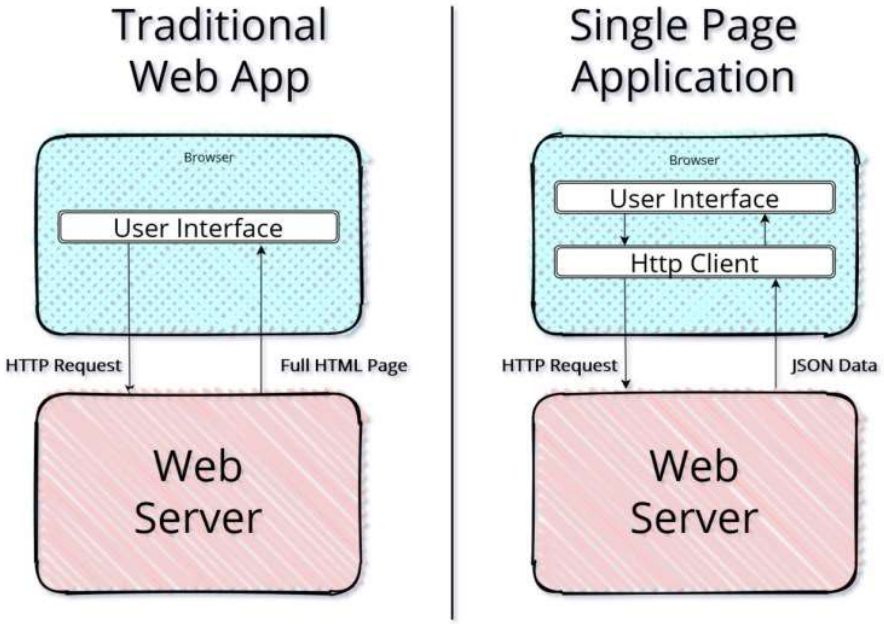
Nevýhody:

- Horší indexace na Google Webmaster.
- Mnoho SEO nástrojů na analýzu může selhat.
- Ostatní nevýhody, které se mohou objevit ve větších projektech.

[49]

3.5.3 Srovnání Blazor a tradiční webové aplikace

Christian Findlay popisuje ve článku *Blazor Vs. Traditional Web Apps* [50] na svém blogu rozdíly mezi Blazor technologií a tradičními webovými aplikacemi založenými na ASP.NET Web Forms nebo ASP.NET Core MVC. V článku shrnuje rozdíly tradiční webové aplikace a SPA, které vysvětluje pomocí obrázku č. 12. Tradiční webová aplikace odesílá celý HTML kód, zatímco SPA pouze část a zbytek zajišťuje HTTP klient v aplikaci.



Obrázek 12: Tradiční webové aplikace a Single page aplikace (zdroj: [50])

Christian Findlay dále uvádí, že SPA je vhodné jak pro interní, tak externí zákazníky. Christian uvádí, že v případě následujících bodů je vhodné využít SPA:

- Uživatel očekává moderní interaktivitu.
- Cílová skupina má moderní prohlížeč.
- Aplikace bude přenášet často data.
- Vývojový tým je znalý v oblasti Blazor technologie nebo podobných.
- Web API je first-class object.

Případy, kdy je vhodnější využít tradiční aplikace:

- Podpora starších prohlížečů.
- Podpora pomalých zařízení, kde skripty mohou zpomalovat zařízení.
- Aplikace má statický obsah nebo jednoduchý formulář.
- Vývojový tým nemá zkušenosti s SPA.
- Externí aplikace nevyužívají Web API tradiční webové aplikace.

Nevýhody Blazor technologie:

- Blazor WebAssembly nedosáhl celého výkonového potenciálu kvůli chybějící Ahead of Time kompilaci, která bude součástí budoucího .NET frameworku.
- Interakce je omezená na možnosti prohlížeče.
- Prvotní načtení aplikace je pomalé, protože se musí stáhnout celý .NET runtime.
- Debuggování client-side je v současné době omezeno.

Závěrem:

Uživatel očekává webové aplikace typu SPA od moderního prohlížeče. Tradiční webové aplikace jsou vhodné v případě scénáře, kdy je vyžadována stará infrastruktura, případně jednoduchost. Využitím Blazor technologie může znamenat pro vývojový tým nepotřebnost znalosti JavaScriptu, případně TypeScriptu.

[50]

3.6 Shrnutí

Webové technologie prošly za uplynulé dvě desetiletí značnou evolucí. Značkovací jazyk HTML 5 nahradil staré technologie, které vyžadovaly externí doplňky jako např.: Flash, Microsoft Silverlight nebo ActiveX. ASP.NET Core MVC je již v současné době zaběhnutá server-side technologie, která se používá v praxi a nahradila původní ASP.NET, který funguje pouze na platformě Windows.

Blazor framework je možné využívat pro produkční aplikace a lze konstatovat, že konkuruje nejběžnějším JavaScriptovým frameworkům Angular, React, případně Vue.js. Framework nabízí dvě varianty: client-side nebo server-side rendering. Nově vydaná technologie Blazor je s plnou podporou pro klienta od poloviny května 2020. Je nutné poznamenat, že Blazor je nadále rozvíjen na základě požadavků komunity a dočká se dalších rozšíření a optimalizací v připravované verzi .NET 5 (listopad 2020).

V současné době existuje několik nástrojů pro porovnání a optimalizaci webových aplikací. Mezi nejznámější patří Google PageSpeed Insight a GTmetrix. Některé nástroje jako Google PageSpeed Insight nepodporují testování s aktivní cache.

Závěr teoretické části je zakončen popisem obdobných prací a jejich výsledkům. První obdobná práce autora Xiaoli Mao srovnává frameworky pro různé programovací jazyky. Došel k závěru, že ASP.NET je nejrychlejší ze všech testovaných (odeslání, výpočet, výběr položek z databáze). Druhý autor Shanin Tamjidi se zaměřil na porovnání MPA a SPA za použití v prvním případě pouze HTML a PHP. V druhém případě pouze HTML a JavaScript s použitím frameworku AngularJs. Práce se zaměřila na tematiku indexace ve vyhledávacím stroji Google, která ve výsledku skončila negativně pro SPA, zejména při indexaci pomocí nástroje Google Webmaster. Třetí práce od Christiana Findlaye shrnuje rozdíly tradičních webových aplikací a SPA (Blazor). Zaměřuje se na výhody obou typů a v jakém případě je využít.

Na základě vědeckých studiích Xiaoli Mao a Shanin Tamjidiho byla zvolena kritéria pro porovnání webových aplikací v praktické části diplomové práce. Dále bylo v praktické části řešeno zachování již indexovaných stránek z provozu ASP.NET Core MVC při přechodu na framework Blazor, které vědecká studie Shanina Tamjidiho neřešila. Praktická část dále potvrzuje nevýhody frameworku Blazor podle autora Christiana Findlaye.

4 Praktická část

Praktická část diplomové práce je zaměřena na srovnání ASP.NET Core s použitím návrhového vzoru MVC vůči Blazor frameworku za použití metriky rychlosti načítání webu, velikosti při načtení webu, velikosti při navigaci webem, testování nástrojem Google PageSpeed Insight a srovnáním velikosti zdrojového kódu.

Pro splnění hlavního cíle – porovnání frameworku Blazor a ASP.NET Core MVC v online zpravodajství byla prováděna implementace dvou různých webových aplikací primárně v programovacím jazyce C#. Webové aplikace byly koncipovány tak, aby je bylo možné navštívit ve většině dostupných webových prohlížečích a zároveň byly z hlediska robotů (Seznam, Google, Bing) možné indexovat se správnými daty.

Se zaměřením hlavního cíle v online zpravodajství byla implementace prováděna ve studentské organizaci iZUN.eu, která je projektem zapsaného spolku ČZU Media z.s. působící na České zemědělské univerzitě v Praze (ČZU). Zkratka iZUN představuje internetové zemědělské univerzitní noviny. Hlavní činností je tvorba obsahu na webových stránkách www.izun.eu. Cílem webové stránky je žurnalistika týkající se dění na ČZU. Webová aplikace založená na technologii ASP.NET Core MVC byla v provozu na doméně izun.eu od února 2018 do září 2020, kdy ji nahradila webová aplikace založená na frameworku Blazor.

Postup praktické části je rozdělen na dvě části:

- Implementace
- Realizace experimentu porovnání

4.1 Prostředí pro implementaci

Pro implementaci webových aplikací byly potřeba následující položky:

- Vývojové IDE – Visual Studio
- .NET Core 3.1 SDK
- Webhosting s podporou .NET Core

Vývojové IDE Visual Studio podporuje kompilaci jazyku C#, které tvoří větší část webové aplikace u obou technologií. V současné době je k dispozici zdarma komunitní verze Visual Studio 2019 nebo v případě studenta ČZU lze získat *Enterprise* edici.

.NET Core 3.1 je možné získat pro platformy Windows, Linux a MacOS ze stránek společnosti Microsoft na adrese https://dotnet.microsoft.com/download/dotnet-core.

Posledním důležitým prvkem je webhosting podporující prostředí .NET Core. Pro účely diplomové práce byl využit stávající webhosting iZUN.eu s umístěním na Praze 6 – Vokovice. iZUN.eu využívá sdíleného hostingu – výkon serveru je sdílený s dalšími zákazníky.

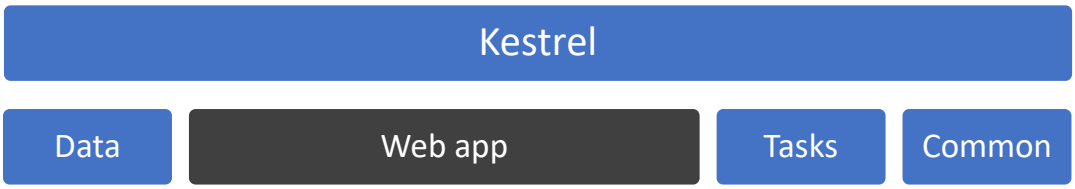
4.2 Implementace

Obě webové aplikace, jak založená na technologii ASP.NET Core MVC (dále jen MVC), tak na technologii Blazor, byly koncipovány tak, aby vycházely ze společného základu (následující kapitola 4.2.1) obsahující společné definice, stejný návrh databáze a pokud možno co nejvíce shodného kódu.

4.2.1 Společný základ

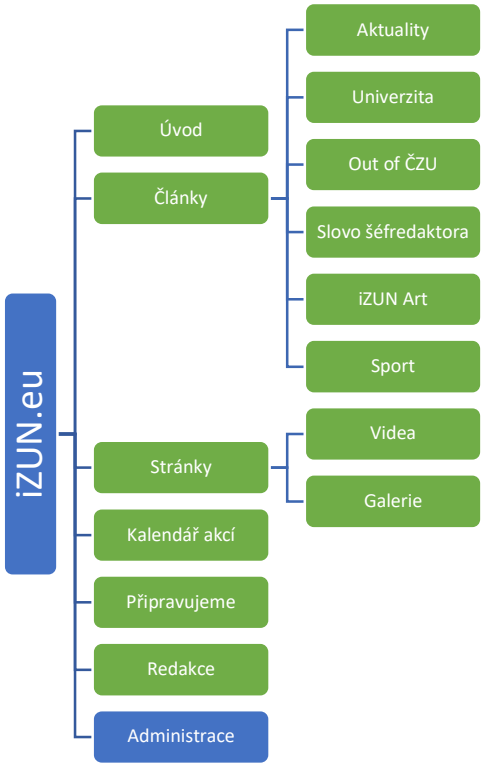
Webové aplikace byly koncipovány tak, aby sdílely základní infrastrukturu, tedy společný kód, databázi a UI. Diagram základní infrastruktury je možné vidět na obrázku č. 13. Implementace části pro MVC nebo server-side Blazor představuje šedivě vyznačený objekt *Web app*. Jednotlivé objekty diagramu jsou popsány níže:

- Kestrel
  - Cross-platform webový server pro ASP.NET Core.
- Data
  - Správa databázového kontextu aplikace – využití technologie Entity Framework Core.
- Web app
  - Implementace webové aplikace MVC (kapitola 4.2.2) nebo serverové části Blazor (kapitola 4.2.3).
- Tasks
  - Implementace úloh na pozadí, např.: získání počasí na ČZU.
- Common
  - Implementace pomocných tříd nebo rozšíření, např.: sitemapa.



Obrázek 13: Základní infrastruktura (zdroj: Vlastní)

Mezi podstatnou část společného základu patří hierarchický (logický) diagram iZUN.eu zobrazený na obrázku č. 14. Jednotlivé objekty diagramu představují části aplikace z pohledu návštěvníka webu. *Úvod* jako úvodní stránku, *Články* jako seznam článků dělené na *Aktuality, Univerzita, Out of ČZU, Slovo šéfredaktora, iZUN Art, Sport* a samotný článek jako objekt. *Stránky* jako statické podstránky, které se aktualizují maximálně ročně a nemají kategorii. *Kalendář akcí* s listem reálných událostí. *Připravujeme* s listem připravovaných článku pro vydání. *Redakce* jako statická stránka vypisující aktuální stav členů v redakci a *Administrace* jako komponenta spravující obsah webu.

