Aplikace demonstrující možnosti webového standardu WebAssembly a webového frameworku Blazor

Michal Horáček

Bakalářská práce
2019

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2018/2019

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Michal Horáček**
Osobní číslo:	**A16437**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Softwarové inženýrství**
Forma studia:	**prezenční**
Téma práce:	**Aplikace demonstrující možnosti webového standardu WebAssembly a webového frameworku Blazor**
Téma anglicky:	**An Application for Demonstrating the Possibilities of using the WebAssembly and Blazor Framework**

Zásady pro vypracování:

1. **Popište webový standard WebAssembly a související technologie.**
2. **Popište webový Framework Blazor a možnosti jeho nasazení.**
3. **Navrhněte ukázkovou aplikaci s využitím webového standardu WebAssembly a webového frameworku Blazor.**
4. **Vytvořte příklady a vzorová řešení demonstrující klíčové prvky řešení typických webových aplikací.**
5. **Popište klíčové části navrženého řešení.**
6. **Demonstrujte výsledky.**

Rozsah bakalářské práce:

Rozsah příloh:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **Webassembly [online]. [cit. 2018-09-26]. Dostupné z: https://webassembly.org**
2. **Blazor [online]. [cit. 2018-09-26]. Dostupné z: https://blazor.net/**
3. **NAGEL, Christian. Professional C# 6 and .Net Core 1.0. Indianapolis, IN: Wrox, a Wiley brand, published by John Wiley & Sons, 2016. ISBN 111909660X.**
4. **J. PRICE, Mark. C# 6 and .NET Core 1.0: Modern Cross-Platform Development. Birmingham: Packt Publishing, 2016. ISBN 9781785285691.**
5. **SINGLETON, James. ASP.NET Core 1.0 High performance. Birmingham: Packt Publishing, 2016. ISBN 9781785881893.**

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

Datum zadání bakalářské práce: **3. prosince 2018**

Termín odevzdání bakalářské práce: **15. května 2019**

Ve Zlíně dne 7. prosince 2018

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

doc. Mgr. Milan Adámek, Ph.D.
děkan

prof. Mgr. Roman Jašek, Ph.D.
garant oboru

Prohlašuji, že

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

Prohlašuji,

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

Ve Zlíně, dne 15. 5. 2019 Michal Horáček, v. r.

ABSTRAKT

Bakalářská práce se zabývá možnostmi webového standardu WebAssembly a webového frameworku Blazor. Cílem je ukázat nový webový framework Blazor a možnosti jeho nasazení. Teoretická část se zabývá webovým standardem WebAssembly, frameworkem Blazor a technologiemi, které využívají. V praktické části je vytvořena demonstrující aplikace a popsána vzorová řešení klíčových prvků typických webových aplikací.

Klíčová slova: WebAssembly, Blazor

ABSTRACT

The bachelor thesis is focused on the possibilities of web standard WebAssembly and web framework Blazor. The aim is to show new Blazor web framework and its deployment options. The theoretical part of thesis deals with the WebAssembly web standard, Blazor web framework and the technologies which they use. In the practical part of thesis is created application for demonstration and described solutions of key elements of typical web application.

Keywords: WebAssembly, Blazor

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

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

OBSAH

ÚVOD **9**
I TEORETICKÁ ČÁST **10**
1 WEBASSEMBLY **11**
1.1 HISTORIE 11
1.2 WEBASSEMBLY FORMÁTY 11
1.3 KOMPILACE KÓDU A VÝKON WEBASSEMBLY 12
1.4 ASM.JS 14
1.5 LLVM 14
1.6 EMSCRIPTEN 15
2 FRAMEWORK BLAZOR **16**
2.1 HISTORIE 16
2.2 JAVASCRIPT INTEROPERABILITA 16
2.3 LADĚNÍ PROGRAMŮ 19
2.4 OPTIMALIZACE 20
2.5 MONO 20
2.5.1 Interpreted 21
2.5.2 AOT 21
2.6 RAZOR 22
2.6.1 Razor components 23
2.7 MOŽNOSTI NASAZENÍ 23
2.7.1 Client-side 23
2.7.2 Server-side 25
2.8 JAZYK C# 26
2.8.1 Historie 26
2.9 .NET 26
2.9.1 .NET Standard 27
2.9.2 .NET implementace 27
2.9.3 .NET runtime 27
2.10 JAVASCRIPT 28
2.10.1 Historie 28
II PRAKTICKÁ ČÁST **29**
3 VYTVOŘENÍ PROJEKTU **30**
4 ZÁKLADNÍ STRUKTURA **31**
4.1 CLIENT 31
4.2 SERVER 32
4.3 SHARED 32
5 UKÁZKA PRÁCE S DATABÁZÍ **33**
5.1 SERVER-SIDE 33
5.1.1 Products 33
5.1.2 CrudTableDataController 34

5.2 CLIENT-SIDE 35
5.2.1 CrudTableComponent 35
5.3 SHARED 38
6 TELERIK KOMPONENTY 40
6.1 NASTAVENÍ PROJEKTU 40
6.2 VYTVOŘENÍ TABULKY 40
7 VELIKOST PROSTŘEDKŮ STAHOVANÝCH DO PROHLÍŽEČE 42
ZÁVĚR 43
SEZNAM POUŽITÉ LITERATURY 44
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK 48
SEZNAM OBRÁZKŮ 49
SEZNAM PŘÍLOH 50

ÚVOD

V dnešní době se v drtivé většině používá JavaScript pro client-side aplikace. S příchodem WebAssembly se nám otvírají možnosti, které doposud byly velmi obtížné realizovat. Tento binární formát nám umožňuje psát client-side a server-side webové aplikace téměř v jednom programovacím jazyce nebo umožňuje zrychlení načítání her v prohlížeči. Firma Microsoft využila tohoto webového formátu k vytvoření nového webového frameworku Blazor.

Tento webový framework umožňuje vytvářet client-side a server-side část aplikace v jediném jazyce. Blazor je stále ve vývoji, ale již teď jde vidět potenciál, který přináší.

Teoretická část práce se zabývá popisem webového frameworku WebAssembly. Popisuje historii, formáty a postup kompilace kódu do WebAssembly. Dále jsou zde popsané technologie, které se využívají společně s WebAssembly. Mezi tyto technologie patří LLVM, ASM.js a Emscripten. Dále se v této části popisuje webový framework Blazor a technologie, které využívá. V závěru teoretické části je popsána JavaScript interoperabilita, ladění programů, optimalizace programů a možnost nasazení aplikace. Mezi využité technologie patří běhové prostředí MONO, programovací syntaxe Razor, jazyk C#, jazyk JavaScript a vývojářská platforma .NET.

Praktická část práce se zabývá návrhem a popisem vytvoření aplikace s využitím frameworku Blazor. V úvodu je popsáno vytvoření nového projektu ve frameworku Blazor a postup vytvoření nového ASP .NET Core Hosted projektu. Dále je rozebrána struktura nového projektu a jeho jednotlivé podprojekty Client, Server a Shared. Následně praktická část popisuje postup vytvoření tabulky s možností editace jednotlivých prvků tabulky. Jednotlivé podkapitoly popisují postup vytvoření API na straně serveru a vytvoření komponenty tabulky na straně klienta. Další kapitola se zabývá ukázkou využití komponentů třetích stran a konkrétně popisuje přidání Telerik NuGet balíčků do projektu. Následně je popsaný postup vytvoření TelerikGrid komponenty. Poslední kapitola se zabývá soubory, které se stahují do prohlížeče při přechodu na webovou stránku. Je zde popsána velikost souborů autorovy webové aplikace.