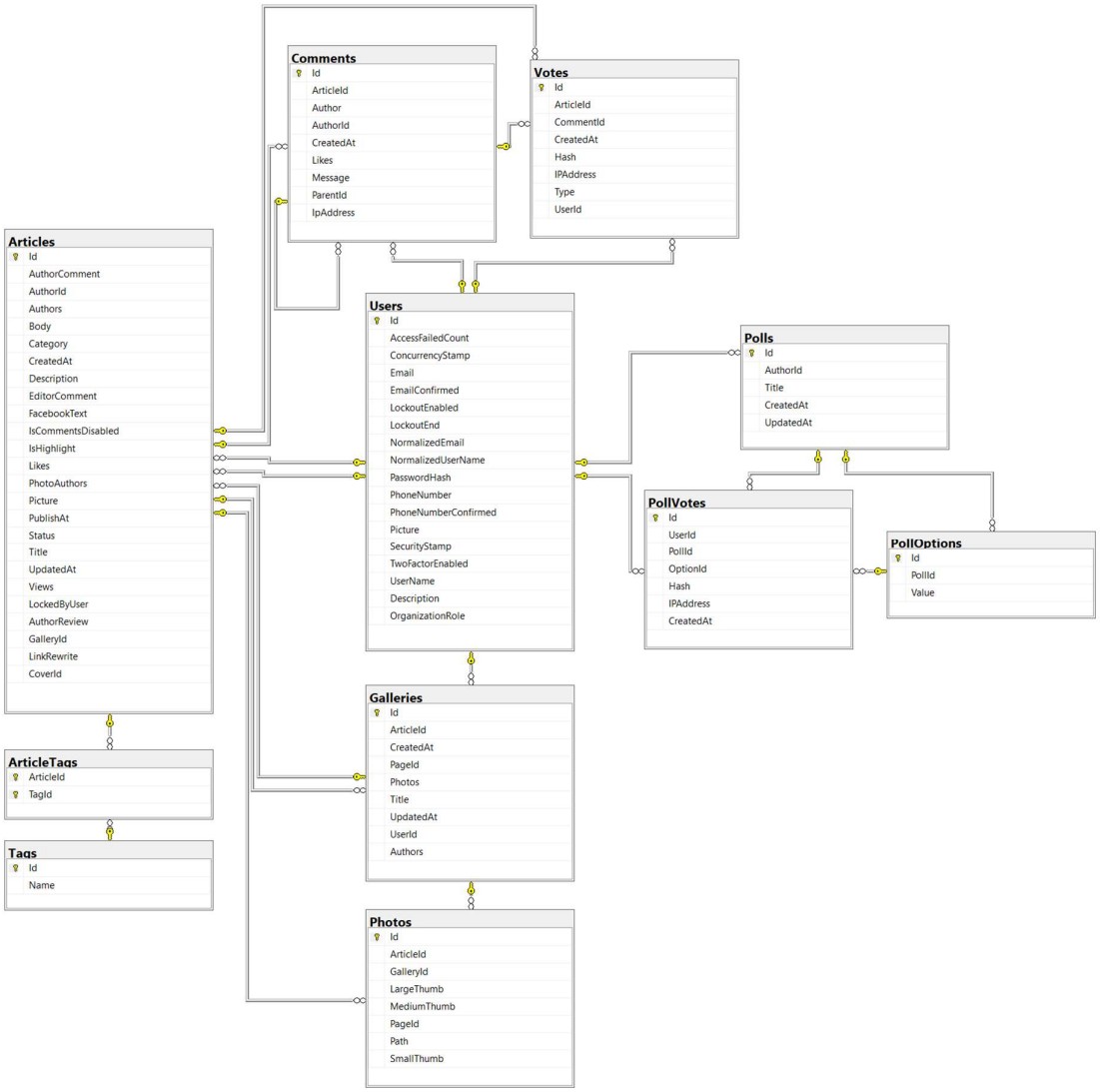


Obrázek 14: Hierarchický diagram iZUN.eu (zdroj: Vlastní)

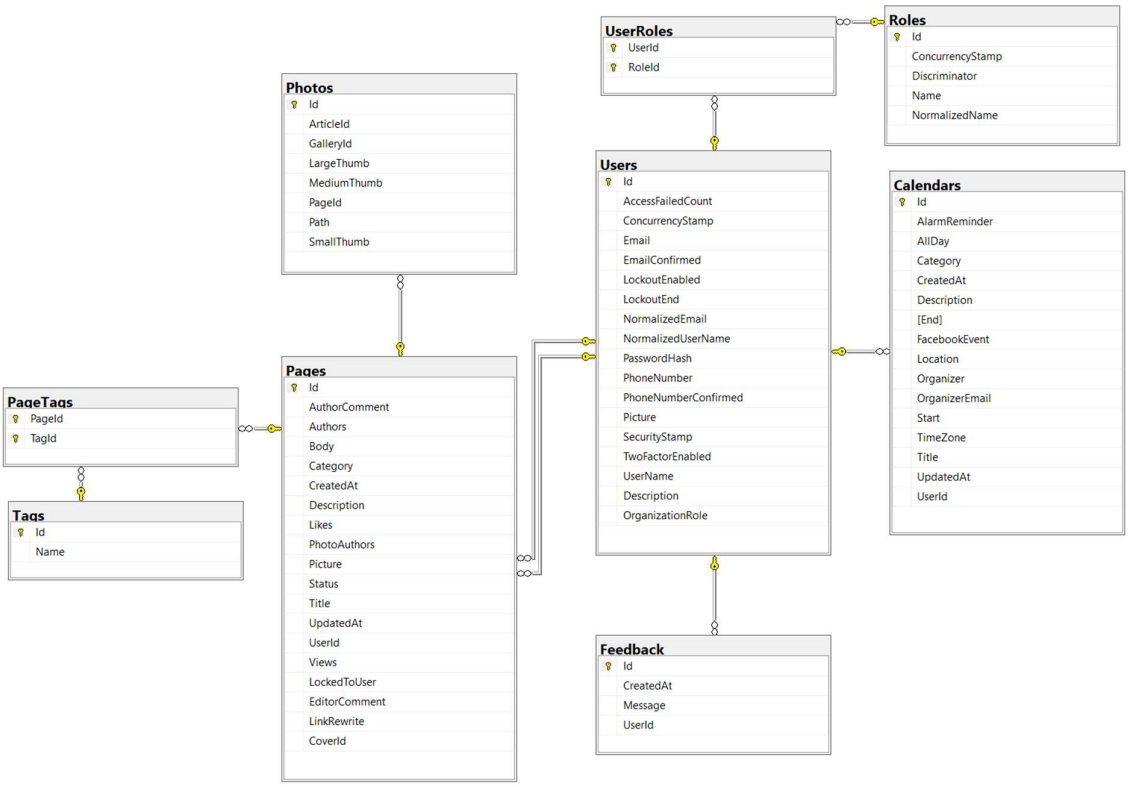
4.2.1.1 Datová infrastruktura

Pro ukládání dat z webové aplikace byla zvolena databáze SQL Server od Microsoftu, která je nativně podporována Entity Frameworkem sloužícím pro práci s databází. Databáze obsahuje přes 20 tabulek. Následující diagramy tvořící tabulky a jejich pole jsou psaná v angličtině podobně jako webové aplikace. Databázový diagram na obrázku č. 15 popisuje vztah zejména mezi článkem, hlasováním a galerií. Diagram na obrázku č. 16 popisuje vztah převážně mezi stránkou, uživatelem a kalendářem. Databázový diagram zakončuje obrázek č. 17, kde neprobíhá žádný vztah, ale jde o samostatné tabulky pro newsletter, nastavení aplikace nebo zobrazující se reklamu na webu.

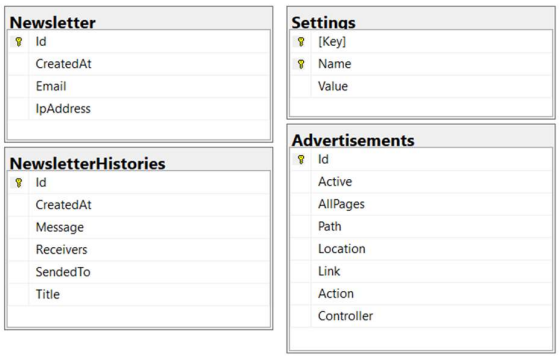
Při pohledu na jednotlivé diagramy je možné jednoduše odvodit vztahy a za pomocí vhodného pojmenování polí lze jednoduše odvodit data uložená v daném poli. V dalších částech implementace se pracuje s jednotlivými tabulkami.



Obrázek 15: Databázový diagram článků, hlasování a galerie (zdroj: Vlastní)



Obrázek 16: Databázový diagram stránek, uživatelů a kalendáře (zdroj: Vlastní)



Obrázek 17: Databázový diagram newsletteru, nastavení a reklamy (zdroj: Vlastní)

Database Context

Pro implementaci datové infrastruktury byla vytvořena třída *DatabaseContext* dědící z třídy *IdentityDbContext*, jenž je součástí frameworku ASP.NET Core. Část třídy je možné vidět ve zdrojovém kódu č. 11 – *Databázový kontext*.

```
public class DatabaseContext : IdentityDbContext<ApplicationUser, ApplicationRole, Guid>
{
    public DbSet<Advertisement> Advertisements { get; set; }
    public DbSet<Article> Articles { get; set; }
    public DbSet<ArticleTag> ArticleTags { get; set; }
    public DbSet<Calendar> Calendars { get; set; }
    public DbSet<Comment> Comments { get; set; }
    public DbSet<EventLog> EventLogs { get; set; }
    public DbSet<Feedback> Feedback { get; set; }
    public DbSet<Gallery> Galleries { get; set; }
    public DbSet<Newsletter> Newsletter { get; set; }
    public DbSet<NewsletterHistory> NewsletterHistories { get; set; }
    public DbSet<Page> Pages { get; set; }
    public DbSet<PageTag> PageTags { get; set; }
    public DbSet<Photo> Photos { get; set; }
    public DbSet<Setting> Settings { get; set; }
    public DbSet<Vote> Votes { get; set; }
    public DbSet<Tag> Tags { get; set; }
```

Zdrojový kód 11: Databázový kontext (zdroj: Vlastní)

Jednotlivé vlastnosti typu *DbSet<T>* používají jako položku modelovou třídu z obrázků s databázovými diagramy. Tedy např.: *DbSet<Advertisement>* mapuje model *Advertisement* do tabulky *Advertisements*.

Třída *DatabaseContext* implementuje přetížení metody *OnModelCreating* ve zdrojovém kódu č. 12 a za pomocí Fluent API dochází k upravení vztahů mezi tabulkami. Např.: model *Setting* má unikátní primární klíč skládající se z proměnné *Key* a proměnné *Name*.

```
builder.Entity<Setting>().HasKey(x => new { x.Key, x.Name });
builder.Entity<ArticleTag>().HasKey(x => new { x.ArticleId, x.TagId });
builder.Entity<PageTag>().HasKey(x => new { x.PageId, x.TagId });
```

Zdrojový kód 12: DatabaseContext Fluent API (zdroj: Vlastní)

Databázový kontext je dále zaregistrován ve třídě *Startup* v metodě *ConfigureServices* za pomocí parametru *services* zavoláním funkce *AddDbContext<DatabaseContext>*. Po registraci je možné z jakéhokoliv controlleru či služby zaregistrované prostřednictvím *services* za pomocí techniky dependency injection získat objekt typu *DatabaseContext*.

Uživatelské role

Pro rozlišení práv uživatele jsou využívané uživatelské role (tabulka *Roles*):

- User
    - Registrovaný uživatel bez přístupu do administrace.
- Redactor
    - Člen redakce s omezenými právy v administraci.
- Editor
    - Člen redakce s většinou práv v administraci.
- Administrator
    - Člen redakce se všemi právy v administraci.

4.2.1.2 Tasks

Tasks (úlohy) jsou důležitou součástí infrastruktury, které provádí danou úlohu v pozadí. Ve webové aplikaci iZUN.eu jsou primárně dvě úlohy:

Počasí:

Pro vyhnutí se opakovanému dotazu na http://meteostanice.agrobiologie.cz/ při každém požadavku je zvolena metoda úlohy na pozadí. Po spuštění aplikace je odeslán požadavek na zmíněnou adresu pro získání počasí, a poté každých 15 minut pro aktualizaci. Tímto dochází k úspoře ohromného množství požadavků, které by zpomalovaly zpracování koncového požadavku. Data o počasí jsou uložena v paměti aplikace.

Newsletter:

Při velkém množství e-mailů odeslané v jeden čas z jednoho serveru by bylo možné vystavit IP adresu serveru blacklistu serveru příjemce, pokud by větší část příjemců sdílela stejný e-mailový server. Aby nedocházelo k takovému kroku, je vytvořena úloha na pozadí, která rozesílá každých 6 hodin maximálně 100 e-mailů. Pro označení, zda byl příjemci již odeslán e-mail, je využíváno pole *SendedTo* v tabulce *NewsletterHistories*. Dále obsahuje pole *Receivers* – příjemci, *Title* – titulek newsletteru a *Message* – obsah newsletteru.

4.2.1.3 Common

Common (v překladu společný) namespace představuje část, kde jsou implementované funkce, případně rozšíření (C# extensions) využívané v několika různých částech infrastruktury (zejména controllery a views).

Pro MVC aplikaci se jedná o místo implementace vlastních ASP.NET Core tag helperů a akčních filtrů.

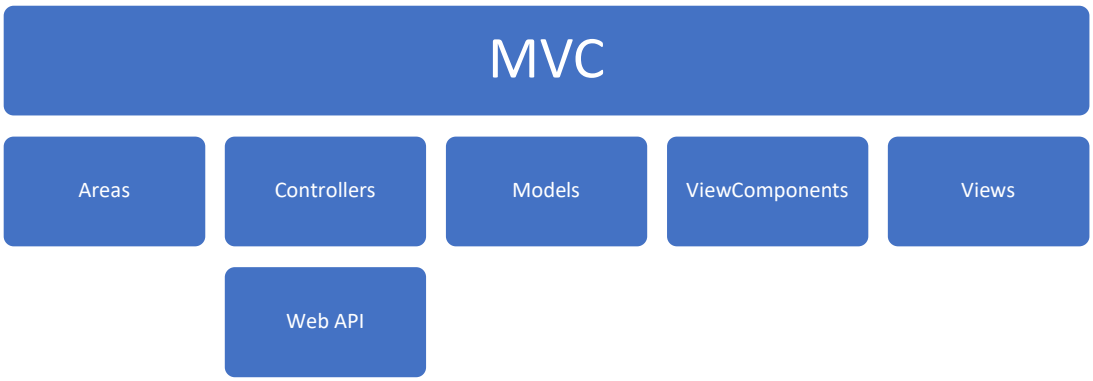
4.2.2 ASP.NET Core MVC

Pro vývoj MVC webové aplikace byl využit CSS framework Foundation dostupný z https://get.foundation/ a JavaScriptový framework VueJS dostupný z https://vuejs.org/.

Před samotným začátkem implementace bylo vhodné sestavit infrastrukturu webové aplikace. Diagram MVC infrastruktury je zobrazen na obrázku č. 18. Jednotlivé objekty diagramu jsou popsány níže:

- Areas
    - MVC architektura pro implementaci dané oblasti, např.: administrace.

- Controllers
    - Implementace business logiky aplikace.
    - Web API
        - Implementace business logiky Web API.
- Models
    - Implementace datových modelů pro předávání mezi *Controllers* a *Views.*
- ViewComponents
    - Implementace komponent pro vykreslování.
- Views
    - Implementace vykreslovací logiky.



Obrázek 18: Infrastruktura MVC aplikace (zdroj: Vlastní)

Layout

Téměř každá interakce uživatele vede k přesměrování např.: odkaz, odeslání formuláře. Při každém požadavku s HTML výstupem se používá sdílená kostra – tzv. layout.

Layout je součástí *Views*, jeho funkcí je zamezit duplicitě HTML značek v separátních souborech (typicky tag <html>, <head> a část <body>).

Zdrojový kód *_Layout.cshtml* obsahuje přes 800 řádků. Nejdůležitější části jsou:

- HTML kostra,
- meta tagy,
- linkování CSS souborů,
- linkování JavaScript souborů,
- menu,
- přihlášení / registrace,
- počasí,
- reklama,

- postranní panel,
- partneři,
- zápatí stránky.

Mezi důležité CSS soubory patří framework Foundation a dále vlastní stylovací soubor. Pro JavaScript soubory to jsou jQuery, VueJS a vlastní JavaScript soubor, ve kterém dochází k inicializaci VueJS aplikace sloužící především pro dynamickou interakci v postranním panelu a načítání asynchronně dat, které by zdržovaly požadavek.