I. TEORETICKÁ ČÁST

1 WEBASSEMBLY

WebAssembly je webový standard definující binární formát, který slibuje skoro nativní rychlosti pro webové aplikace. Je navržen tak, aby se dal teoreticky jakýkoli programovací jazyk přeložit do WebAssembly. V dnešní době všechny hlavní prohlížeče podporují WebAssembly. Zatím se používá WebAssembly spíše jako doplněk k JavaScriptu ale v budoucnosti by ho mohl zcela nahradit. Kde JavaScript je flexibilní, dynamicky deklarovaný a přeposílaný v lidsky čitelném kódu, WebAssembly je velmi rychlý, staticky deklarovaný a přeposílaný v kompaktním binárním kódu. Vývojáři by měli uvažovat o WebAssembly u případů, kde je velký důsledek na výkon například u her, streamování hudby nebo úprava videa.[1]

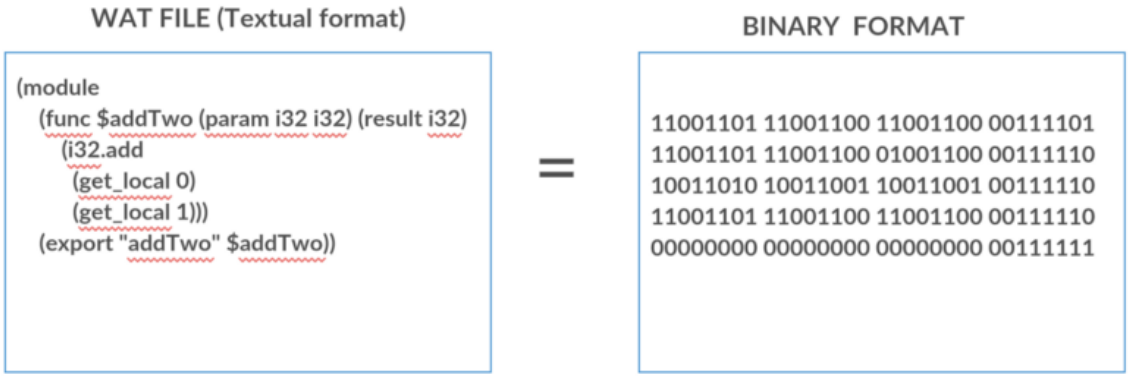
Obrázek 1 – Blazor oficiální logo [19]

1.1 Historie

Poprvé byl uveden 17. června 2015 [3]. V roce 2016 byla demonstrována hra Angry Bots, která byla založená na herním enginu Unity v prohlížečích Firefox, Chromium, Google Chrome a Microsoft Edge [4]. V listopadu 2017 firma Mozzila oznámila podporu WebAssembly ve všech hlavních prohlížečích [5]. V únoru 2018 zveřejnila pracovní skupina WebAssembly Working Group tři návrhy standardů pro Core Specification, Javascript Interface a Web API [6].

1.2 WebAssembly formáty

V březnu 2017 skupina WebAssembly Comunity Group, která reprezentuje čtyři prohlížeče (Chrome, Edge, Firefox a WebKit), dosáhla shody na podobě binárního formátu, Javascriptového API a interpreteru referencí. Definovali WebAssembly binární formát, který není určen pro používání lidmi a také lidmi čitelný linear assembly bytecode (text) formát, který se podobá tradičním assembly jazykům (viz Obrázek číslo 2) [7].



Obrázek 2 - Ukázka WebAssembly formátu kódu [46]

1.3 Kompilace kódu a výkon WebAssembly

Pro kompilaci kódu v jazyce C do WebAssembly je potřeba překladač, který podporuje tuhle funkci. Pro tento případ se použije překladač Enscripten[13]. Následující postup je převzatý ze stránky Mozzila.org [2].

Nejprve je potřeba nainstalovat a aktivovat Emscripten SDK [14]. Vytvoří se soubor hello.c s následujícím kódem a uloží na disku.

```
#include <stdio.h>

int main(int argc,char ** argv)
{
    printf("Hello World\n");
}
```

Otevře se příkazový řádek a přemístí se do složky s připraveným kódem. Nyní se spustí následující příkaz pro přeložení kódu do WebAssembly:

emcc hello.c -s WASM=1 -o hello.html

Zvolené možnosti v tomto příkazu jsou:

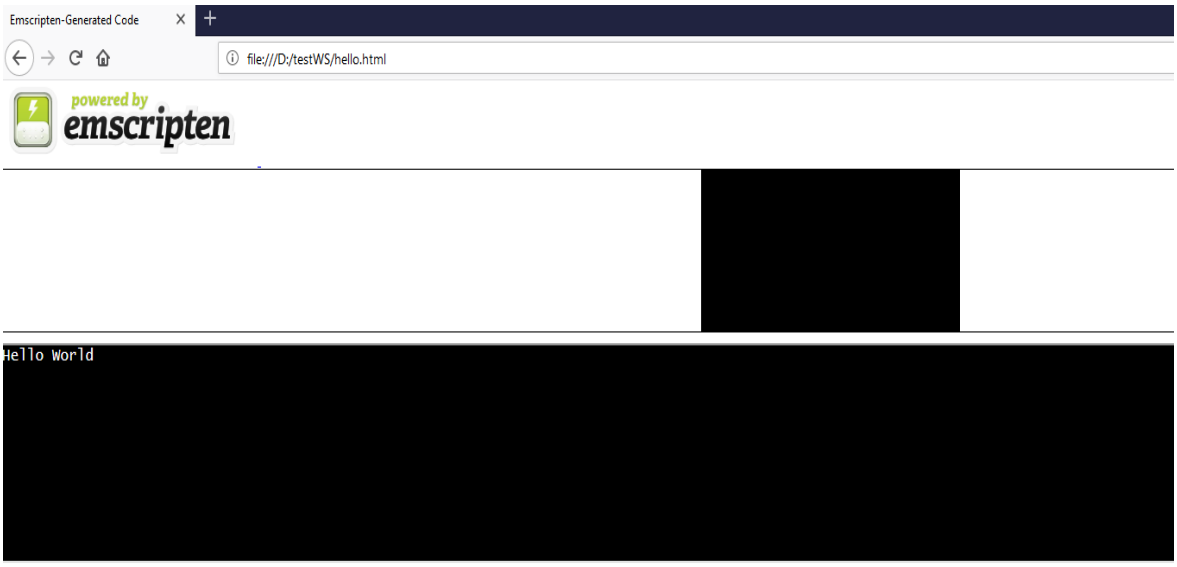
- -s WASM=1: Specifikuje, že výstup bude ve formátu WebAssembly, pokud není přímo specifikované, tak Emscripten překládá do formátu asm.js.

- -o hello.html: Specifikuje, že Emscripten bude přímo generovat HTML stránku ze vstupního kódu a zároveň javascript kód, který zkompiluje a vytvoří instanci WebAssembly pro použití na web.

Spuštění příkazu se vygenerují 3 soubory:

- Binární soubor s WebAssembly kódem (hello.wasm)
- JavasScript soubor obsahující „lepící“ kód pro překládání nativních C funkcí a JavaScriptu (hello.js).
- HTML soubor pro načtení, překládání a vytváření instance wasm kódu a zobrazení výsledku v prohlížeči (hello.html)

Pro zobrazení výsledku se otevře výsledný hello.html soubor v prohlížeči, který podporuje WebAssembly. V základu ho obsahují Firefox 52+, Chrome 57+ a Opera 44+ (WebAssembly kód se dá spustit ve Firefox 47+ povolením javascript.options.wasm v about:confiq nebo Chrome 51+ povolením Experimental WebAssembly možnosti v chrome://flags). Výsledek se nachází v Emscripten konzoli na webové stránce (viz Obrázek číslo 3).

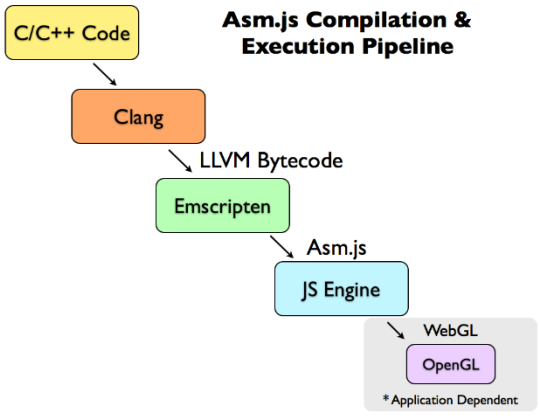


Obrázek 3 - Otevření výsledné stránky

Z měření vyplívá, že průměrné zpomalení WebAssembly oproti nativnímu kódu je 1,43 pro Firefox a 1,92 pro Chrome [45]. Pokud budeme srovnávat WebAssembly a asm.js zjistíme, že WebAssembly je 1,68x rychlejší v Chromu a 11,71x rychlejší ve Firefoxu [44].

1.4 Asm.js

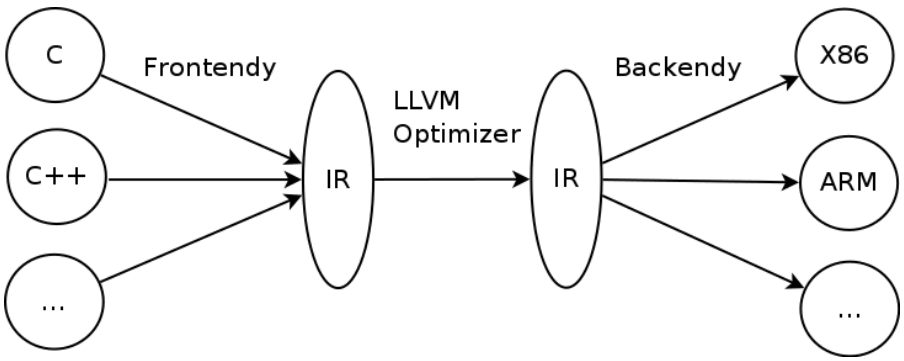
Asm.js je podmnožina programovacího jazyka JavaScript. Je určena pro zrychlení běhu JavaScriptových aplikací. Asm.js je jen vysoce optimalizovaný JavaScript, tudíž není potřeba žádný plugin nebo funkce pro jeho spuštění [9]. Často se používá na spouští her v prohlížeči [41]. Prohlížeče, které podporují asm.js, jsou v současné době Mozilla Firefox, Google Chrome, Opera a Microsoft Edge [42]. Na Obrázku číslo 4 je zobrazen proces kompilace C/C++ kódu do Asm.js.