Ve zdrojovém kódu č. 13 je možné vidět implementaci dynamického titulku stránky, který určuje dětský objekt (např.: Úvod, článek, statická stránka).

```
@if ((string)ViewData["Title"] == "")
{
<meta property="og:title" content="Dojíme pro Vás informace">
<meta property="og:type" content="website">

<title>iZUN.eu Univerzitní noviny ČZU | Dojíme pro Vás informace</title>
}
else
{
<meta property="og:title" content="@ViewData["Title"]">

<title>@ViewData["Title"] | iZUN.eu Univerzitní noviny ČZU</title>
}
```

Zdrojový kód 13: Dynamický titulek (zdroj: Vlastní)

V případě, že objekt *ViewData* s klíčem *Title* je prázdný – zobrazí se výchozí titulek. Pokud dětský objekt definoval hodnotu do daného objektu s klíčem, bude titulek stránky kombinace hodnoty a pevně stanoveného textu „ | iZUN.eu Univerzitní noviny ČZU“. Tímto způsobem je docíleno dynamického titulku stránky primárně pro roboty a sdílení přes sociální sítě.

Důležitou částí pro uživatele je postranní panel. Data pro postranní panel spravuje framework VueJS. Inicializace probíhá v souboru *app.js*. Zjednodušená verze inicializace se získáním dat počasí je možná vidět ve zdrojovém kódu č. 14.

Zdrojový kód definuje proměnnou *app* instancí objektu *Vue* s příslušnými parametry. Parametr *el* označuje, které HTML značce se má přivázat (bind), a ve které hledat změny. Parametr *mixins* slouží pro spojení metod třídy *Vue*, které se definují v dílčích objektech aplikace. Parametr *data* slouží jako pole proměnných, s kterými VueJS pracuje. Funkce *mounted* definuje, co se provede po inicializaci třídy. V tomto případě proběhne požadavek

typu *GET* na adresu *api/Common/Weather*. Po dokončení úspěšného požadavku nastává přiřazení hodnoty do proměnné *weather*.

```
var app = new Vue(
    {
        el: "#app-root",
        mixins: mixins,
        data:
        {
            weather: 0,
        },
        mounted: function()
        {
            axios.get("api/Common/Weather").then(function(response)
            {
                app.weather = response.data;
            }).catch(function(error)
            {
                console.log(error);
            });
        }
    });
```

Zdrojový kód 14: app.js - počasí (zdroj: Vlastní)

Výpis počasí v souboru *_Layout.cshtml* je možný vidět ve zdrojovém kódu č. 15. Zdrojový kód obsahuje atypický atribut *v-cloak*, který skryje veškerý obsah, dokud není inicializována třída *Vue* na některém rodičovském tagu. Atribut *v-cloak* tímto předchází zobrazení textu „{{ weather }}“ koncovému uživateli. Po úspěšné inicializaci třídy *Vue* dojde k nahrazení zmíněného textu hodnotou proměnné uvnitř dvojitých závorek, tedy proměnnou *weather*, která činí po inicializaci „0“. Po dokončení požadavku *GET* ve zdrojovém kódu č. 14 proběhne automatická aktualizace vypsané hodnoty na příslušnou hodnotu získanou z požadavku.

```
<div class="small-4 cell text-right show-for-large" id="temperature">
    <div class="text-center">
        <span class="temp" v-cloak>{{ weather }}</span>
        <span class="gray">Meteo ČZU</span>
    </div>

    <img src="~/images/celsium.png" alt="Stupně Celsia">
</div>
```

Zdrojový kód 15: Výpis počasí (zdroj: Vlastní)

View komponenta: Partners

View komponenta *Partners* je využívána layoutem *_Layout.cshtml*, který nemá svůj controller. Data o partnerech jsou uložená v databázi v tabulce *Settings*, které se načítají prostřednictvím třídy *DatabaseContext*, který lze získat technikou dependency injection

v controlleru, případně ve zvláštních případech i ve *Views* syntaxí *@inject*. Pro oddělení business logiky (načtení z databáze) byla zvolena metoda View Component, kterou lze využít jako vlastní HTML tag, přičemž při požadavku dochází k renderování na bázi virtuálního MVC. Při nevyužití metody View Component by se při každé akci musel duplikovat kód pro získání dat a následně je předat například pomocí *ViewData* objektu.

Business logika komponenty *Partners* je možná vidět ve zdrojovém kódu č. 16.

```
[ViewComponent(Name = "Partners")]
public class PartnersComponent : ViewComponent
{
    readonly DatabaseContext db;

    public PartnersComponent(DatabaseContext context)
    {
        db = context;
    }

    public async Task<IViewComponentResult> InvokeAsync()
    {
        return View(new PartnerSettingModel
        {
            Links = (await db.Settings.Where(x => x.Key == "Partners" && x.Name == "Links").SingleOrDefaultAsync())?.Value,
            Images = (await db.Settings.Where(x => x.Key == "Partners" && x.Name == "Images").SingleOrDefaultAsync())?.Value.Replace("\r\n", ""),
            Alt = (await db.Settings.Where(x => x.Key == "Partners" && x.Name == "Alts").SingleOrDefaultAsync())?.Value
        });
    }
}
```

Zdrojový kód 16: View Component - Partners (zdroj: Vlastní)

Třída *PartnersComponent* dědí z třídy *ViewComponent*. Je označena atributem *ViewComponent* s parametrem *Name*. Instruktor třídy obsahuje parametr *DatabaseContext*, který je získán technikou dependency injection. Metoda *InvokeAsync* provádí business logiku komponenty – získání dat z databáze a vrací je v modelu *PartnerSettingModel*.

View komponenty je možné vidět ve zdrojovém kódu č. 17, kde dochází k použití modelu *PartnerSettingsModel* a za pomocí dvou *for* cyklů je prováděn výpis partnerů. Komponentu lze poté zavolat z jakékoliv View části za pomocí tagu *<vc:partners></vc:partners>* nebo syntaxí *@await Component.InvokeAsync(„Partners“)*.

```
@model iZUN.Models.PartnerSettingModel

<div class="partners-container">
    <div class="orbit" role="region" aria-label="Partneři" data-orbit>
        <div class="orbit-wrapper">
            <ul class="orbit-container">
                @for (int i = 0; i < (int)Math.Floor((double)Model.ImagesList.Count / 5.0); i++)
                {
                <li class="orbit-slide @Html.Raw(i == 0 ? "is-active" : "")">
                    <figure class="orbit-figure">
                        @for (int j = i * 5; j < Math.Min((i + 1) * 5, Model.ImagesList.Count); j++)
                        {
                        <a href="@Html.Raw(Model.LinksList[j])"><img class="orbit-image" src="@Html.Raw(Model.ImagesList[j])" alt="@Html.Raw(Model.AltList[j])"></a>
                        }
                    </figure>
                </li>
                }
            </ul>
        </div>
    </div>
</div>
```

Zdrojový kód 17: View - Partners (zdroj: Vlastní)

Úvod

Implementace business logiky úvodní stránky byla provedena v tzv. výchozím controlleru *HomeController*. Funkce *async Task<IActionResult> Index* zpracovává poslední 4 položky z tabulky *Articles* a poslední 3 položky z tabulky *Comments*. Při výběru z tabulky *Articles* probíhá podmínka článku ve stavu publikace (pole *Status* v tabulce), a případně publikovat v (pole *PublishAt* v tabulce) musí být menší nebo rovno momentálnímu času a dnu. Funkce *Index* vrací akci *View* s modelem typu *ArticleIndexModel* obsahující jen nezbytně nutná data pro výpis.

V souboru *Home/Index.cshtml* poté probíhá výpis za pomocí Razor Engine a HTML značek. Soubor dále obsahuje rozšíření JavaScriptové třídy *Vue* o dynamické načítání dalších článků za pomocí tlačítka.

Články a stránky

Implementace business logiky článků a stránek byla velice obdobná, protože stránky jsou články bez funkcí např. hlasování a komentáře. Implementace stránky s výpisem článků a článkem je provedena v controlleru *ArticlesController* (v případě statických stránek v controlleru *PagesController*). Controller obsahuje několik metod:

- Index(category, tag)
    - Funkce vrací list článků s filtrováním podle vstupních parametrů *category* a *tag*.
    - URL vzor: „clanky/{category?}/{tag?}“.
- Details(category, title)
    - Funkce vrací článek z paměti (pokud článek není v paměti, je načten z tabulky *Articles* podle pole *LinkRewrite* – kombinace parametru *category* a *title* a uložen do paměti) na základě parametrů *category* a *title.*
    - Článku je provedena inkrementace pole *Views* o hodnotu 1 a uložena do databáze.
    - Článek automaticky doplňuje všechny obrázky o odkaz pro rozkliknutí.
    - Článek automaticky nahrazuje v obsahuje článku klíčové slovo „{NEJLEPSI_KOMENTAR}“ za HTML značky pro zobrazení nejoblíbenějšího komentáře.
    - Článek automaticky nahrazuje v obsahu článku klíčové slovo „{ANKETA:číslo ankety}“ za HTML značky pro zobrazení hlasovacího formuláře ankety s číslem v klíčovém slově.

    - Funkce předává prostřednictvím objektu *ViewData* tři podobné články na základě společných tzv. tagů – v databázi N:M (*Articles : ArticleTags*) relace.
    - URL vzor: „{category}/{title}“.
- Vote(id)
    - Funkce typu *POST* vrací odpověď typu *application/json* s odpovědí úspěchu nebo neúspěchu hlasování o oblíbenosti článku s ID podle parametru *id*.
    - Přičítá nebo odečítá hlas článku.
- VoteComment(id)
    - Funkce na stejném principu jako funkce *Vote*, kde probíhá odeslání hlasu o oblíbenosti komentáře.
- PostComment(id, comment)
    - Funkce typu *POST* vrací odpověď typu *application/json* s odpovědí úspěchu a nově vloženým komentářem nebo neúspěchu v podobě erroru. Vstupními argumenty jsou *id* článku a model *comment* typu *Comment*.
    - Funkce ověřuje, zda článek existuje, je publikovaný, nemá zakázané komentáře a případně zda existuje kořenový komentář.
    - Při správném postupu probíhá zápis do tabulky *Comments*, je odeslán e-mail Team Leaderům redakce a autorovi článku.

Kalendář akcí

Implementace kalendáře akcí probíhala v controlleru *CalendarsController* s jedinou funkcí *Index* (URL „/kalendar-akci“), kde probíhá výběr 10 položek z tabuly *Calendars* a zároveň, kde je pole *Start* větší než dnešní datum a čas. Výběr probíhá od nejbližšího začínajícího data. Výstup je předáván pomocí listu s typem *CalendarModel*.

Připravujeme

Pro implementaci byl zvolen výchozí controller *HomeController*. Redakce iZUN plánuje vydávání článku za pomocí Google kalendáře, a tedy funkce *Pripravujeme* s URL „/pripravujeme“ odesílá požadavek na Google API, kde vybírá maximálně 10 položek řazené podle data. Na základě těchto položek je tvořen list s typem *ArticlePrepareModel* obsahující informace o autorovi článku, názvu článku, popisku článku a data předpokládaného vydání.

Redakce

Implementace stránky redakce byla provedena ve výchozím controlleru *HomeController* s URL „/redakce“. Stránka redakce vypisuje současný stav členů redakce a dělí se do několika sekcí na základě organizační role – pole *OrganizationUnit* v tabulce *Users*:

- Vedení
    - Předseda
    - Místopředseda
    - Šéfredaktor
    - Vedoucí fotografů
    - Vedoucí grafiků
    - Vedoucí organizátorů
    - Vedoucí videoprodukce
    - HR
    - Sociální sítě
    - Tajemník
- Fotografové
- Grafici
- Korektoři
- Organizátoři
- Redaktoři
- Videoprodukce
- Ostatní
    - Programátor
- Externisté

Administrace

Administrace byla implementována pomocí oblasti tzv. Area. Bylo vhodné z pohledu infrastruktury rozdělit aplikaci na části, kde administrace představuje velkou část. V administraci byly implementovány funkce s rozdělením podle uživatelské role (kapitola 4.2.1.1, sekce *Uživatelské role*):

- Přehled
    - Informuje člena redakce, jak dlouho nenapsal žádný článek.
    - Informuje člena redakce pomocí grafu, kolik napsal článků a v jaké kategorii.

    - Informuje člena redakce pomocí grafu, kolik vytvořil galerií.
    - Informuje člena redakce o jeho nejčtenějším článku s počtem zobrazení.
- Články
    - Umožňuje vytvářet, upravovat a smazat článek.
    - Záložka „Moje články“ zobrazuje list článků daného člena.
    - Záložka „Nevydané články“ zobrazuje list nevydaných článků.
    - Záložka „Všechny články“ zobrazuje list o všech článcích redakce s informacemi – Název, kategorie, Highlight, Počet zobrazení, Počet to se mi líbí, Autor, Stav a Datum vytvoření.
- Stránky
    - Dostupné pouze pro roli „Editor“ a výše.
    - Umožňuje vytvářet, upravovat a smazat články
    - Záložka „Všechny články“ zobrazuje list o všech článcích redakce s informacemi – Název, Počet zobrazení, Počet to se mi líbí, Autor a Stav.
- Ankety
    - Umožňuje vytvářet, upravovat a smazat anketu.
- Galerie
    - Umožňuje vytvářet, upravovat a smazat galerii.
    - Umožňuje nahrávat fotografie s automatickým přidáním vodoznaku.
    - Záložka „Moje galerie“ zobrazuje list galerií daného člena.
    - Záložka „Všechny galerie“ zobrazuje list o všech galeriích redakce s informacemi – Název, počet fotek a autor.
- Kalendář akcí
    - Umožňuje vytvářet, upravovat a smazat akci.
    - Záložka „Všechny akce“ zobrazuje list o všech akcích s informacemi – Název, kategorie a začátek akce.
- Uživatelé
    - Dostupné pouze pro roli „Administrator“.
    - Umožňuje spravovat údaje, organizační roli a systémovou roli uživatele.
- Newsletter
    - Dostupné pouze pro roli „Administrator“.
    - Vypisuje e-mail a datum vytvoření odběru novinek.
    - Umožňuje rozeslat newsletter.

- Feedback
    - Umožňuje vytvořit feedback členovi redakce o chybě webové aplikace.
    - Při vytvoření odesílá e-mail programátorovi.
- Feedbacky
    - Dostupné pouze pro roli „Administrator“.
    - Zobrazuje list všech feedbacků.

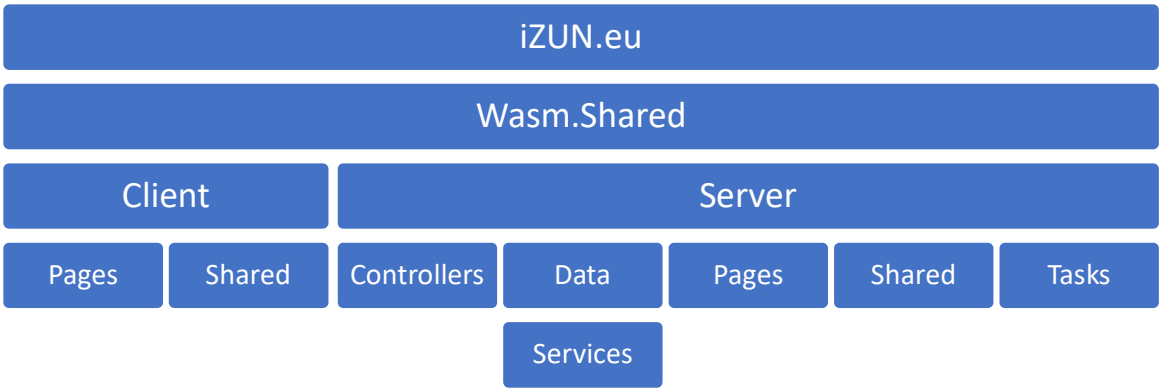
4.2.3 Blazor

Pro vývoj Blazor webové aplikace byl použit CSS framework Bootstrap dostupný z https://getbootstrap.com/.

Podobně jako u implementace MVC byl sestaven diagram infrastruktury webové aplikace. Rozdíl proti MVC aplikaci je jednoznačný a je zobrazen na obrázku č. 19. Protože je Blazor WebAssembly projekt rozdělen na tři části (client-side, server-side a sdílená knihovna), zatímco MVC jen na jeden (server-side), diagram obsahuje prvky základní infrastruktury. Jednotlivé objekty diagramu jsou popsány níže:

- Wasm.Shared
    - Knihovna obsahující sdílený kód pro klienta a zároveň server.
    - Obsahuje především datové modely pro předávání nezbytných dat při požadavku klient-server.
    - Obsahuje „Common“ část ze základní infrastruktury s definicí, např.: řadící atribut.
- Client
    - Implementace business logiky klientské aplikace.
    - Pages
        - Implementace jednotlivých komponent s URL kontextem.
    - Shared
        - Implementace sdílených komponent (bez URL kontextu).
        - Sdílené komponenty jsou využívané ve sdílených či jednotlivých komponentech.
- Server
    - Implementace business logiky serverové aplikace.
    - Controllers
        - Implementace Web API controllerů, které odpovídají na dotazy klienta.

- Data
  - Implementace správy databáze.
  - Services
    - Implementace služeb pro server-side rendering.
- Pages
  - Implementace jednotlivých komponent s URL kontextem – server-side prerendering.
- Shared
  - Implementace sdílených komponent jako u klienta pro server-side prerendering.
- Tasks
  - Implementace úloh na pozadí. Provádí stejnou funkci jako v MVC aplikaci.



Obrázek 19: Infrastruktura Blazor aplikace (zdroj: Vlastní)

4.2.3.1 Client-side

Klientská část aplikace je v samostatném projektu *iZUN.Wasm.Client*. Výstupem jsou soubory s příponou .dll, které jsou spuštěny v sandboxu webového prohlížeče. Aplikační logika byla převážně programovaná v jazyce C#, který byl posléze zkompilován do binární formy WebAssembly kódu – wasm.

MainLayout:

Blazor využívá soubory s příponou *.razor*, nikoliv *.cshtml*. Stejně tak jako MVC má i Blazor výchozí layout – nejběžněji označený jako *MainLayout*. Na rozdíl od MVC *MainLayout* neobsahuje celou kostru HTML, ale pouze obsah těla. To znamená, že neobsahuje ani tag *<body>*. Při inicializaci webové aplikace se načte klientská aplikace a naváže se na kořenový

element. Ve výchozím stavu se tento element nazývá *app*, lze jej upravit v třídě *Startup* klientského projektu.

Zdrojový kód *MainLayout.razor* je vzhledem k MVC tenký, ale na druhou stranu obsahuje logickou část, která se stará např.: o načtení počasí (v MVC aplikaci v separátním souboru app.js). Mezi nejdůležitější části patří:

- menu,
- přihlášení / odhlášení,
- počasí,
- kostra postranního panelu,
- zápatí stránky.

Na základě položek seznamu výše je možné vyvodit, že *MainLayout* spravuje daleko méně funkcí, které nejsou nemožné spravovat (obsah tagu <*head*>) nebo jsou rozložené na dílčí komponenty.

Podobně jako MVC implementuje postranní panel, *MainLayout* obsahuje kostru, která volá dílčí komponenty postranního panelu a je možné je vidět ve zdrojovém kódu č. 18. Komponenty <*FoodBar*> a další jsou definované v samostatném souboru podle názvu tagu s příponou *.razor* ve složce *Shared*.

```
<FoodBar></FoodBar>

<div id="triple-bar">
    <div class="row no-gutters">
        <div class="col-4 col">
            <a href="/stranky/otviracky" class="text-decoration-none">
                <img src="/images/clocks.png" alt="Hodiny" loading="lazy">
                <span>Otvíračky</span>
            </a>
        </div>

        <div class="col-4 col">
            <a href="/stranky/izun-pruvodce-po-czu-podmante-si-svou-skolu" class="text-decoration-none">
                <img src="/images/hand.png" alt="Ruka" loading="lazy">
                <span>Průvodce studenta</span>
            </a>
        </div>

        <div class="col-4 col">
            <a href="/videa" class="text-decoration-none">
                <img src="/images/director.png" alt="iZUN TV" loading="lazy">
                <span>iZUN TV</span>
            </a>
        </div>
    </div>
</div>

<BusBar></BusBar>
<CalendarBar @ref="Calendar"></CalendarBar>
<GalleryBar @ref="Gallery"></GalleryBar>
<YouTubeBar @ref="YouTube"></YouTubeBar>
```

Zdrojový kód 18: MainLayout – postranní panel (zdroj: Vlastní)

V MVC aplikaci je logika pro získávání počasí umístěna v souboru *app.js*. Blazor je plnohodnotný framework obsahující třídu *HttpClient*, se kterou je možné provádět HTTP dotazy. Implementace pro získání počasí je umístěna uvnitř *MainLayout.razor* a je znázorněna ve zdrojovém kódu č. 19.

```
protected override async Task OnInitializedAsync()
{
    try
    {
        HttpResponseMessage response = await Http.GetAsync("api/Common/Weather");
        response.EnsureSuccessStatusCode();

        Weather = (await response.Content.ReadAsStringAsync()).Replace(",", ".");
    }
    catch
    {
    }
}
```

Zdrojový kód 19: MainLayout - Logika počasí (zdroj: Vlastní)

Zdrojový kód přetěžuje funkci *OnInitializedAsync*, která se zavolá hned po inicializaci komponenty. V bloku *try* dochází pomocí objektu *Http* typu *HttpClient* volání *GET* na adrese „api/Common/Weather“. Výstup HTTP požadavku je uložen do proměnné *response* a následně je provedeno ověření úspěšnosti požadavku funkcí *EnsureSuccessStatusCode*. Následuje přečtení hodnoty obsahu požadavku do proměnné *Weather*. Ve zdrojovém kódu č. 20 je možné vidět výpis dané hodnoty počasí.

```
<div class="col-4 text-right d-none d-lg-block d-xl-block" id="temperature">
    <div class="text-center">
        <span class="temp">@Weather</span>
        <span class="gray">Meteo ČZU</span>
    </div>

    <img src="/images/celsium.png" alt="Stupně Celsia" loading="lazy">
</div>
```

Zdrojový kód 20: MainLayout - výpis počasí (zdroj: Vlastní)

Komponenty

Implementace klientské části všech stránek jsou rozdílné proti MVC. Data se zde předávají nikoliv modelem uvnitř infrastruktury, ale za pomocí HTTP požadavků, jak již bylo ukázáno ve zdrojovém kódu č. 20. Úvodní stránka v MVC tvoří přes 250 řádků. V Blazor aplikaci je to nemnoho přes 100 řádků. Je to dramatické zmenšení za pomocí využití komponent. Část úvodní stránky lze vidět ve zdrojovém kódu č. 21.

```razor
@page "/"

@inject HttpClient Http
@inject LayoutService LayoutService

<Title>iZUN.eu Univerzitní noviny ČZU | Dojíme pro Vás informace</Title>

<div class="row no-gutters" id="homepage">
    <Loader Object="Model">
        <div class="col-12" id="big-news">
            <div id="first-news">
                <a href="@Model.Link">
                    <div class="embed-responsive embed-responsive-3by2">
                        <img src="/clanky/@Model.ImageLarge" alt="@Model.Title" class="embed-responsive-item" loading="lazy">
                    </div>
                </a>

                <div class="overlay-bar d-none d-md-block">
                    <span>@Model.Date</span>
                    @if (!Model.CommentsDisabled)
                    {
                    <span><a href="@Model.Link#comments"><i class="fas fa-comment-dots"></i> @Model.Comments</a></span>
                    }
                    <span><button type="button" class="btn shadow-none" @onclick="LikeArticleAsync"><i class="@(Model.Liked ? "fas fa-heart text-danger" : "far fa-heart")"></i>
                      @TotalLikes</button></span>

                    <a href="@Model.Link">
                        <h2>@Model.Title</h2>
                    </a>
                </div>
            </div>

            <div class="mobile d-md-none @(string.IsNullOrWhiteSpace(Model.Image) ? "no-photo" : "")">
                <span>@Model.Date</span>
                @if (!Model.CommentsDisabled)
                {
                <span><a href="@Model.Link#comments" class="text-decoration-none"><i class="fas fa-comment-dots"></i> @Model.Comments</a></span>
                }
                <span><button type="button" class="btn shadow-none p-0" @onclick="LikeArticleAsync"><i class="@(Model.Liked ? "fas fa-heart text-danger" : "far fa-heart")"></i>
                  @TotalLikes</button></span>

                <h3>@Model.Title</h3>

                <a href="@Model.Link" class="read">Číst více</a>
            </div>
        </div>

        <div class="col-12" id="articles">
            <h3><i class="fas fa-book"></i> Další články</h3>

            <ArticleList Articles="articles"></ArticleList>
        </div>

        <a href="/clanky" class="btn btn-warning btn-block text-white rounded-0">Přejít na stránku s články <i class="fas fa-external-link-alt float-right"></i></a>
    </Loader>

    <TopArticles></TopArticles>
    <TopComments></TopComments>
</div>
```

Zdrojový kód 21: Úvodní stránka - Blazor (zdroj: Vlastní)

Tmavě zelené elementy např.: <*Title*> nebo <*Loader*> jsou vlastní komponenty, které usnadňují vývoj. Na prvním řádku je možné vidět syntaxi *@page*, která určuje URL vzor komponenty, v tomto případě tedy „/“ značí jako výchozí stránku aplikace. Dále jsou za pomocí syntaxe *@inject* získané objekty *HttpClient* pro tvoření HTTP dotazů a vlastní služby *LayoutService* pro správu rozložení stránky.

Největším problémem SPA aplikací je bezesporu SEO. Aplikace je načtena pouze poprvé, a to tedy znamená, že v případě úvodní stránky načteme název stránky (tag <*title*> - nikoliv <*Title*> ve zdrojovém kódu) někde ze statického souboru, který odešle při prvním požadavku server. V případě, že přejdeme na jinou stránku, např.: článek, prohlížeč nadále neprovádí obnovení stránky a nenačítá vše znovu, pouze odnačte komponentu úvodní stránky a místo ní načte komponentu článku. Tento problém je řešen za pomocí vlastního tagu <*Title*>, který zavolá JavaScriptovou metodu a nastaví hodnotu proměnné *document.title* na obsah tagu <*Title*>. Je nutné upozornit, že tato metoda je uživatelsky přívětivá, nikoliv pro roboty, kteří ve většině případů neumí interpretovat JavaScript.

Vlastní komponenta <*Loader*> je využívána zejména v případě, že uživatel nemá data a čeká na ně. Do atributu *Object* je umístěn odkaz na objekt a uvnitř komponenty je provedena podmínka. Za předpokladu, že objekt je prázdný, se vypisuje HTML značky znázorňující načítání v podobě kolečka s animací. Pokud objekt prázdný není, vypisuje se obsah uvnitř značky <*Loader*>.

Velká část zdrojového kódu byla ušetřena pomocí komponenty <*ArticleList*> využívána na stránce seznamu s články a v článku samotném, komponentě <*TopArticles*> a <*TopComments*> využívané na stránce seznamu s články.

4.2.3.2 Server-side

Serverová část aplikace je v samostatném projektu *iZUN.Wasm.Server*. Tato část aplikace je v některých případech velice podobná MVC aplikaci.

Controllery:

Aplikace obsahuje pouze 2 standardní controllery: *ExternalLoginController* spravující přihlašování pomocí externí služby (Google, Facebook, atd.) a *SitemapController* generující sitemapu webu v datovém formátu XML.

Web API:

Ostatní controllery jsou ve formě Web API (REST služby). Ve většině případů přijímají a odpovídají s datovým formátem *application/json*. Všechny tyto controllery se označují atributem *ApiController* a dědí pouze ze třídy *ControllerBase*. *ApiController* atribut provádí automaticky validaci vstupního modelu a není tedy potřebova dělat manuální kontrolu za pomocí podmínky s proměnnou *ModelState.IsValid*. Při chybném modelu je automaticky vrácena chyba HTTP 400.

_Host.cshtml:

Soubor *_Host.cshtml* obsahuje kostru HTML (tagy <html>, <head>, <meta>, <body>) a chová se jako Razor Page. Zároveň nahrazuje soubor *index.html*, který je vytvořen při založení projektu Blazor WebAssembly (klientská část). ASP.NET Core nadále slouží jako zprostředkovatel dat. Je možné vypisovat pomocí Razor Page obsah při prvotním požadavku. Pro změnu souboru bylo zapotřebí změnit mapování v souboru *Startup.cs* ze souboru *index.html* na stránku *_Host*.

Další výhodou souboru *_Host* je možnost využití Blazor server-side. Za pomocí server-side verze Blazoru je možné zpětně podporovat staré prohlížeče, které nemají implementované WebAssembly. Jedná se především o prohlížeč Internet Explorer nebo starší verze Safari.

Inicializace Blazor aplikace je možné vidět ve zdrojovém kódu č. 22.

```
<body>
    <app>
        <component type="typeof(iZUN.Server.App)" render-mode="ServerPrerendered">
    </app>

    <div id="components-reconnect-modal" class="components-reconnect-hide my-reconnect-modal"></div>

    <div id="blazor-error-ui">
        Došlo k neočekávané chybě!
        <a href="" class="reload">Načíst znovu</a>
        <a class="dismiss">✕</a>
    </div>
```

Zdrojový kód 22: Inicializace Blazor aplikace (zdroj: Vlastní)

Vlastní tag ASP.NET Core *<component>* inicializuje Blazor aplikaci typu *iZUN.Server.App* s módem *ServerPrerendered*. To znamená, že celá aplikace je při požadavku vyrenderována na serveru. Tato funkce je vhodná zejména pro roboty, kteří neumí interpretovat JavaScript jako webové prohlížeče. Obsah požadavku by pro ně byl prázdný, což není vhodné, pokud má být web indexován na vyhledávačích.