Obrázek 4 – Proces kompilace C/C++ kódu na asm.js[10]

1.5 LLVM

LLVM je sada nástrojů pro optimalizaci během kompilace (viz Obrázek číslo 6) a generování nativního kódu. Základním prvkem LLVM je Intermediate Representation (IR), tedy nízko úrovňový programovací jazyk podobný jazyku symbolických adres. Přestože první frontend byl implementován pro C/C++, LLVM podnítil vznik široké škály frontendů, například Swift, Rust, Haskell, Python a další. [11]



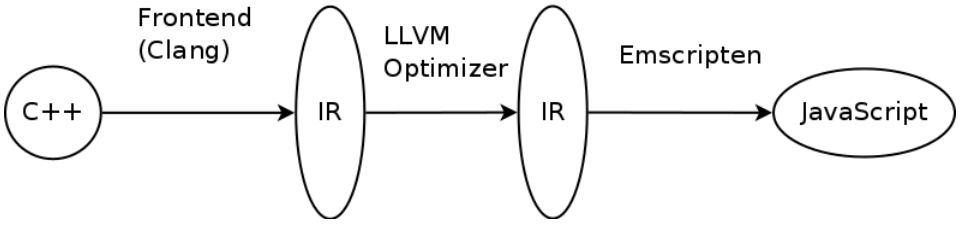
Obrázek 5 - Kompilace jazyků do binární formy pomocí LLVM [12]

Frontend je komponenta, která převádí původní jazyk do Intermediate Representation (IR). Optimizer optimalizuje IR výstup frontendu. Backend převádí IR výstup optimizéru na nativní kód. [12]

1.6 Emscripten

Emscripten je překladač, který běží jako backend pro LLVM kompilátor (viz Obrázek číslo 7) a vytváří podmnožinu JavaScriptu známou jako asm.js nebo WebAssembly. To umožňuje například vzít program napsaný v c++, přeložit do JavaScriptu a spustit v prohlížeči (viz Obrázek číslo 12). Emscriptem dokáže [15]:

- Přeložit C a C++ kód do JavaScriptu.
- Přeložit jakýkoli kód, který může být přeložen do LLVM bitového kódu, do JavaScriptu.
- Přeložit C a C++ běhové prostředí jiných jazyků do JavaScriptu, a pak spouštět kódy v jiných jazycích na nepřímo (například pro Python a Lua).



Obrázek 6 - Kompilace jazyků do JavaScriptu pomocí Emscripten [12]

2 FRAMEWORK BLAZOR

Framework Blazor je webový framework založený na technologiích C#, Razor a HTML. Blazor umožňuje webovým vývojářům psát webové aplikace založené na .NET, které běží na straně uživatele ve webových prohlížečích [17]. Blazor je vyvíjen pod licencí Apache 2.0 [18] a stále je v experimentální fázi. Produkční verze server-side má být dostupná v roce 2019 společně s .NET Core 3 a client-side Blazor má být dostupný v rámci pozdějších verzích .NET Core 3.0 [19].

2.1 Historie

Blazor začal jako osobní projekt Steva Sandersona z Microsoftu. V roce 2017 Steve ukázal Blazor na konferenci NDC Oslo. Tahle první verze byla postavena na interpreteru .NET CIL (Common intermediate Language) běhového prostředí s názvem DotNetAnywhere. Zatímco možnosti Blazoru byly omezené, potenciál byl zřejmý. Od tohoto dema Microsoft přidal Blazor do jejich ASP.NET GitHub organizace jako experimentální projekt. Jako součást oficiálního přijetí, byl Blazor přepsán od nuly týmem ASP.NET. DotNetAnywhere byl nahrazen Mono, který má mnohem pokročilejší a plně funkční nabídku z hlediska .NET běhového prostředí. Mono je součástí společnosti Microsoft od roku 2016. V současné době Mono využívá například Xamarin framework nebo Unity herní engine. [16]

2.2 JavaScript interoperabilita

Blazor umožňuje pracovat přímo s JavaScriptem přes JavaScript interop. Jednotlivé komponenty jsou schopné pracovat se všemi knihovnami nebo API, se kterými dokáže pracovat JavaScript. C# kód může volat JavaScript kód. [21]

Pro volání JavaScriptu z .NET je potřeba použít IJSRuntime abstrakci. Metodou InvokeAsync<T> se volají JavaScript funkce. V této metodě se specifikuje identifikátor funkce a libovolný počet argumentů serializovatelných pomocí protokolu JSON. Identifikátor funkce je relativní ke globálnímu rozsahu window. Není potřeba registrovat funkci před jejím voláním. Návratová hodnota T musí být taky serializovatelná pomocí JSON. [21]

Následující ukázka je založena na experimentálním JavaScript dekodéru TextDecoder [20]. Ukázka znázorňuje, jak volat JavaScript funkci z C# metody. Do JavaScript funkce vstupuje bitové pole ze C# metody, dekóduje bitové pole a vrátí text do komponenty pro zobrazení. [21]

V elementu <head> v souboru wwwroot/index.html poskytneme funkci, která používá TextDecoder.

```
<script>
  window.ConvertArray = (win1251Array) => {
    var win1251decoder = new TextDecoder('windows-1251');
    var bytes = new Uint8Array(win1251Array);
    var decodedArray = win1251decoder.decode(bytes);
    console.log(decodedArray);
    return decodedArray;
  };
</script>
```

JavaScript kód se může načíst i přímo ze souboru. V takovém případě stačí přidat odkaz na soubor.

```
<script src="exampleJsInterop.js"></script>
```

Následuje popis komponenty, která volá při stisku tlačítka ConvertArray JavaScript funkci pomocí JsRuntime. Po stisknutí tlačítka se bitové pole převede na formát string a zobrazí na stránce.

Nejprve je potřeba vložit IJSRuntime:

```
@page "/"
@inject IJSRuntime JsRuntime;
```

Komponenta obsahuje:

- Nadpis popisující, co tahle komponenta dělá.
- Tlačítko Convert Array, které volá funkci pro převod pole.
- Odstavec pro výsledný text.

```
<h1>Call JavaScript Function Example</h1>

<button type="button" class="btn btn-primary" onclick="@ConvertArray">
    Convert Array
</button>

<p class="mt-2" style="font-size:1.6em">
    <span class="badge badge-success">
        @ConvertedText
    </span>
</p>
```

Nakonec je potřeba deklarovat C# funkci ConvertArray, která zavolá JavaScript funkci ConvertArray a předá výsledek do proměnné ConvertedText.

```
@functions {

    private MarkupString ConvertedText =
        new MarkupString("Select the <b>Convert Array</b> button.");

    private uint[] QuoteArray = new uint[]
        {
            60, 101, 109, 62, 72, 101, 108, 108, 111, 32, 87, 111, 114, 108,
            100, 60, 47, 101, 109, 62
        };

    private async void ConvertArray()
    {
        var text =
            await JsRuntime.InvokeAsync<string>("ConvertArray", QuoteArray);

        ConvertedText = new MarkupString(text);

        StateHasChanged();
    }
}
```

Po zavolání komponenty a stisknutí tlačítka se nám zobrazí přeložený text Hello Word.

Pro volání .NET metody z JavaScriptu je potřeba použít DotNet.invokeMethod nebo DotNet.invokeMethodAsync funkce. Tyto funkce potřebují identifikátor metody, název assembly obsahující funkci a jakékoli argumenty, které potřebujeme. Asynchronní verze se preferuje pro podporu server-side případů. Pro to, aby se metoda dala volat z JavaScriptu, musí být veřejná, statická a označena atributem [JSInvokable]. Ve výchozím stavu identifikátor metody je její název, ale dá se změnit pomocí atributu JSIvokableAttribute. Volání obecných metod není v současné době podporované.

Následující ukázka obsahuje C# metodu, která vrací pole čísel a její volání z JavaScriptu. Metoda je označena atributem JSInvokable.