JavaScript kód ve zdrojovém kódu č. 23 poté zkouší, zda je podporovaný modul WebAssembly. Pokud je přítomen, je provedena inicializace WebAssembly klienta, v opačném případě se provede inicializace Server-side klienta prostřednictvím SignalR spojení se serverem.

```
<script type="text/javascript">
    var supported = (function ()
    {
        try
        {
            if (typeof WebAssembly === "object" && typeof WebAssembly.instantiate === "function")
            {
                var module = new WebAssembly.Module(Uint8Array.of(0x0, 0x61, 0x73, 0x6d, 0x01, 0x00, 0x00, 0x00));

                if (module instanceof WebAssembly.Module)
                    return new WebAssembly.Instance(module) instanceof WebAssembly.Instance;
            }
        } catch (e)
        {
        }

        return false;
    })();

    var script = document.getElementById("blazor-script");

    if (!supported)
    {
        if (/MSIE \d|Trident.*rv:/.test(navigator.userAgent))
            document.write('<script src="js/blazor.polyfill.min.js"><\/script>');

        document.write("<script src=\"_framework/blazor.server.js\"><\/script>");

        document.getElementById("components-reconnect-modal").innerHTML = '<div class="show">Připojení ztraceno. Vyčkejte prosím, probíhá pokus o připojení</div><div
          class="failed">Nelze se připojit. Klikněte pro pokus o připojení: <button type="button" class="btn btn-primary" onclick="window.Blazor.reconnect()">Připojit</button></
          div><div class="refused">Připojení bylo odmítnuto. <button type="button" class="btn btn-primary" onclick="window.location = \'/\'">Přejít na úvod</button></div>';
    }
    else
    {
        document.write("<script src=\"_content/Microsoft.AspNetCore.Components.WebAssembly.Authentication/AuthenticationService.js\"><\/script>");
        document.write("<script src=\"_framework/blazor.webassembly.js\"><\/script>");
    }
</script>
```

Zdrojový kód 23: JavaScript - WebAssembly podpora (zdroj: Vlastní)

V případě, že prohlížeč nepodporuje WebAssembly je serverová část aplikace využívána plně, protože implementuje klientskou část s rozdílem, že využívá serverové služby (v diagramu Data – Services) registrované v souboru *Startup* namísto HTTP dotazů.

Poslední část, kterou zaštiťuje soubor *_Host* je podpora dynamických meta tagů a názvu stránky pro SEO. Microsoft na tuto část při prvotním vydání nemyslel a bude pravděpodobně přidána v .NET 5. Ve zdrojovém kódu č. 24 je možné vidět část řešení, kde za pomocí *Request.Path.Value* proměnné je možné získat cestu, kterou uživatel nebo robot požaduje. Funkcí *string.Split* je možné cestu rozdělit na kousky a porovnat je se vzorem odpovídající článku. V případě, že vzor cesty je určen pro dynamické stránky, je z databáze vybrán titulek a případně popisek, které jsou dosazeny do příslušných tagů. Tímto je zaručena podpora i pro roboty, kteří dosud nepodporují interpretaci JavaScriptu.

```
try
{
    string path = Request.Path.Value;

    string[] split = path.Split('/', StringSplitOptions.RemoveEmptyEntries);

    if (split.Length > 0)
    {
        switch (split[0])
        {
            case "menza":
                ViewData["Title"] = "Jídelníček v menze ČZU";
                break;

            case "redakce":
                ViewData["Title"] = "Redakce";
                break;

            case "videa":
                ViewData["Title"] = "Videa";
                break;

            case "kalendar":
                ViewData["Title"] = "Kalendář akcí";
                break;

            case "clanky":
                ViewData["Title"] = "Články";
                break;

            case "galerie":
                ViewData["Title"] = "Galerie";
                break;

            case "hledat":
                ViewData["Title"] = "Hledat";
                break;

            default:
                {
                    string category = split[0];

                    if (split.Length > 1 && split[1] != "api")
                    {
                        var result = await db.Articles.Include(x => x.Cover).Where(x => x.LinkRewrite == split[1]).Select(x => new { x.Title,
                          x.Description, x.Cover.MediumThumb }).AsNoTracking().FirstOrDefaultAsync();

                        if (result != null)
                        {
                            ViewData["Title"] = result.Title;
                            ViewData["Description"] = string.IsNullOrWhiteSpace(result.Description) ? null : result.Description.Length > 160 ?
                              result.Description.Substring(0, 160) + "..." : result.Description;

                            if (!string.IsNullOrEmpty(result.MediumThumb))
                                ViewData["OgImage"] = $"/clanky/{result.MediumThumb}";
                        }
                    }
                }
                break;
        }
    }
}
```

Zdrojový kód 24: SEO - Request.Path.Value (zdroj: Vlastní)

4.3 Porovnání

Pro porovnání byly zvolené příslušná kritéria, které jsou spjaté s aplikacemi. Zvolené parametry pro porovnávání:

- Celková velikost a rychlost stránky
  - prvního načtení
  - druhého načtení (soubory jsou cachované)
  - při navigaci webem
- Hodnocení PageSpeed Insights
- Počet řádků zdrojového kódu (bez externích knihoven)

Porovnávání webových aplikací probíhalo pro aplikaci postavenou na ASP.NET Core MVC na adrese https://mvc.izun.eu. Pro aplikaci postavenou na Blazor frameworku na adrese https://izun.eu.

4.3.1 Celková velikost a rychlost stránky při prvním načtení

Prvotním načtením stránky se rozumí načtení úvodní stránky, kdy prohlížeč měl vymazanou cache. Testování probíhalo za pomocí nástroje Chrome DevTools verze 86 s připojením na optické síti o rychlostí 200 Mbps.

MVC

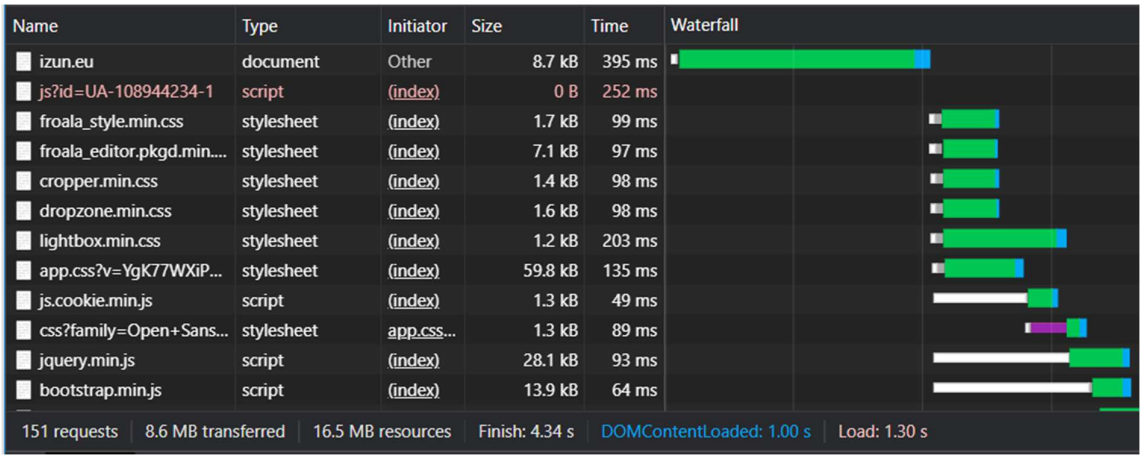
Name	Type	Initiator	Size	Time	Waterfall
mvc.izun.eu	document	Other	10.4 kB	373 ms	
js?id=UA-108944234-1	script	(index)	0 B	465 ms	
font-awesome.min.css	stylesheet	(index)	7.1 kB	27 ms	
lightbox.min.css	stylesheet	(index)	1.0 kB	356 ms	
froala_style.min.css	stylesheet	(index)	1.7 kB	32 ms	
froala_editor.pkgd.min....	stylesheet	(index)	6.9 kB	55 ms	
animate.min.css	stylesheet	(index)	3.7 kB	41 ms	
app.min.css?v=J3ANQ...	stylesheet	(index)	32.8 kB	253 ms	
vue.min.js	script	(index)	29.0 kB	68 ms	
css?family=Open+San...	stylesheet	app.mi...	946 B	36 ms	
axios.min.js	script / Redirect	(index)	155 B	129 ms	
vee-validate.min.js	script	(index)	18.1 kB	124 ms	

90 requests | 3.2 MB transferred | 5.6 MB resources | Finish: 11.91 s | DOMContentLoaded: 1.23 s | Load: 1.92 s

Obrázek 20: První načtení MVC (zdroj: Vlastní)

Na obrázku č. 20 je možné vidět vodopád načítání webové stránky. Úvodní dokument má velikost 10.4 kB a je načten za 373 ms. Na spodní liště je možné vidět, že bylo provedeno celkem 90 dotazů pro načtení celé stránky. Celková velikost při prvním načtení úvodní stránky je 3.2 MB dat a načtení trvalo 1.92 s.

Blazor



Name	Type	Initiator	Size	Time	Waterfall
izun.eu	document	Other	8.7 kB	395 ms	
js?id=UA-108944234-1	script	(index)	0 B	252 ms	
froala_style.min.css	stylesheet	(index)	1.7 kB	99 ms	
froala_editor.pkgd.min....	stylesheet	(index)	7.1 kB	97 ms	
cropper.min.css	stylesheet	(index)	1.4 kB	98 ms	
dropzone.min.css	stylesheet	(index)	1.6 kB	98 ms	
lightbox.min.css	stylesheet	(index)	1.2 kB	203 ms	
app.css?v=YgK77WXiP...	stylesheet	(index)	59.8 kB	135 ms	
js.cookie.min.js	script	(index)	1.3 kB	49 ms	
css?family=Open+Sans...	stylesheet	app.css...	1.3 kB	89 ms	
jquery.min.js	script	(index)	28.1 kB	93 ms	
bootstrap.min.js	script	(index)	13.9 kB	64 ms	

151 requests | 8.6 MB transferred | 16.5 MB resources | Finish: 4.34 s | DOMContentLoaded: 1.00 s | Load: 1.30 s

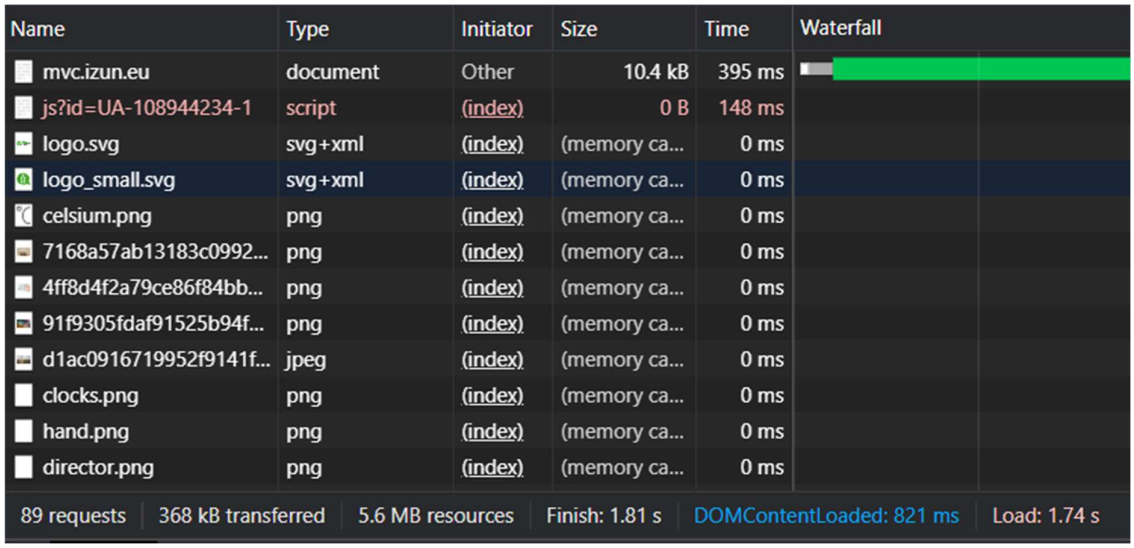
Obrázek 21: První načtení Blazor (zdroj: Vlastní)

Na obrázku č. 21 je možné vidět znovu vodopád, tentokrát aplikace Blazor. Úvodní dokument má pouze 8.7 kB, ale má menší prodlevu načítání v podobě 395 ms. Prodlevu ovlivňuje především server a konektivita (s každým dotazem je jiná). Pro načtení stránky bylo provedeno 151 dotazů, což je velké zhoršení. Jen 50 dotazů tvoří soubory Blazor aplikace s příponou .dll. Dalším zhoršením je celkový přenos dat, kdy celková velikost při prvním načtení úvodní stránky je 8.6 MB. Blazor aplikace je v celkovém načítání rychlejší o 620 ms.

4.3.2 Celková velikost a rychlost stránky při druhém načtení

Druhým načtením stránky se rozumí načtení úvodní stránky, kdy prohlížeč má uložené obrázky, CSS, JavaScript nebo .dll soubory v cache paměti.

MVC



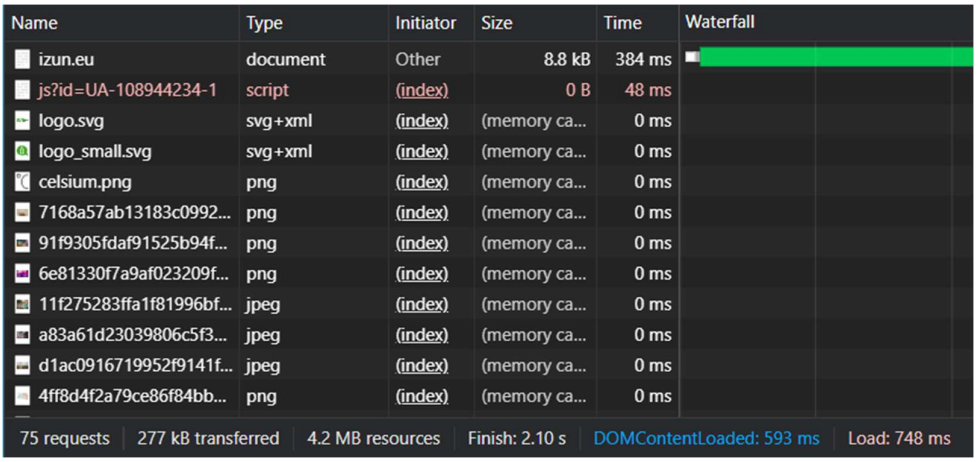
Name	Type	Initiator	Size	Time	Waterfall
mvc.izun.eu	document	Other	10.4 kB	395 ms	
js?id=UA-108944234-1	script	(index)	0 B	148 ms	
logo.svg	svg+xml	(index)	(memory ca...	0 ms	
logo_small.svg	svg+xml	(index)	(memory ca...	0 ms	
celsium.png	png	(index)	(memory ca...	0 ms	
7168a57ab13183c0992...	png	(index)	(memory ca...	0 ms	
4ff8d4f2a79ce86f84bb...	png	(index)	(memory ca...	0 ms	
91f9305fdaf91525b94f...	png	(index)	(memory ca...	0 ms	
d1ac0916719952f9141f...	jpeg	(index)	(memory ca...	0 ms	
clocks.png	png	(index)	(memory ca...	0 ms	
hand.png	png	(index)	(memory ca...	0 ms	
director.png	png	(index)	(memory ca...	0 ms	

89 requests | 368 kB transferred | 5.6 MB resources | Finish: 1.81 s | DOMContentLoaded: 821 ms | Load: 1.74 s

Obrázek 22: Druhé načtení MVC (zdroj: Vlastní)

Na obrázku č. 22 je možné vidět vodopád druhého načtení webové aplikace MVC. Úvodní dokument měl v tomto požadavku horší odezvu. Celkově bylo přeneseno 368 kB dat v 89 požadavcích. Načtení stránky trvalo 1.74 sekundy.