Pages/JsInterop.razor:

```
<button type="button" class="btn btn-primary"
        onclick="exampleJsFunctions.returnArrayAsyncJs()">
    Trigger .NET static method ReturnArrayAsync
</button>

@functions {
    [JSInvokable]
    public static Task<int[]> ReturnArrayAsync()
    {
        return Task.FromResult(new int[] { 1, 2, 3 });
    }
}
```

wwwroot/exampleJsInterop.js:

```
window.exampleJsFunctions = {
  showPrompt: function (text) {
    return prompt(text, 'Type your name here');
  },
  displayWelcome: function (welcomeMessage) {
    document.getElementById('welcome').innerText = welcomeMessage;
  },
  returnArrayAsyncJs: function () {
    DotNet.invokeMethodAsync('BlazorSample', 'ReturnArrayAsync')
      .then(data => {
        data.push(4);
          console.log(data);
    });
  },
  sayHello: function (dotnetHelper) {
    return dotnetHelper.invokeMethodAsync('SayHello')
      .then(r => console.log(r));
  }
};
```

Po kliknutí na tlačítko se zavolá JavaScript funkce returnArrayAsyncJs, která volá C# funkci ReturnArrayAsync. Po otevření konzole v prohlížeči lze vidět, že nám funkce vrací pole se čtyřmi čísly.

```
Array(4) [ 1, 2, 3, 4 ]
```

2.3 Ladění programů

V době psaní této bakalářské práce je ladění programů zatím ve vývoji, ale existuje předčasná podpora ladění programů client-side Blazor aplikací v Chromu. Podporované funkce jsou [22]:

- Přidání a odebírání breakpointů.
- Krokování kódu nebo pokračování běhu kódu.
- V záložce s lokálníma proměnnýma lze pozorovat lokální proměnné typu int, string a bool.
- V záložce zásobníku volání (Call stack) jsou zobrazeny veškeré volání, které probíhají mezi Javascript a .NET.

Mezi funkce, které zatím nejsou podporované, patří:

- Pozorování hodnot lokálních proměnných, které nejsou int, string nebo bool.
- Pozorování hodnot třídních proměnných.
- Přejíždět myší nad proměnnými k zobrazení jejich hodnot.

- Vyhodnocování výrazů v konzoli.
- Krokování přes asynchronní volání.

K ladění client-side Blazor aplikace v prohlížeči Chrome je zapotřebí:

- Spustit Blazor aplikaci v Debug konfiguraci
- Spustit Blazor aplikaci v Chrome (verze 70 nebo vyšší)
- Stisknout kombinaci na klávesnici Shift+Alt+D na Windows nebo Shift+Cmd+D na macOS

2.4 Optimalizace

Kritickým faktorem použitelnosti aplikace je velikost souborů, které musí prohlížeč stáhnout. Blazor pro client-side aplikace optimalizuje tuhle velikost těmito způsoby [23]:

- Nepoužívané části .Net assemblies jsou odstraněny během sestavování aplikace.
- HTTP odpovědi se komprimují.
- .NET běhové prostředí a assemblies jsou ukládány do vyrovnávací paměti prohlížeče.

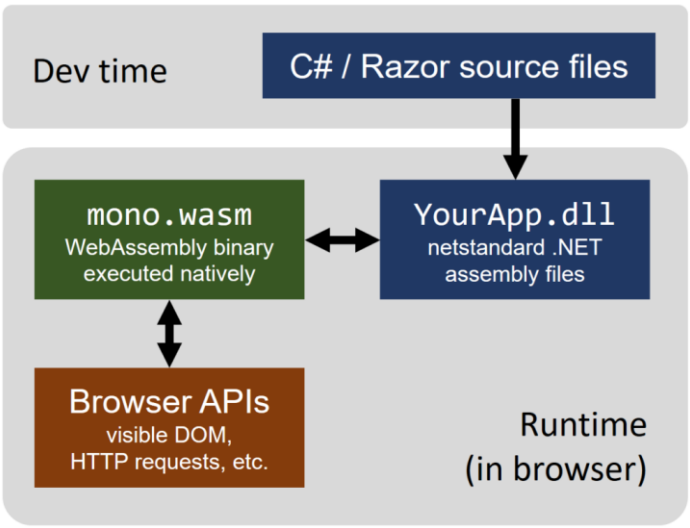
2.5 Mono

Mono je open source implementace .NET frameworku založeného na ECMA standardech pro C# a Common Language Runtime (CLR) [24]. Na obrázku číslo 7 jde vidět logo Mona. V roce 2017 Mono tým publikoval svoje první výsledky snažení, dostat Mono a s ním i C#, CLR a .NET Framework do WebAssembly [26]. Blazor aplikace neobsahují běhové prostředí, to zde přináší Mono. Mono má za cíl 2 typy běhu s WebAssembly, interpretovaný (Interpreted) a předběžný (AOT). [25]

Obrázek 7 – Mono logo [24]

2.5.1 Interpreted

Mono běhové prostředí je kompilované do WebAssembly, které pak následně je načtené a spuštěné prohlížečem. Knihovny Blazor aplikace, které jsou obyčejné standardní .NET knihovny, můžou být pak načteny a spuštěny pomocí Mono běhového prostředí. Tento přístup se zdá být nejrychlejší z pohledu rychlosti vývoje, protože znovu načtení aplikačních knihoven je velmi rychlé [25]. Na Obrázku číslo 9 jde vidět proces interpreted módu.

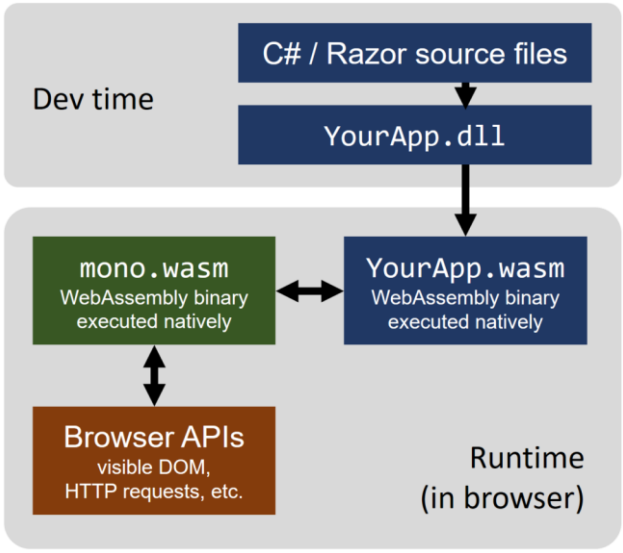


Obrázek 8 - Interpreted mód [25]

2.5.2 AOT

V AOT přístupu Blazor aplikace je zkompilována přímo do WebAssembly už při jejím sestavení. Části Mono runtime jsou pořád načteny prohlížečem pro řízení nízko úrovňových

operací jako například garbage collection. V podstatě aplikace je spuštěna jako regulární WebAssembly [25]. Na Obrázku číslo 10 jde vidět proces AOT módu.



Obrázek 9 – AOT mód [25]

2.6 Razor

Razor je ASP.NET programovací syntaxe použitá pro vytváření dynamických webových stránek s C# programovacím jazykem. Ve stránce, která používá Razor syntaxi, jsou bloky: uživatelský obsah a serverový kód. Uživatelský obsah jsou věci, které jsou obvykle na webových stránkách, jako například HTML značky, CSS stylování nebo čistý text. Syntaxe Razor nám dovoluje přidat serverový kód k tomuto uživatelskému obsahu. Pokud je na stránce nějaký serverový kód, tak se nejprve vyhodnotí serverový kód na serveru a až pak se pošle stránka do prohlížeče. Tím, že je kód vyhodnocován na serveru, můžeme vyhodnocovat daleko komplexnější věci, jako například dotazování serverových databázích. Velkou výhodou vyhodnocování kódu na serveru je to, že můžeme dynamicky vytvářet uživatelský obsah. Z pohledu prohlížeče se generovaný uživatelský obsah ze serverového kódu nijak neliší od jiného uživatelského obsahu. [27]

ASP.NET webové stránky, které obsahují Razor syntaxi, mají speciální souborovou koncovku .cshtml. Server rozpozná koncovku souboru, spustí kód s Razor syntaxí a pošle celou stránku prohlížeči. [27]

Ukázka Razor syntaxe:

```
<tbody>
    @foreach (var forecast in forecasts)
    {
        <tr>
            <td>@forecast.Date.ToShortDateString()</td>
            <td>@forecast.TemperatureC</td>
            <td>@forecast.TemperatureF</td>
            <td>@forecast.Summary</td>
        </tr>
    }
</tbody>
```

2.6.1 Razor components

Ve většině SPA(Single Page Application) frameworkách jsou aplikace tvořeny komponenty. Komponenta většinou představuje část UI, jako například stránky, dialogy, tabulky, nebo formuláře. Komponenty můžou být zanořeny do sebe, znovupoužity a sdíleny skrz projekty. [25]

V Blazoru komponenta je .NET třída, kterou je možné napsat jako C# třídu, nebo jako Razor stránku s koncovkou .razor. Celá aplikace Blazoru se dá považovat jako jedna velká komponenta [25]. Ukázka komponenty:

```
<h1>Counter</h1>

<p>Current count: @currentCount</p>

<button class="btn btn-primary" onclick="@IncrementCount">Click me</button>

@functions {
    int currentCount = 0;

    void IncrementCount()
    {
        currentCount++;
    }
}
```

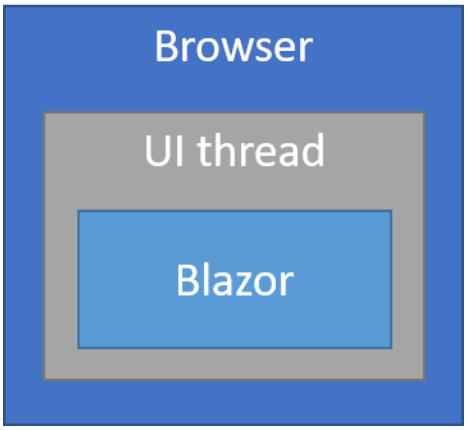
2.7 Možnosti nasazení

Máme dvě možnosti, jak nasadit Blazor aplikaci na web. Na straně klienta tzv. client-side a na straně serveru tzv. server-side. Každá možnost má své klady i zápory. [28]

2.7.1 Client-side

Hlavní model hostovaní Blazoru je spouštění na straně klienta v prohlížeči za pomocí WebAssembly. Blazor aplikace, její závislosti a .NET běhové prostředí jsou staženy přímo do prohlížeče. Aplikace je spuštěna v UI vláknu prohlížeče (viz Obrázek číslo 11). UI obnovuje

a obsluhuje události vyskytnuté ve stejném procesu. Prostředky aplikace jsou vystaveny jako statické soubory web serveru nebo jako služba schopna obsluhy statického obsahu uživatelům. [28]



Obrázek 10 – Client-side

Pro vytvoření Blazor aplikace s použitím client-side modelu je potřeba použít jeden z následujících šablon:

- Blazor client-side – Vystaveno jako sada statických souborů.
- Blazor (ASP.NET Core Hosted) – Hostováno na ASP.Net Core serveru. ASP .NET Core aplikace předává data Blazor aplikaci. Client-side Blazor aplikace může komunikovat se serverem přes síť pomocí API volání nebo SignalIR.

Šablony obsahují components.webassembly.js skript, který obstarává:

- Stahování .NET běhového prostředí, aplikace a aplikačních závislostí .
- Inicializaci běhového prostředí pro běh aplikace.

Klady client-side hostování:

- Nemá žádné .NET server-side závislosti.
- Plně využívá klientských schopností a zdrojů.
- Přenechává veškerou práci serveru klientovi.
- Podporuje běh v offline režimu.

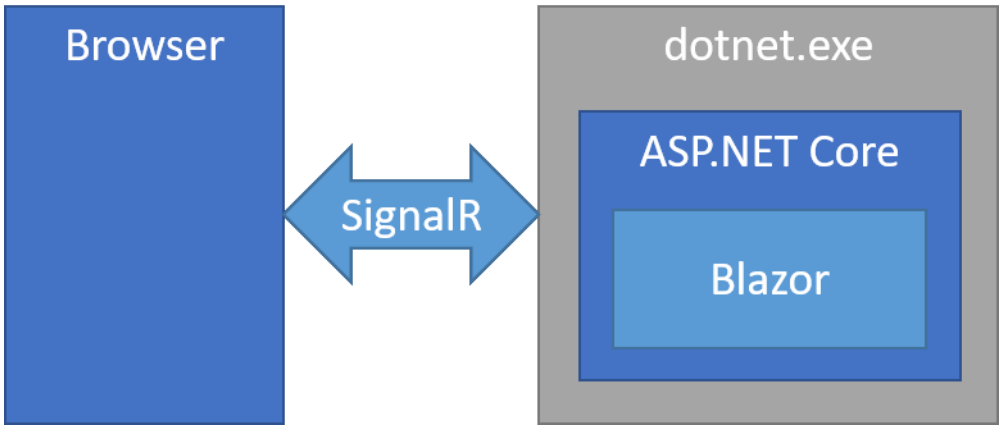
Zápory client-side hostování:

- Omezuje aplikace na možnosti prohlížeče.
- Vyžaduje schopný klientský hardware a software (například podporu WebAssembly)

- Má velký obsah dat nutných ke stažení do prohlížeče a delší načítání aplikace.
- Má méně vyspělé .NET běhové prostředí a menší podporu nástrojů (například omezení v podpoře .NET Standardu a v ladění)

2.7.2 Server-side

U modelu server-side je aplikace spuštěna na serveru zevnitř ASP .NET Core aplikace. Aktualizace UI, obsluha událostí a Javascript volání jsou obstarávány skrz SignalIR spojení (viz Obrázek číslo 12). [28]



Obrázek 11 – Server-side [28]

Pro vytvoření Razor Components aplikace pomocí server-side modelu, je potřeba použít Blazor server-side šablonu. Aplikace ASP .NET Core hostuje Razor Components server-side aplikaci a nastavuje endpoint, na který se klient připojí. Components.server.js skript stanovuje klientské spojení s aplikací. Aplikace má odpovědnost za udržování a obnovování aplikačního stavu (například při ztrátě připojení k síti). [28]

Klady server-side modelu:

- Má menší velikost aplikace a načítá se rychleji.
- Plně může využívat serverových schopností, například použití jakékoli API kompatibilní s API .NET Core.
- Aplikace běží na serveru .NET Core, tudíž není problém s laděním.
- Dokáže pracovat s prohlížeči, které neobsahují podporu WebAssembly.

Zápory server-side modelu:

- Má větší latenci. Každá uživatelská interakce zahrnuje dotaz přes síť.

- Nenabízí žádnou offline podporu. Pokud selže spojení s klientem, aplikace přestane fungovat.
- Má sníženou škálovatelnost. Server musí řídit několik klientských připojení a spravovat stav klientů.
- Vystavení aplikace bez serveru není možné

2.8 Jazyk C#

Jazyk C# je moderní objektově orientovaný programovací jazyk, který vychází z programovacích jazyků C/C++/Java. Překladače jazyka rozlišují velká a malá písmena tak zvaně jsou case sensitive. Základními charakteristiky jazyka jsou [29]:

- Obsahuje dědičnost s možností násobné implementace rozhraní.
- Vedle členských dat a metod přidává vlastnosti a události.
- Správa paměti je automatická. O správné uvolňování paměti se stará proces garbage collector.
- Podporuje zpracování chyb pomocí výjimek.
- Jazyk C# je čistě objektově orientovaný

2.8.1 Historie

Jazyk C# byl vyvinutý firmou Microsoft vedenou Ander Hejlsbergem a jeho týmem ze skupiny .Net initiative v roce 2002 [32]. V roce 2003 byl schválen jako standard evropskou neziskovou organizací ECMA (ECMA-334) a ISO (ISO/IEC 23270:2003) [31]. V době psaní bakalářské práce se používá C# 7.3 ale má vyjít verze 8.0 společně s .NET Core 3 [30].

2.9 .NET

.Net je bezplatná vývojářská platforma pro vytváření aplikací pro web, mobil, desktop nebo IoT. Aplikace .NET se dají psát v jazycích C#, F# nebo Visual Basic [40]. Mezi základní funkce patří:

- Podpora více programovacích jazyků.
- Asynchronní a souběžné modely programování
- Automatické zpravování paměti
- Velká podpora XML

2.9.1 .NET Standard

.NET standard je soubor rozhraní, které jsou implementovány v knihovně Base Class v .NET implementaci. Lépe řečeno je to formální specifikace .NET rozhraní, proti kterým se sestavuje uživatelský kód. Tyto rozhraní jsou implementovány v každé .NET implementaci. Tohle umožňuje přenosnost napříč různými .NET implementacemi, což umožňuje spouštět uživatelský kód kdekoli. .Net Standard je také tzv. target framework. Pokud uživatelský kód cílí na určitou verzi .NET standardu, tak může být spuštěn na jakékoli .NET implementaci, která podporuje tuhle verzi standardu. [39]

2.9.2 .NET implementace

Každý implementace .NET obsahuje následující komponenty [39]:

- 1 nebo více běhových prostředí. Například CLR pro .NET Framework, CoreCLR a CoreRT pro .NET Core.
- Knihovnu tříd, která implementuje .NET Standard a může implementovat další aplikační rozhraní. Například knihovna .NET framework Base Class nebo knihovna .NET Core Base Class.
- Volitelně může obsahovat jeden nebo více aplikačních frameworků. Například ASP.NET, Windows Forms nebo Windows Presentations Foundation (WPF).
- Volitelně může obsahovat vývojářské nástroje. Některé vývojářské nástroje jsou sdíleny mezi několik implementací.