Blazor



Obrázek 23: Druhé načtení Blazor (zdroj: Vlastní)

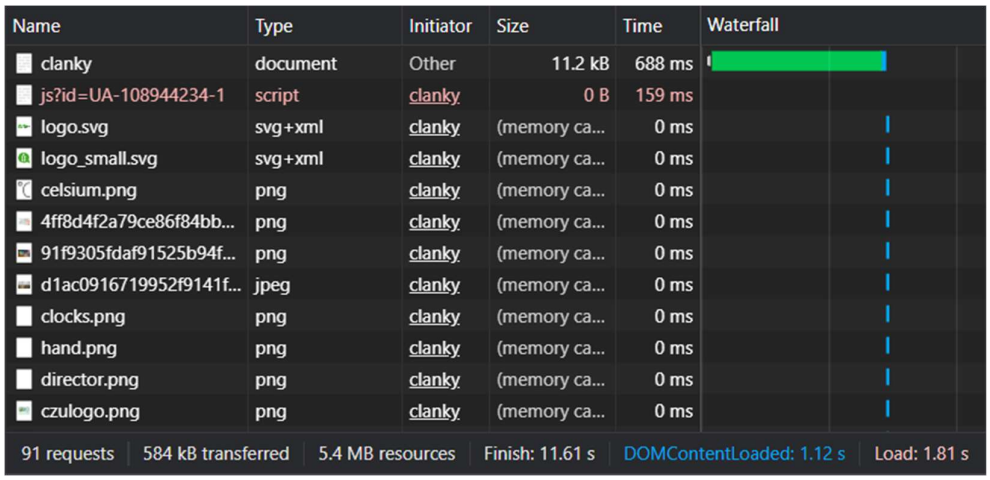
Na obrázku č. 23 je možné vidět vodopád druhého načtení Blazor aplikace. Celkově bylo přeneseno 277 kB dat v 75 požadavcích. Načtení stránky trvalo 748 ms.

Lze tedy konstatovat, že Blazor aplikace si vede ve všech směrech lépe od druhého načtení.

4.3.3 Celková velikost a rychlost stránky při navigaci webem

Pro porovnání navigací webem byla vybrána podstránka „Články“, protože je často navštěvována uživatelem, a to za podmínek bez vymazání cache. Navigací webem se rozumí načtení stránky „Články“ bez vymazání cache.

MVC



Obrázek 24: Navigace webem MVC (zdroj: Vlastní)

Na obrázku č. 24 je možné vidět vodopád při načítání podstránky „Články“. Pro načtení stránky bylo zapotřebí celkově 91 požadavků o celkové velikosti 584 kB s časem 1.81 sekundy.

Blazor

Name	Type	Initiator	Size	Time	Waterfall
0/	fetch	dotnet....	4.4 kB	534 ms	
TopArticles	fetch	dotnet....	1.3 kB	440 ms	
TopComments	fetch	dotnet....	317 B	262 ms	
7168a57ab13183c0992...	png	Other	124 kB	262 ms	
42e029392db77a365d...	jpeg	Other	20.0 kB	217 ms	
acef9e6da373a0490e8...	png	Other	23.1 kB	225 ms	
6 requests	174 kB transferred	172 kB resources			

Obrázek 25: Navigace webem Blazor (zdroj: Vlastní)

Na obrázku č. 25 je možné vidět vodopád při načítání podstránky „Články“ klientské aplikace Blazor. Pro načtení stránky bylo zapotřebí celkem 6 požadavků o velikosti 174 kB. První požadavek obsahující data článků má velikost pouze 4.4 kB a byl načten za 534 ms. Po tuto dobu se uživateli zobrazuje indikátor načítání a vůči MVC je jednoznačně rychlejší.

4.3.4 Hodnocení PageSpeed Insights

Hodnocení nástroje Google PageSpeed Insights je důležité z pohledu indexace webovým vyhledávačem. Zároveň nástroj odhaluje slabiny webové aplikace ať už v podobě pomalého serveru, nebo špatné konfigurace cache. Výstupem testu je skóre v rozmezí 0 až 100 jak pro mobil, tak pro desktop. Je důležité poznamenat, že test se chová, jako kdyby stránku nikdy předtím nenavštívil.

Google PageSpeed Insights test obsahuje 6 základních metrik:

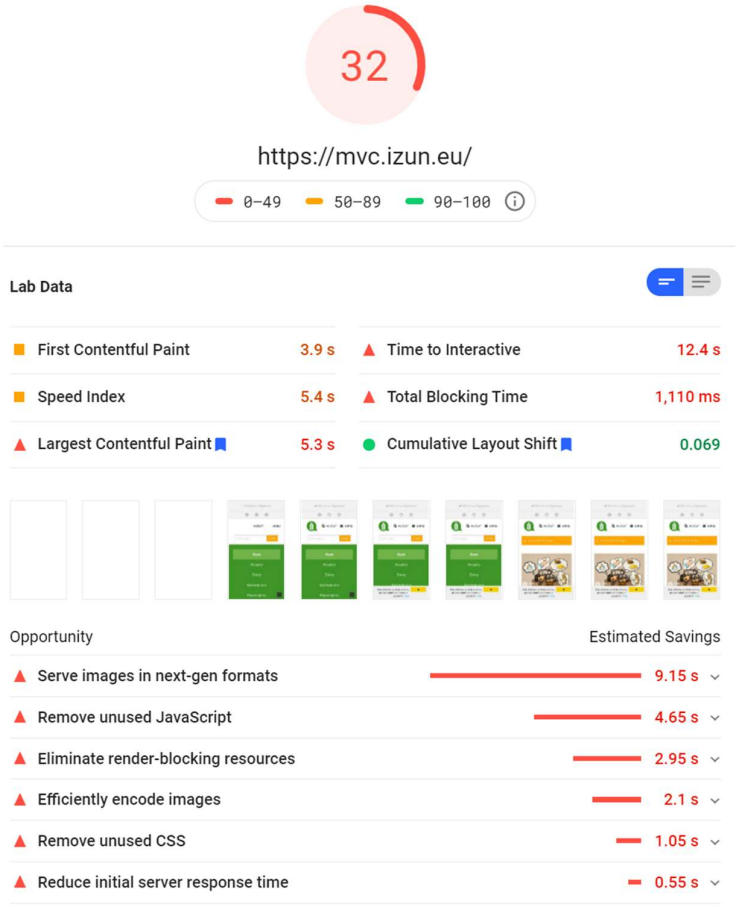
- First Contentful Paint
  - Označuje čas, kdy je vykreslen první text nebo obrázek.
- Speed Index
  - Označuje čas, za jak dlouho byl vidět obsah stránky.
- Largest Contentful Paint
  - Označuje čas, za jak dlouho byl vykreslen nejnáročnější text nebo obrázek.
- Time to Interactive
  - Označuje čas, za jak dlouho je stránka plně interaktivní.
- Total Blocking Time
  - Suma času mezi First Contentful Paint a Time to Interactive, kdy je úloha delší než 50 ms.

- Cumulative Layout Shift
    - Označuje míru pohybu viditelných objektů v okně při načtení.

Všechny následující testy z nástroje Google PageSpeed Insights byly pro webové aplikace ASP.NET Core MVC i Blazor testované 5x. Výsledky testů byly podobné v rámci skóre s rozdílem 2-3 body.

MVC

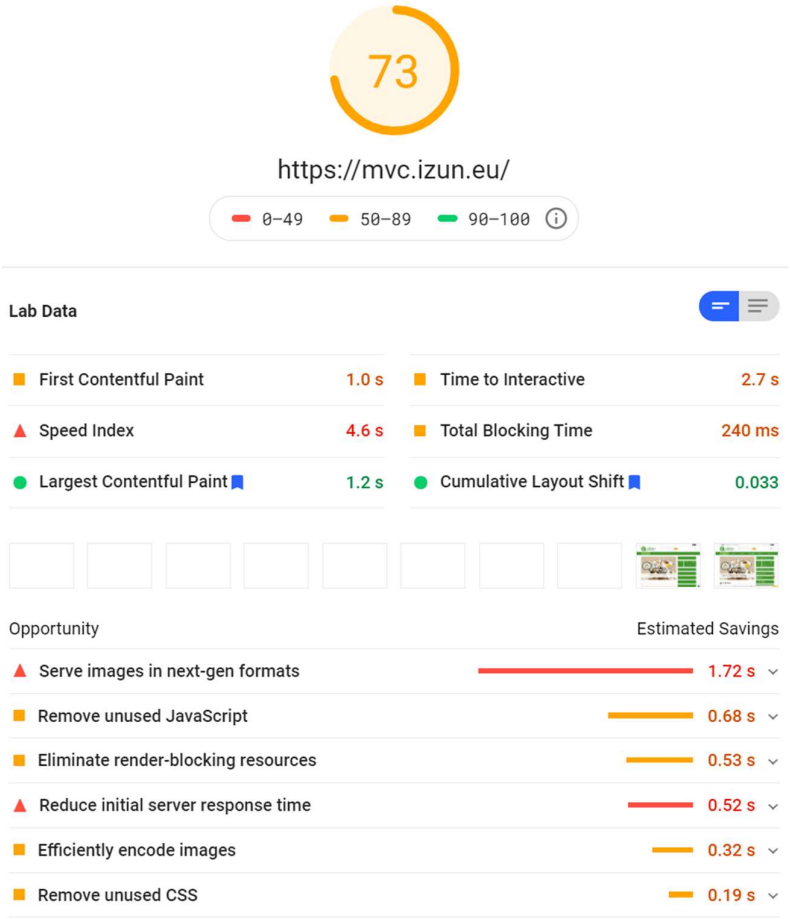
Mobil:



Obrázek 26: PageSpeed Insights MVC mobil (zdroj: Vlastní)

Na obrázku č. 26 lze vidět výstup hodnocení MVC pro mobil, které získalo skóre 32 bodů. Velká část penalizace je za špatnou kompresi obrázků a test doporučuje využít tzv. „next-gen“ formáty. Při využití next-gen formátů by se zajisté dosáhlo lepších výsledků, ale staré prohlížeče jako Internet Explorer nebo starší verze Safari by jej neuměly zpracovat.

Desktop:

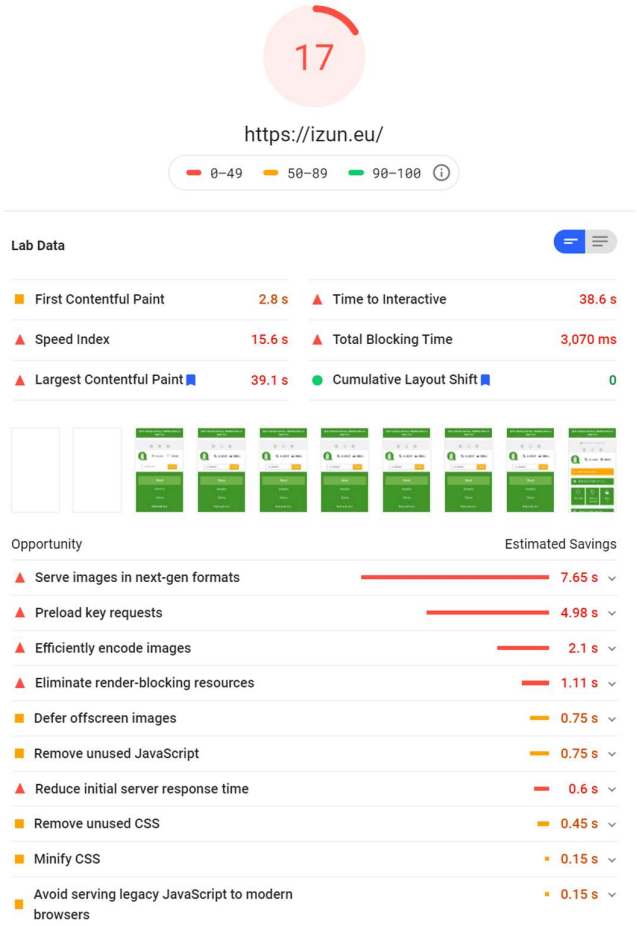


Obrázek 27: PageSpeed Insights MVC desktop (zdroj: Vlastní)

Na obrázku č. 27 je možné vidět výstup hodnocení MVC pro desktop, které získalo 73 bodů. Penalizace je zde znovu za špatnou kompresi obrázků a dále za Speed Index, který představuje, jak rychle je zobrazen obsah. V červených číslech se ocitá i položka odezvy serveru. Tuto položku není možné z pohledu vývoje ovlivnit (v době testu mohl být server vytížen jinými zákazníky).