Firma Microsoft aktivně vyvíjí a udržuje 3 implementace .NET Core, .NET Framework, Mono.

2.9.3 .NET runtime

.NET runtime je běhové prostředí pro spravovaný program. Operační systém je část běhového prostředí ale není to část .NET běhového prostředí. Zde jsou několik příkladů .NET běhového prostředí [39]:

- Common Language Runtime (CLR) pro .NET Framework
- Core Common Language (CoreCLR) pro .NET Core
- .NET Native pro Universal Windows Platform
- Běhové prostředí Mono pro Xamarin.iOS, Xamarin.Android, Xamarin.Mac a Mono desktop framework

2.10 JavaScript

JavaScript je webový objektově orientovaný skriptovací jazyk. Standardizovaná verze JavaScriptu se jmenuje ECMAScript, ze které byly odvozeny další implementace, jako například ActionScript. V dnešní době dosáhl vývoj jazyka do bodu, kdy se používá nejen k vytváření webových stránek, ale i mobilních a desktopových aplikací. Každý prohlížeč má svůj vlastní JavaScript interpreter [34]. Nejznámější interpretery jsou:

- Chakra – Od společnosti Microsoft pro prohlížeč Microsoft Edge [37]
- SpiderMonkey – Původně napsaný samotným Brendanem Eichem, v dnešní době udržovaný společností Mozilla Foundation. Využitý v prohlížeči Mozilla Firefox [38].
- Chrome V8 – Vytvořenou skupinou The Chromium Project pro prohlížeč Chrome [36].
- JavaScriptCore – Vytvořen firmou Apple pro prohlížeč Safari [35].

2.10.1 Historie

Javascript byl vytvořen v roce 1995 Brendanem Eichem během toho, co pracoval ve společnosti Netscape Communications. Úplně první pracovní název dnešního JavaScriptu byl Mocha, později byl přejmenován na LiveScript a ještě později byl přejmenován na dnešní JavaScript. V roce 1996 Javascript se stal tak populární, že byl předán asociaci ECMA, která je zodpovědná za vývoj a udržování tohoto jazyka do dnes. Výsledkem tohoto kroku byl standardizovaný skriptovací jazyk ECMAScript, kterému se stále říká JavaScript. V dnešní době je JavaScript všude, je to nejpoužívanější client-side skriptovací jazyk. [33]

II. PRAKTICKÁ ČÁST

3 VYTVOŘENÍ PROJEKTU

V době implementace (duben 2019) framework Blazor nabízel dvě šablony pro client-side aplikaci, Blazor client-side a Blazor ASP .NET Core hosted. Jelikož autor chtěl mít rozdělený client-side a server-side, rozhodl se použít Blazor ASP .NET Core hosted.

Před vytvořením projektu bylo potřeba splnit několik požadavku. Nejprve bylo potřeba nainstalovat nejnovější verzi .NET Core 3.0, který lze stáhnout ze stránek https://dotnet.microsoft.com/download/dotnet-core/3.0. Dalším požadavkem byla nutnost mít nainstalovanou Preview verzi Visual Studia 2019 s rozšířením vývoje pro ASP.NET a web. V době psaní bakalářské práce byla aktuální verze SDK 3.0.100-preview4-011223. Po nainstalování Visual studia se muselo v nastavení povolit použití preview verzí .NET Core SDK. Posledním požadavkem bylo nainstalování a povolení šablon Blazor. Nainstalování Blazor šablon bylo dosaženo spuštěním následujícího příkazu v příkazovém řádku:

```
dotnet new -i Microsoft.AspNetCore.Blazor.Templates::3.0.0-preview4-19216-03
```

Povolení šablon ve Visual Studiu bylo dosaženo stažením Blazor rozšíření.

4 ZÁKLADNÍ STRUKTURA

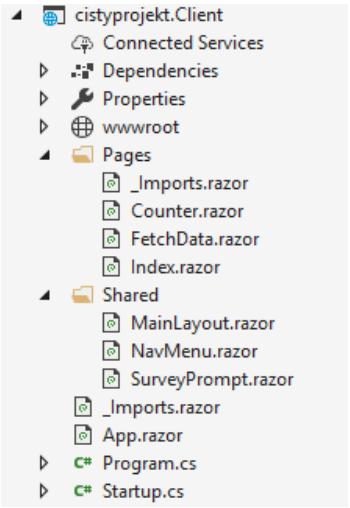
Nový projekt se skládá z tři podprojektů a to:

- Client
- Server
- Shared

4.1 Client

Tento projekt obsahuje client-side část aplikace. Nacházejí se tu jednotlivé stránky aplikace a jejich komponenty. Client projekt obsahuje (viz Obrázek číslo 13):

- /Wwwroot – Zde se nachází standardní webové prostředky jako například CSS, JavaScript, HTML, JSON soubory.
- /Pages – Složka obsahující Razor (.razor) aplikační stránky.
- /Shared – Složka obsahující společné Razor komponenty a layouts.
- Program.cs – Soubor, který nastavuje a spouští Blazor hostování aplikace.
- Startup.cs – Soubor, který obsahuje nastavení client projektu a služeb.
- App.razor – Je to jediný soubor, který je specifický pro Blazor. Blazor je dodáván s client-side routerem a tahle stránka ho povoluje pro danou aplikaci.

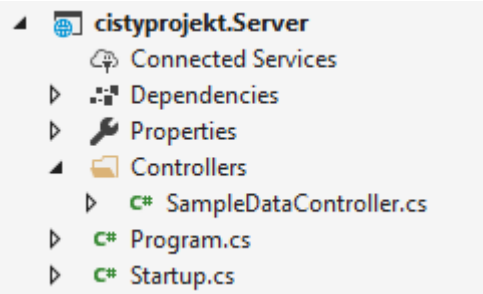


Obrázek 12 – Struktura projektu Client

4.2 Server

Tento projekt obsahuje server-side část aplikace. Nacházejí se tu Web API controllery, které poskytují data Client projektu. Server projekt obsahuje (viz Obrázek číslo 14):

- /Controllers – Složka obsahující web API controllery.
- Program.cs - Soubor, který nastavuje a spouští ASP .NET Core hostování.
- Startup.cs - Soubor, který obsahuje nastavení projektu a služeb.



Obrázek 13 – Struktura projektu Server

4.3 Shared

Tento projekt obsahuje modely, které jsou používány v Client a Server projektu. Instalují se zde i všechny balíčky, které se používají v obou projektech.

5 UKÁZKA PRÁCE S DATABÁZÍ

Tato kapitola se zabývá vytvořením tabulky s databázi a demonstruje vytvoření, čtení, aktualizaci, smazání řádků tabulky. Při vytváření řádku tabulky se demonstruje také validace vstupů formuláře.

5.1 Server-side

Tato podkapitola popisuje vytvoření Controlleru s metodami pro obsluhu dat tabulky a statické třídy, která udržuje data tabulky.

5.1.1 Products

V server projektu se vytvoří složka Database, kde budou data pro aplikaci. Ve složce se vytvoří statická třída Products.cs, která bude simulovat databázi. Ve statické třídě se vytvoří list produktů ProductList, ve kterém je list produktů pro CRUD tabulku.

```
public static List<Product> ProductList { get; set; }
```

Pro inicializaci dat se vytvoří funkce Init. Funkce Init vytvoří nový list produktů se čtyřmi předpřipravenými produkty, pokud je list hodnoty null.

```
public static void Init()
{
    if (ProductList == null)
    {
        ProductList = new List<Product>() {
            new Product()
            {
                Id = 1,
                Name = "Pomeranč",
                Price = 30
            },
            new Product()
            {
                Id = 2,
                Name = "Jablko",
                Price = 200
            },
            new Product()
            {
                Id = 3,
                Name = "Kiwi",
                Price = 10
            },
            new Product()
            {
                Id = 4,
                Name = "Hruška",
                Price = 50
            }
        };
```

```
    }
}
```

Funkce GetIdOfLastProduct vrátí Id posledního produktu v listu produktů. Pokud je list produktů prázdný vrátí se 0. Funkce GetProducts vrátí list produktů a volá funci Init pro inicializaci dat.

```
public static int GetIdOfLastProduct()
{
    if (ProductList.Count == 0)
        return 0;
    return ProductList[ProductList.Count - 1].Id;
}

public static List<Product> GetProducts()
{
    Init();
    return ProductList;
}
```