Blazor

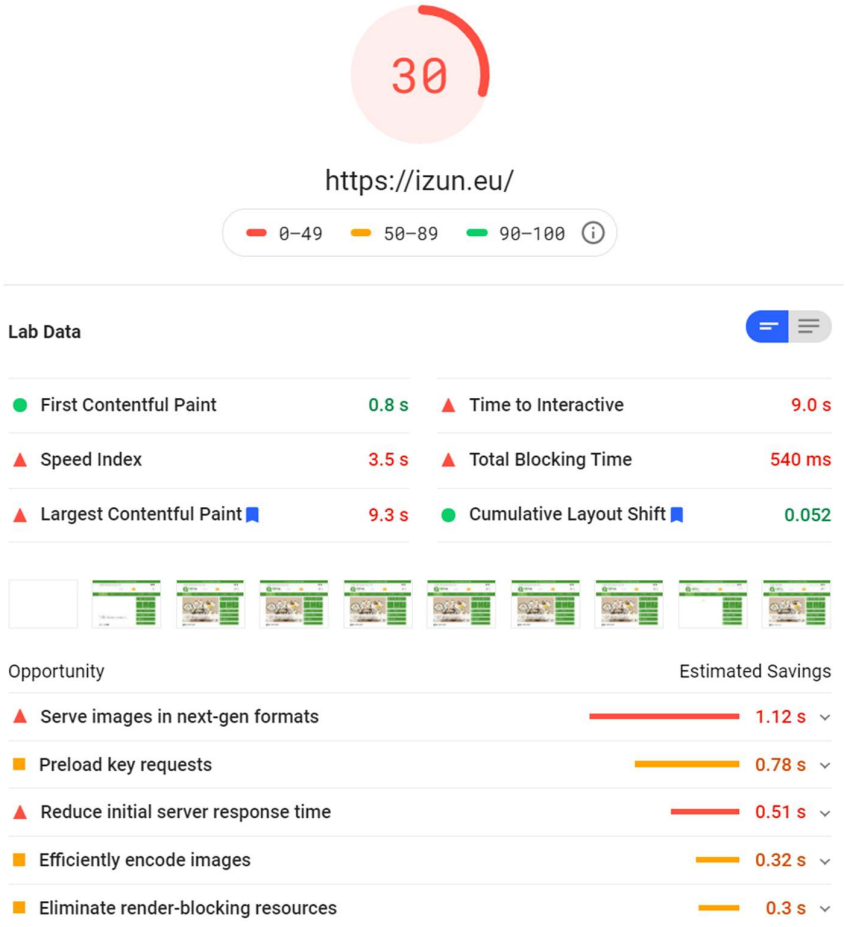
Mobil:



Obrázek 28: PageSpeed Insights Blazor mobil (zdroj: Vlastní)

Na obrázku č. 28 lze vidět výstup hodnocení Blazor aplikace pro mobil, které získalo pouhých 17 bodů. Velká část penalizace je udělena především za pomalé načítání (Speed Index je proti MVC mobilu až 3x horší, stejně tak Time To Interactive a Largest Contentful Paint) a velkou velikost aplikace, která má minimálně 8 MB při prvním načtení. Kladně je hodnocen Cumulative Layout Shift – tedy uživatelské prvky nebyly při načítání nijak posunuty.

Desktop:



Obrázek 29: PageSpeed Insights Blazor desktop (zdroj: Vlastní)

Na obrázku č. 29 je možné vidět výstup hodnocení Blazor aplikace pro desktop, které získalo 30 bodů a znamená to propad proti MVC. Opakuje se stejný problém s dlouhým načítáním aplikace, která vede k penalizaci (Time to Interactive a Largest Contentful Paint). Speed Index je proti MVC rychlejší o 1.1 sekundy

4.3.5 Počet řádků zdrojového kódu

Počet řádků zdrojového kódu je získán za pomocí příkazové řádky *Git Bash* s příkazem *git ls-files | xargs wc -l*. Výstupem je list řádků, kde každý řádek obsahuje cestu k souboru a počet řádků. Na posledním řádku je suma ve všech souborech.

V tabulce č. 1 je možné vidět, která webová aplikace měla kolik řádků ve zdrojovém kódu. Byly procházeny pouze soubory s příponou *.cs*, *.cshtml* a *.razor*.

Tabulka 1: Počet řádků zdrojového kódu v projektech (zdroj: Vlastní)

Blazor	**Blazor**		**MVC**
Projekt	**Počet řádků**	**Počet řádků**	**Počet řádků**
Client	8630	22896	25165
Server	12310		
Shared	1956		

Na základě součtu řádků v dílčích projektech webové aplikace Blazor – 22896 je vůči aplikaci MVC o 2269 řádků kratší a zároveň aplikace vykazuje stejné funkce.

5 Výsledky a diskuse

Výsledkem práce je porovnání dvou webových aplikací ve studentské organizaci iZUN.eu. První aplikace byla založena na návrhovém vzoru MVC a druhá na frameworku Blazor.

Na základě dvouletého provozu webové aplikace založené na ASP.NET Core MVC lze konstatovat, že jde o vhodnou technologii pro implementaci webových stránek na téma online zpravodajství. Stejně tak je možné využít webovou aplikaci postavenou na technologii Blazor. Hlavní výhoda technologie Blazor frameworku je, že přispívá ke zlepšení UX, kdy při navigaci není zapotřebí obnovit stránku.

5.1 Podpora starších prohlížečů

Tabulka č. 2 – Podpora prohlížečů zaznamenává přehled podporovaných prohlížečů v rámci webové aplikace MVC a Blazor. U webové aplikace na bázi MVC nemusela být řešena podpora prohlížečů, neboť technologie nijak nelimituje podporu prohlížečů. Naopak nová technologie Blazor není podporována v některých prohlížečích, zejména starších. V rámci diplomové práce byl implementován miniaturní JavaScript kód, který zjistí, zda je podporován WebAssembly modul. Pokud je podporován, načte se WebAssembly aplikace, v opačném případě je použit Blazor mód server-side rendering s polyfill knihovnou.

Tabulka 2: Podpora prohlížečů (zdroj: Vlastní)

Prohlížeč	MVC	Blazor
Apple Safari, včetně iOS	✓	Server-side rendering s polyfill knihovnou + WebAssembly od verze 11 a výše.
Google Chrome, včetně Androidu	✓	Server-side rendering s polyfill knihovnou + WebAssembly od verze 57 a výše.
Microsoft Edge	✓	Server-side rendering s polyfill knihovnou + WebAssembly od verze 16 a výše.

Microsoft Internet Explorer	✓	Pouze server-side rendering s polyfill knihovnou.
Mozilla Firefox	✓	Server-side rendering s polyfill knihovnou + WebAssembly od verze 53 a výše.

5.2 SEO

ASP.NET Core MVC technologie má výhodu – jedná se pouze o server-side technologii. S HTML kostrou lze manipulovat nejčastěji za pomocí objektu *ViewData*, případně podle vlastní architektury. Celý render stránky probíhá na straně serveru a robot dostává všechny informace bez potřeby interpretace JavaScriptu.

Blazor technologie, zejména WebAssembly část, která se spouští v klientské části, naráží na problém inicializace až na straně klienta. Větší část robotů stahují pouze HTML obsah bez následné interpretace JavaScript kódu či WebAssembly. Tímto celá technologie trpí na SEO v případě chybějícího server-side prerenderingu. Dalším problémem je chybějící implementace ze strany Microsoftu pro změnu meta tagů a nadpisu stránky. Programátor je v současné situaci nucen provést implementaci vlastního řešení.

V rámci diplomové práce je tento problém řešen za pomocí server-side prerenderingu a rozdělením URL cesty požadavku. Výstupem je obsah identický MVC aplikaci, který posléze inicializuje WebAssembly nebo server-side klienta. Ve WebAssembly je dále implementován vlastní tag <*Title*>, který pomocí JavaScript mění nadpis stránky. Přehled podpory SEO pro roboty je možné vidět v tabulce č. 3.

Tabulka 3: Podpora pro roboty (zdroj: Vlastní)

MVC	Blazor
Plná podpora pro roboty.	Plná podpora pro roboty pouze s vynuceným server-side pre-renderingem.

5.3 Rychlost a velikost

První část porovnání webových aplikací v praktické části tvořily testy z nástroje Chrome DevTools verze 86 s připojením na optické síti o rychlosti 200 Mbps. Výsledky testů jsou zaznamenané v tabulce č. 4. ASP.NET Core MVC bylo v parametrech *velikost dat celkem* a *počet dotazů* u akce *první načtení* lepší než Blazor framework. V ostatních parametrech/akcích

vykazoval Blazor framework lepších výsledků. Především akce *navigace webem* má velké rozdíly v parametrech. Je na místě zmínit, že velikost 8.6 MB při prvním načtení Blazor aplikace je 2,68x vyšší než u ASP.NET Core MVC, což může pro některé uživatele mobilních zařízení znamenat dlouhé načítání.

Tabulka 4: Chrome DevTools výsledky (zdroj: Vlastní)

Akce	Parametr	MVC	Blazor	Rozdíl
První načtení	Velikost dokumentu	10.4 kB	8.7 kB	2.7 kB
	Velikost dat celkem	3.2 MB	8.6 MB	5.4 MB
	Počet dotazů	90	151	61
	Doba načtení	1.92 s	1.30 s	620 ms
Druhé načtení	Velikost dat celkem	368 kB	277 kB	91 kB
	Počet dotazů	89	75	14
	Doba načtení	1.74 s	748 ms	992 ms
Navigace webem	Velikost dat celkem	584 kB	174 kB	410 kB
	Počet dotazů	91	6	85
	Doba načtení	1.81 s	534 ms	1 276 ms

• horší výsledek • lepší výsledek

Google PageSpeed Insights testy ohodnotily ASP.NET Core MVC kladně v porovnání s Blazor frameworkem, velké rozdíly jsou hlavně u desktopové verze. Výsledky testů jsou zaznamenané přehledně v tabulce č. 5. ASP.NET Core v mobilní verzi získal o 15 bodů více než framework Blazor a u desktopové verze získal o 43 bodů více. Naopak Blazor framework vynikal v parametru First Contentful Paint neboli první vykreslení textu či obrázku s výsledkem 2.8 sekund u mobilního zařízení (zrychlení 1.1 sekund) a u desktopového zařízení 0.8 sekund (zrychlení 0.2 sekund).

Tabulka 5: Google PageSpeed Insights výsledky (zdroj: Vlastní)

	Mobil			Desktop		
Parametr	**MVC**	**Blazor**	**Rozdíl**	**MVC**	**Blazor**	**Rozdíl**
First Contentful Paint	3.9 s	2.8 s	1.1 s	1.0 s	0.8 s	0.2 s
Speed Index	5.4 s	15.6 s	10.2 s	4.6 s	3.5 s	1.1 s
Laregst Contentful Paint	5.3 s	39.1 s	33.8 s	1.2 s	9.3 s	8.1 s
Time to Interactive	12.4 s	38.6 s	26.2 s	2.7 s	9.0 s	6.3 s
Total Blocking Time	1 110 ms	3 070 ms	1 960 ms	240 ms	540 ms	300 ms
Cumulative Layout Shift	0.069	0	0.069	0.033	0.052	0.019
Celkové skóre	32	17	15	73	30	43

• pomalý • dostačující • rychlý

Blazor WebAssembly v ostatních parametrech získává špatné hodnocení především v důsledku velké velikosti klientské aplikace (soubory s příponou .dll). Na základě výsledků lze konstatovat, že se aplikace Blazor WebAssembly nehodí pro uživatele mobilních zařízení, protože je náročná na přenos dat a výpočetní výkon, který ovlivňuje vykreslení. Přenos dat na mobilních zařízeních může být u některých uživatelů s nízkým datovým tarifem kritickým faktorem.

5.4 Zdrojový kód

Implementace webové aplikace na ASP.NET Core MVC obsahovala o 2269 řádků kódu více než nová technologie Blazor (přehled v tabulce č. 6). Je nutné poznamenat, že web byl koncipován na technologii Blazor pro podporu co nejvíce starších prohlížečů a tím pádem obsahuje některé komponenty dvakrát z důvodu použití jiných metod pro získávání dat (HTTP, služba uvnitř aplikace).

Z hlediska vývoje je subjektivním názorem autora jednodušší vývoj s technologií Blazor, protože komponenty nejsou nijak striktně vázané jako *views* s controllery v případě ASP.NET Core MVC. Christian Findlay taktéž uvádí, že pro vývoj aplikací by měl být zvolen Blazor s

tvrzením „Development teams with C# experience should consider Blazor for their next web app.“ [50].

Tabulka 6: Přehled velikosti zdrojového kódu (zdroj: Vlastní)

Počet řádků		
Blazor	**MVC**	**Rozdíl**
22896	25165	2269

5.5 Srovnání kritérií

Tabulka 7: Přehled srovnání kritérií (zdroj: Vlastní)

Název kritéria	Blazor	MVC	Rozdíl
Celková velikost při prvním načtení	8.6 MB	3.2 MB	5.4 MB
Celková velikost při druhém načtení	277 kB	368 kB	91 kB
Celková velikost při navigaci webem	174 kB	584 kB	410 kB
Rychlost při prvním načtení	1.3 s	1.92 s	620 ms
Rychlost při druhém načtení	748 ms	1.74 s	992 ms
Rychlost při navigaci webem	534 ms	1.81 s	1 276 ms
Hodnocení PageSpeed Insights – desktop (0-100)	30	73	43
Hodnocení PageSpeed Insights – mobil (0-100)	17	32	15
Počet řádků zdrojového kódu	22896	25165	2269

• horší výsledek • lepší výsledek

Tabulka č. 7 – Přehled srovnání kritérií zaznamenává výsledky kritérií z kapitoly 4.3 – Porovnání. Postupy měření jednotlivých kritérií jsou uvedeny v podkapitolách zmíněné kapitoly 4.3. Výsledky jednotlivých kritérií byly podrobněji popsány v předchozích kapitolách *Rychlost a velikost* a *Zdrojový kód*. Kritérium *celková velikost* a *rychlost* byly prováděny v rámci společného měření v podkapitolách 4.3.1, 4.3.2 a 4.3.3. Kritérium *hodnocení PageSpeed Insights* bylo provedeno v kapitole 4.3.4 a kritérium *počet řádků zdrojového kódu* v kapitole 4.3.5.

5.6 Možnosti rozšíření

V rámci diplomové práce nebyla prováděna implementace webové aplikace, která je postavena pouze na Blazor server-side. Tímto by technologie mohla v prvním načtení konkurovat ASP.NET Core MVC, protože neobsahuje velké soubory s příponou .dll, které se spouští na straně klienta, ale obsahuje pouze tenkého klienta pro komunikaci se serverem.

Další rozšíření, které by mohlo posunout technologii Blazor vůči ASP.NET Core MVC, se dočká v budoucím rozšíření .NET 5 nebo později. .NET 5 především míří na optimalizaci Blazor WebAssembly. Autor diplomové práce předpokládá dosáhnutí lepších výsledků v rámci testů za následujících podmínek:

- Server podporující Brotli kompresi.
    - Brotli komprese přináší proti GZip kompresi menší soubor průměrně až o 15 %.
- Lepší komprese obrázků za cenu ztráty kvality.
- Výkonnější server.
- Využití budoucí .NET 5 frameworku pro optimalizaci WebAssembly.
- Využití funkce Ahead of Time v budoucím .NET 6 frameworku pro sloučení .dll souborů do jednoho a komplikace do nativního WebAssembly.

6 Závěr

V teoretické části v kapitole 3.1 – Webové technologie byla nejprve provedena charakteristika současných webových technologií. Bylo vymezeno rozdělení na client-side a server-side technologii, používané protokoly a nejčastější datové formáty. Byla představena technologie WebAssembly fungující v sandboxu webového prohlížeče, která umožňuje využít prakticky jakýkoliv programovací jazyk pro programování klientské aplikace. Také byly představené nástroje pro testování rychlosti a výkonu webových stránek.

Kapitola 3.2 – ASP.NET Core MVC je zaměřená na charakteristiku technologie ASP.NET Core s návrhovým vzorem MVC. Byla představena historie technologie od roku 2002. V kapitole je popsán životní cyklus požadavku a další funkce jako např.: routing nebo model binding.