5.1.2 CrudTableDataController

Ve složce Controllers se vytvoří nová třída CrudTableDataController.cs, která dědí od třídy Controller. Atribut Route se nastaví na api/[controller], aby cesta k této třídě byla tvořena jejím názvem. Dále bylo potřeba implementovat jednotlivé metody třídy.

```
[HttpGet("[action]")]
public List<Product> GetProducts()
{
    return Products.GetProducts();
}

[HttpPost("[action]")]
public ActionResult AddProduct([FromBody] Product product)
{
    product.Id = Products.GetIdOfLastProduct() + 1;
    Products.ProductList.Add(product);
    return new EmptyResult();
}

[HttpPost("[action]")]
public ActionResult DeleteProduct([FromBody] int id)
{
    var itemToRemove = Products.ProductList.Single(r => r.Id == id);
    Products.ProductList.Remove(itemToRemove);
    return new EmptyResult();
}

[HttpPost("[action]")]
public ActionResult UpdateProduct([FromBody] Product product)
{
    Products.ProductList.Single(r => r.Id == product.Id).Name = product.Name;
    Products.ProductList.Single(r => r.Id == product.Id).Price = product.Price;
    return new EmptyResult();
}
```

Htttp get metoda GetProduct volá funkci Products.GetProducts, která vrací list všech produktů. Http post metoda AddProduct slouží k přidávání produktu do tabulky. Metoda obsahuje parametr product. Pomocí atributu [FromBody] je specifikováno, že parametr product je obsažen v těle post požadavku. Metoda nejprve zjistí poslední id z tabulky produktů, přičte k němu jedničku a uloží do product parametru. Následně přidá produkt z parametru do listu produktů. Http post metoda DeleteProduct odstraňuje produkt z listu podle id. Metoda vybere pomocí LINQ dotazu produkt s id z parametru funkce a odstraní ho z listu produktů. Http post metoda UpdateProduct aktualizuje název a cenu produktu podle parametru product za pomoci LINQ dotazu.

5.2 Client-side

Tato podkapitola se zabývá vytvořením CrudTable komponenty a stránky obsahující komponentu.

5.2.1 CrudTableComponent

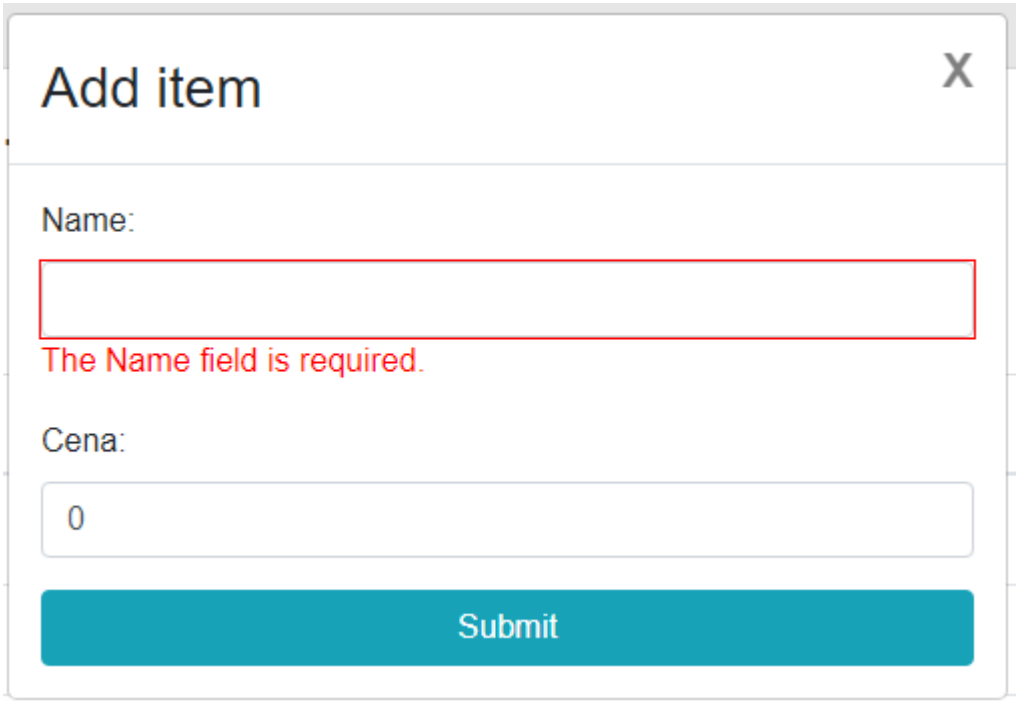
Ve složce Shared se vytvoří CrudTableComponent.razor komponenta. Jelikož je potřeba posílat dotazy na API, vloží se HttpClient do vytvořené komponenty. Vytvoří se Bootstrap HTML tabulka, kde obsah tabulky je dynamicky generován přes C# foreach cyklus. Každý cyklus se přidá jeden produkt z listu produktů prodList a vytvoří se dvě tlačítka pro funkce DeleteItemDBAsync a ShowUpdateModal. Pro přidání nového produktu do tabulky se vytvoří tlačítko napojené na funkci ShowAddModal. Celá tabulka s tlačítkem pro přidání produktu se vykreslí pouze za předpokladu, že prodList neobsahuje hodnotu null.

```
<div style="float:left">
    <button class="btn btn-primary" onclick="@ShowAddModal">Create</button>
</div>
<table class="table">
    <thead>
        <tr>
            <th>Název</th>
            <th>Cena</th>
            <th></th>
        </tr>
    </thead>
    <tbody>
        @foreach (var product in prodList)
        {
            <tr>
                <td>@product.Name</td>
                <td>@product.Price</td>
                <td>
```

```
                    <button onclick=@(() => DeleteItemDBAsync(product.Id))>
                        <span class="oi oi-circle-x" style="color:red"></span>
                    </button>
                    <button onclick=@(() => ShowUpdateModal(product))>
                        <span class="oi oi-justify-center" style="color:red"></span>
                    </button>
                </td>
            </tr>
        }
    </tbody>
</table>
```

Pro přidání a aktualizaci produktů se vytvoří Bootstrap dialog. Dialog se u přidání a aktualizace produktů liší pouze v počáteční podmínce pro zobrazení a funkce, která se volá při potvrzení dialogu. Pro validaci formuláře byla použita komponenta EditForm. Komponenta validuje formulář oproti validační logice v modelu product. Pokud položka formuláře neprojde validací, tak se chybobá zpráva vloží do odpovídající ValidationMessage komponenty (viz. Obrázek číslo 15).

```
@if (isAdd)
{
    <div class="modal" tabindex="-1" style="display:block" role="dialog">
        <div class="modal-dialog">
            <div class="modal-content">
                <div class="modal-header">
                    <h3 class="modal-title">Add item</h3>
                    <button type="button" class="close" onclick="@closeModal">
                        <span aria-hidden="true">X</span>
                    </button>
                </div>
                <div class="modal-body">
                    <EditForm Model="@product" OnValidSubmit="@AddItemDBAsync">
                        <DataAnnotationsValidator />
                        <p>
                            <label class="control-label" for="Name">Name: </label>
                            <InputText class="form-control" id="Name" bind-Va-
lue="@product.Name" />
                            <ValidationMessage For="@(() => @product.Name)" />
                        </p>
                        <p>
                            <label class="control-label" for="Price">Cena: </label>
                            <InputNumber class="form-control" id="Price" bind-Va-
lue="@product.Price" />
                            <ValidationMessage For="@(() => @product.Price)" />
                        </p>
                        <button class="btn btn-block btn-info" type="submit">Sub-
mit</button>
                    </EditForm>
                </div>
            </div>
        </div>
    </div>
}
```



Obrázek 14 – Validace v dialogu

Funkce OnInitAsync je asynchronní metoda, která se volá hned po inicializace komponenty. Funkce pošle HTTP get požadavek na endpoint api/CrudTableData/GetProducts a výsledek uloží do listu produktů prodList.