Bylo provedeno představení nové technologie v kapitole 3.3 – Blazor. Kapitola obsahuje charakteristiku technologie a výhod, které přináší. Klíčovou funkcí této technologie jsou tzv. komponenty. Technologie umožňuje dvě podoby: client-side pomocí technologie WebAssembly a server-side prostřednictvím SignalR komunikace.

Byla představena obdobná řešení od autorů Xiaoli Mao, Shanin Tamjidi a Christiana Findlaye. První řešení porovnává výkon frameworku Symfony, ASP.NET MVC a Node.js. Druhé řešení srovnává Single Page App a Multi Page App v rámci SEO. Třetí řešení srovnává tradiční webové aplikace a Single Page App – Blazor.

V praktické části bylo ověřeno na základě vytvoření a otestování dvou reálných webových aplikací, že lze využít jak ASP.NET Core MVC, tak Blazor framework pro online zpravodajství. Implementace probíhala ve studentské organizaci iZUN.eu za pomocí vývojového prostředí Visual Studio. Webová aplikace založená na ASP.NET Core MVC fungovala na doméně izun.eu od 12. února 2018 do 28. září 2020, kdy byla nahrazena technologií Blazor. Na základě funkčního několikaletého provozu je technologie ASP.NET Core MVC vhodná pro implementaci v online zpravodajství. Byly nalezeny nedostatky v podobě chybějící podpory u frameworku Blazor WebAssembly, která neobsahuje podporu pro změnu důležitých SEO tagů při změně stránky. Dále bylo zjištěno, že je nevhodné načítat webovou aplikaci založenou na Blazor WebAssembly poprvé na omezeném mobilním připojení z důvodu velké velikosti jednotlivých souborů aplikace. Pozitivní částí Blazor frameworku je menší zdrojový kód, vyšší uživatelská přívětivost a načítání pouze nezbytně nutných dat při navigaci. Z provedených měření vyplývá, že technologie je použitelnější ve všech směrech od druhého načtení stránky. Několikaměsíční provoz webové aplikace na

technologii Blazor se setkal s drobnými chybami, které byly posléze odstraněny. Technologie je podobně jako ASP.NET Core MVC vhodná pro implementaci v rámci online zpravodajství. Je nutné poznamenat, že rychlost prvního načtení v důsledku velké velikosti klientské aplikace by mohla řadu uživatelů odradit.

Z výsledků se nedá usuzovat, že by aktuálně byla jedna technologie lepší než druhá. Na základě syntézy poznatků z teoretické a praktické části lze říct, že v budoucnu se dá očekávat optimalizace frameworku Blazor jakožto nové technologie a odstranění nedostatků.

V rámci kapitoly 5.1 – Možnosti rozšíření, autor uvedl několik bodů, jak by mohla vypadat případná rozšíření ve prospěch výkonu.

7 Seznam použitých zdrojů

[1] ENGHEIM, Erik. A Guide to Different Web Technologies and How They Are Related. *Medium* [online]. Medium, 2019 [cit. 2020-07-07]. Dostupné z: https://medium.com/@Jernfrost/a-guide-to-different-web-technologies-and-how-they-are-related-a0951c7b43f4

[2] *WebAssembly* [online]. Mozilla, 2020 [cit. 2020-07-07]. Dostupné z: https://developer.mozilla.org/en-US/docs/WebAssembly

[3] Introduction *to the server side* [online]. Mozilla, 2019 [cit. 2020-07-07]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Introduction#:~:text=Server%2Dside%20code%20can%20be,%2C%20and%20NodeJS(JavaScript).

[4] CRAMPETE. Client-Side Scripting- Top Languages to Learn 2020. *Medium* [online]. Medium, 2019 [cit. 2020-07-07]. Dostupné z: https://medium.com/@crampeteb/client-side-scripting-top-languages-to-learn-4c8d736fa19a

[5] CRHONEK, Vítězslav. HISTORIE ZNAČKOVACÍCH JAZYKŮ. *Masarykova Univerzita* [online]. Brno: Masarykova Univerzita, 2003 [cit. 2020-07-07]. Dostupné z: https://www.fi.muni.cz/usr/jkucera/pv109/2003/xcrhonek.htm

[6] *Common client-side web technologies* [online]. Redmond: Microsoft, 2020 [cit. 2020-12-20]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-client-side-web-technologies

[7] *JEŽEK*, David. HTML5 je hotové, vývoj trval 15 let. *Deep in IT* [online]. Praha: CDR, 2014 [cit. 2020-07-07]. Dostupné z: https://diit.cz/clanek/html5-je-hotove-vyvoj-trval-15-let

[8] Comparison *of CSS versions* [online]. Telangana: Tutorialspoint, 2018 [cit. 2020-07-07]. Dostupné z: https://www.tutorialspoint.com/Comparison-of-CSS-versions

[9] New *CSS3 Features: Learn How This Version Improved CSS2* [online]. Bitdegree, 2016 [cit. 2020-07-07]. Dostupné z: https://www.bitdegree.org/learn/css3-features

[10] *JavaScript*. *MDN Web docs* [online]. Mountain View: Mozilla, 2020 [cit. 2020-11-21]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript

[11] HIWARALE, Uday. How does JavaScript and JavaScript engine work in the browser and node?. *Medium* [online]. Medium, 2018 [cit. 2020-11-21]. Dostupné z: https://medium.com/jspoint/how-javascript-works-in-browser-and-node-ab7d0d09ac2f

[12] GUOSS, Danny. The history and legacy of jQuery. *Logrocket* [online]. Logrocket, 2019 [cit. 2020-07-08]. Dostupné z: https://blog.logrocket.com/the-history-and-legacy-of-jquery/

[13] *WANYOIKE*, Michael. History of front-end frameworks. *Logrocket* [online]. Logrocket, 2018 [cit. 2020-07-08]. Dostupné z: https://blog.logrocket.com/history-of-frontend-frameworks/

[14] *React: A JavaScript library for building user interfaces* [online]. Menlo Park: Facebook, 2021 [cit. 2021-03-06]. Dostupné z: https://reactjs.org/

[15] *Vue.js: The Progressive JavaScript Framework* [online]. Evan You, 2021 [cit. 2021-03-06]. Dostupné z: https://vuejs.org/

[16] WebAssembly *Concepts* [online]. Mozilla, 2020 [cit. 2020-07-09]. Dostupné z: https://developer.mozilla.org/en-US/docs/WebAssembly/Concepts

[17] Can *I use...* [online]. Fyrd, 2020 [cit. 2020-07-09]. Dostupné z: https://caniuse.com/#feat=wasm

[18] *BASU*, Sayanee. Ruby on Rails Study Guide: The History of Rails. *Envato* [online]. Envato, 2013 [cit. 2020-07-08]. Dostupné z: https://code.tutsplus.com/articles/ruby-on-rails-study-guide-the-history-of-rails--net-29439

[19] Evolution *of HTTP* [online]. Mozilla, 2019 [cit. 2020-07-07]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Evolution_of_HTTP

[20] HTTP/2. In: *Cyber Security Leader* [online]. Imperva [cit. 2020-07-07]. Dostupné z: https://www.imperva.com/learn/performance/http2/

[21] HTTP *request methods* [online]. Mozilla, 2020 [cit. 2020-07-07]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

[22] *What is REST* [online]. restfulapi.net [cit. 2020-07-07]. Dostupné z: https://restfulapi.net/

[23] *What is REST?: Learn about how to design web services using the REST paradigm* [online]. New York: Codeacademy, 2021 [cit. 2021-03-06]. Dostupné z: https://www.codecademy.com/articles/what-is-rest

[24] SHARMA, Vaibhav. AJAX : Asynchronous JavaScript And XML. *Medium* [online]. Medium, 2017 [cit. 2020-07-07]. Dostupné z: https://medium.com/@vsvaibhav2016/ajax-asynchronous-javascript-and-xml-1914da30458f

[25] *GARRETT*, James. AJAX: a new approach to Web applications. In: *Adaptivepath* [online]. Adaptivepath [cit. 2020-07-07]. Dostupné z: http://adaptivepath.org/ideas/ajax-new-approach-web-applications/

[26] WebSockets - *A Conceptual Deep-Dive* [online]. Alby, 2018 [cit. 2020-07-07]. Dostupné z: https://www.ably.io/concepts/websockets

[27] Websocket vs Ajax polling scheme. In: *Resellers Panel* [online]. Resellers Panel, 2016 [cit. 2020-07-07]. Dostupné z: https://blog.resellerspanel.com/hepsia-control-panel/supervisor-run-background-processes.html/attachment/websocket-scheme

[28] SAFRIS, Seva. A Deep Look at JSON vs. XML, Part 1: The History of Each Standard. *Toptal* [online]. Toptal, 2019 [cit. 2020-07-08]. Dostupné z: https://www.toptal.com/web/json-vs-xml-part-1

[29] *JSON vs XML: What's the Difference?* [online]. Ahmedabad City: Guru99, 2021 [cit. 2021-03-06]. Dostupné z: https://www.guru99.com/json-vs-xml-difference.html

[30] *SAFRIS*, Seva. A Deep Look At JSON vs. XML, Part 2: The Strengths and Weaknesses of Both. *Toptal* [online]. Toptal, 2019 [cit. 2020-07-08]. Dostupné z: https://www.toptal.com/web/json-vs-xml-part-2

[31] *SIMIC*, Sofija. 7 Best Website Speed And Performance Testing Tools. *Phoenix NAP* [online]. phoenixNAP, 2019 [cit. 2020-07-15]. Dostupné z: https://phoenixnap.com/kb/best-website-speed-performance-test-tools

[32] *Chrome DevTools* [online]. Mountain View: Google, 2020 [cit. 2020-07-15]. Dostupné z: https://developers.google.com/web/tools/chrome-devtools

[33] GARCIA, Daniel Jimenez. The History of ASP.NET – Part I. *DotNetCurry* [online]. DotNetCurry, 2019 [cit. 2020-07-09]. Dostupné z: https://www.dotnetcurry.com/aspnet/1492/aspnet-history-part-1

[34] GARCIA, Daniel Jimenez. The History of ASP.NET – Part II. *DotNetCurry* [online]. DotNetCurry, 2019 [cit. 2020-07-09]. Dostupné z: https://www.dotnetcurry.com/aspnet/1493/aspnet-history-part-2-mvc

[35] *GARCIA*, Daniel Jimenez. The History of ASP.NET – Part III (Covers ASP.NET Core). *DotNetCurry* [online]. DotNetCurry, 2019 [cit. 2020-07-09]. Dostupné z: https://www.dotnetcurry.com/aspnet/1494/aspnet-history-part-3-core

[36] *What's new in ASP.NET Core 3.0* [online]. Redmond: Microsoft, 2019 [cit. 2020-07-09]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/release-notes/aspnetcore-3.0?view=aspnetcore-3.1

[37] *Overview of ASP.NET Core MVC* [online]. Redmond: Microsoft, 2020 [cit. 2020-07-10]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-3.1

[38] *Filters in ASP.NET Core: How filters work* [online]. Redmond: Microsoft, 2020 [cit. 2020-07-10]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/filters?view=aspnetcore-3.1#how-filters-work

[39] *SANDERSON*, Steve. Blazor: a technical introduction: Deeper technical details about Blazor. *Steve Sanderson's blog* [online]. 2018 [cit. 2020-07-11]. Dostupné z: https://blog.stevensanderson.com/2018/02/06/blazor-intro/

[40] *Introduction to ASP.NET Core Blazor* [online]. Redmond: Microsoft, 2020 [cit. 2020-07-11]. Dostupné z: https://docs.microsoft.com/en-gb/aspnet/core/blazor/?view=aspnetcore-3.1

[41] TAUQIR. Blazor Lifecycle Methods in .Net Core 3.1 – Basic Explanation. *Execute Commands: Software Application Articles* [online]. 2020 [cit. 2020-07-13]. Dostupné z: https://executecommands.com/aspcore-blazor-lifecycle-methods/

[42] ASP.*NET Core Blazor hosting models* [online]. Redmond: Microsoft, 2020 [cit. 2020-07-12]. Dostupné z: https://docs.microsoft.com/en-gb/aspnet/core/blazor/hosting-models?view=aspnetcore-3.1

[43] ASP.*NET Core Blazor routing* [online]. Redmond: Microsoft, 2020 [cit. 2020-07-13]. Dostupné z: https://docs.microsoft.com/en-gb/aspnet/core/blazor/fundamentals/routing?view=aspnetcore-3.1

[44] Call *JavaScript functions from .NET methods in ASP.NET Core Blazor* [online]. Redmond: Microsoft, 2020 [cit. 2020-07-13]. Dostupné z: https://docs.microsoft.com/en-gb/aspnet/core/blazor/call-javascript-from-dotnet?view=aspnetcore-3.1

[45] ASP.*NET Core Blazor event handling* [online]. Redmond: Microsoft, 2020 [cit. 2020-07-13]. Dostupné z: https://docs.microsoft.com/en-gb/aspnet/core/blazor/components/event-handling?view=aspnetcore-3.1

[46] ASP.*NET Core Blazor data binding* [online]. Redmond: Microsoft, 2020 [cit. 2020-07-13]. Dostupné z: https://docs.microsoft.com/en-gb/aspnet/core/blazor/components/data-binding?view=aspnetcore-3.1

[47] ASP.*NET Core Blazor layouts* [online]. Redmond: Microsoft, 2020 [cit. 2020-07-13]. Dostupné z: https://docs.microsoft.com/en-gb/aspnet/core/blazor/layouts?view=aspnetcore-3.1

[48] MAO, Xiaoli. *COMPARISON BETWEEN SYMFONY, ASP.NET MVC, AND NODE.JS EXPRESS FOR WEB DEVELOPMENT* [online]. Fargo, North Dakota, 2018, 42 s. [cit. 2020-10-14]. Dostupné z: https://library.ndsu.edu/ir/bitstream/handle/10365/28191/Comparison%20between%20Symfony,%20ASP.NET%20MVC,%20And%20Node.js%20Express%20for%20Web%20Development.pdf?sequence=1. Diplomová práce. North Dakota State University of Agriculture and Applied Science. Vedoucí práce Dr. Simone Ludwig.

[49] TAMJIDI, Shahin. *Comparison between SPA and MPA: Competition to get the best ranking on SEO* [online]. Karlskrona Sweden, 2017, 42 s. [cit. 2020-10-14]. Dostupné z: https://www.diva-portal.org/smash/get/diva2:1143657/FULLTEXT02. Diplomová práce. Blekinge Institute of Technology. Vedoucí práce Conny Johansson.

[50] FINDLAY, Christian. Blazor Vs. Traditional Web Apps. *CHRISTIAN FINDLAY* [online]. Melbourne: Findlay, 2020 [cit. 2020-11-25]. Dostupné z: https://christianfindlay.com/2020/07/09/blazor-vs-traditional-web-apps