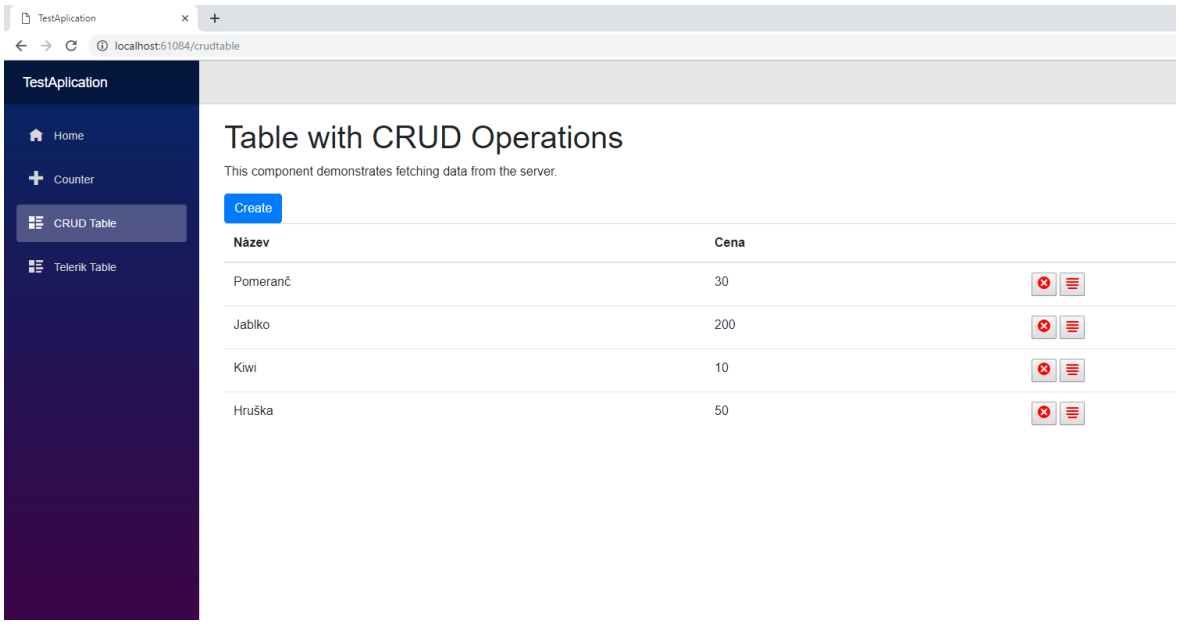
```
protected override async Task OnInitAsync()
{
    prodList = await Http.GetJsonAsync<List<Product>>("api/CrudTableData/GetProducts");
}
```

Funkce AddItemDBAsync je asynchronní metoda, která přidává produkt do databáze. Volá se po potvrzení dialogu přidání produktu. Metoda pošle HTTP post požadavek s produktem k přidání na endpoint api/CrudTableData/AddProduct. Po přidání se znova uloží do prodList aktuální list produktů a zavře se modal zavoláním funkce closeModal. Funkce DeleteItemDBAsync a UpdateItemDBAsync funguje na podobném principu, ale rozdíl je v endpointu a přidání přepisování aktualizovaného produktu. Na Obrázku číslo 16 lze vidět výslednou tabulku.

```
protected async Task AddItemDBAsync()
{
    await Http.SendJsonAsync(HttpMethod.Post, "api/CrudTableData/AddProduct", product);
    prodList = await Http.GetJsonAsync<List<Product>>("api/CrudTableData/GetProducts");
    closeModal();
}
```

```
protected async Task UpdateItemDBAsync()
{
    product.Id = productIDToUpdate;
    await Http.SendJsonAsync(HttpMethod.Post, "api/CrudTableData/UpdateProduct",
product);
    prodList = await Http.GetJsonAsync<List<Product>>("api/CrudTableData/GetProdu-
cts");
    closeModal();
}

protected async Task DeleteItemDBAsync(int id)
{
    await Http.SendJsonAsync(HttpMethod.Post, "api/CrudTableData/DeleteProduct",
id);
    prodList = await Http.GetJsonAsync<List<Product>>("api/CrudTableData/GetProdu-
cts");
    closeModal();
}
```



Obrázek 15 – Výsledná tabulka s funkcemi CRUD

5.3 Shared

Do Shared se vytvoří třída Product, která bude použita v client-side i server-side části aplikace. Třídá produkt obsahuje položky Id, Name a Price. Položka Id nemá žádnou validační logiku, protože slouží jenom k identifikaci položky. Položka Name bude vyžadována k vyplnění při validaci a zároveň nesmí mít víc jak 15 znaků. Položka Price bude vyžadována při validaci a zároveň musí být kladné číslo.

```
public class Product
{
    public int Id { get; set; }
    [Required]
    [StringLength(15, ErrorMessage = "Name is too long.")]
    public string Name { get; set; }
```

```
    [Required]
    [Range(0, Int32.MaxValue, ErrorMessage = "Price must be a positive number")]
    public int Price { get; set; }
}
```

6 TELERIK KOMPONENTY

Tato kapitola demonstruje vytvoření tabulky, která obsahuje stránkování, řazení a vyhledávání prvku tabulky pomocí Telerik UI komponent. Telerik UI je sada komponent, které nabízí firma Progress pro client-side i server-side Blazor [43].

6.1 Nastavení projektu

Nejdříve je potřeba si nastavit privátní Telerik NuGet kanál podle stránek vývojářů. Po nastavení NuGet kanálu se nainstaluje Telerik.UI.for.Blazor NuGet balíček do Client projektu pomocí NuGet správce balíčků. Dalším krokem je nastavení stylu komponent. Telerik nabízí default, bootstrap a materiál styl komponent. Pro nastavení stylu je potřeba vložit odkaz na styl do souboru /wwwroot/index.html.

```
<link id="kendoCss" rel="stylesheet" href="https://unpkg.com/@progress/kendo-theme-default@latest/dist/all.css" />
```

Do stejného souboru je nutné přidat JavaScriptový soubor telerik-blazor.min.js:

```
<script src="https://kendo.cdn.telerik.com/blazor/1.0.0/telerik-blazor.min.js" defer></script>
```

V době psaní bakalářské práce je problém v Monu a je potřeba vypnout IL linker. Pro vypnutí IL linkeru se přidá následující kód do souboru client.csproj.

```
<BlazorLinkOnBuild>false</BlazorLinkOnBuild>
```

Posledním krokem je registrace Telerik Blazor služby v Startup.cs souboru.

```
services.AddTelerikBlazor();
```

6.2 Vytvoření tabulky

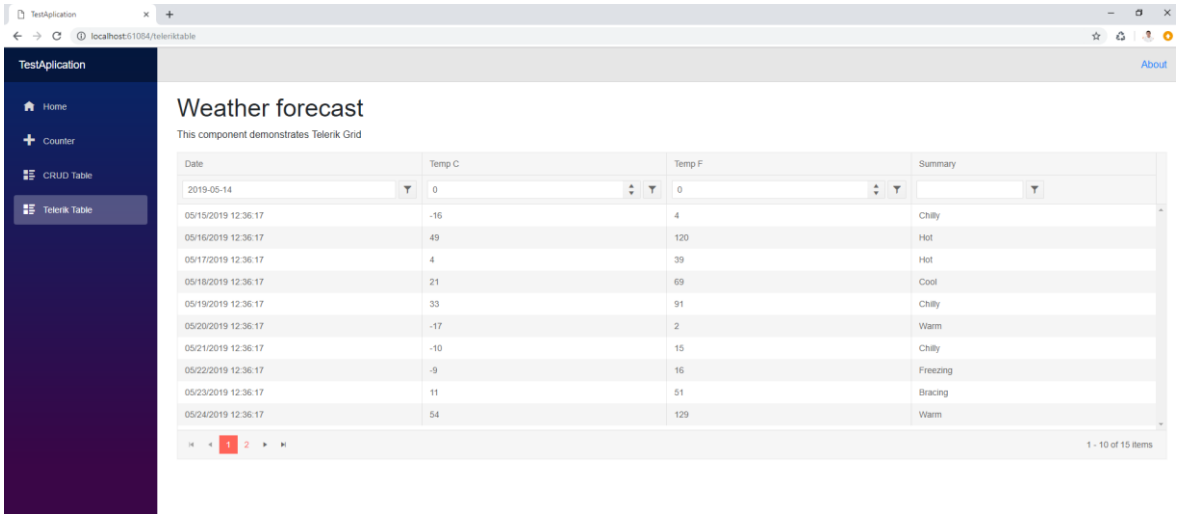
Pro získání dat tabulky se použije SampleDataController, který je obsažen v každém novém Blazor projektu. Vytvoří se nová stránka TelerikTable.razor, kde se vloží HttpClient na získání dat ze serveru. Vytvoří se pole WeatherForecast tříd s názvem forecasts a přidá se funkce OnInitAsync, která uloží data ze serveru do pole forecasts.

```
WeatherForecast[] forecasts;

protected override async Task OnInitAsync()
{
    forecasts = await Http.GetJsonAsync<WeatherForecast[]>("api/SampleData/WeatherForecasts");
}
```

Vytvoří se komponenta TelerikGrid s parametry Pageable, Sortable a Filterable na hodnotě true. V komponentě se specifikuje zdroj dat pro tabulku na pole forecasts a specifikují se jednotlivé sloupce tabulky. Na Obrázku číslo 17 je vidět výsledná tabulka.

```
@if (forecasts == null)
{
    <p><em>Loading...</em></p>
}
else
{
    <TelerikGrid Data="forecasts" Pageable="true" Sortable="true" Filterable="true">
        <TelerikGridColumns>
            <TelerikGridColumn Field="Date" Title="Date" />
            <TelerikGridColumn Field="TemperatureC" Title="Temp C" />
            <TelerikGridColumn Field="TemperatureF" Title="Temp F" />
            <TelerikGridColumn Field="Summary" Filterable="true" />
        </TelerikGridColumns>
    </TelerikGrid>
}
```



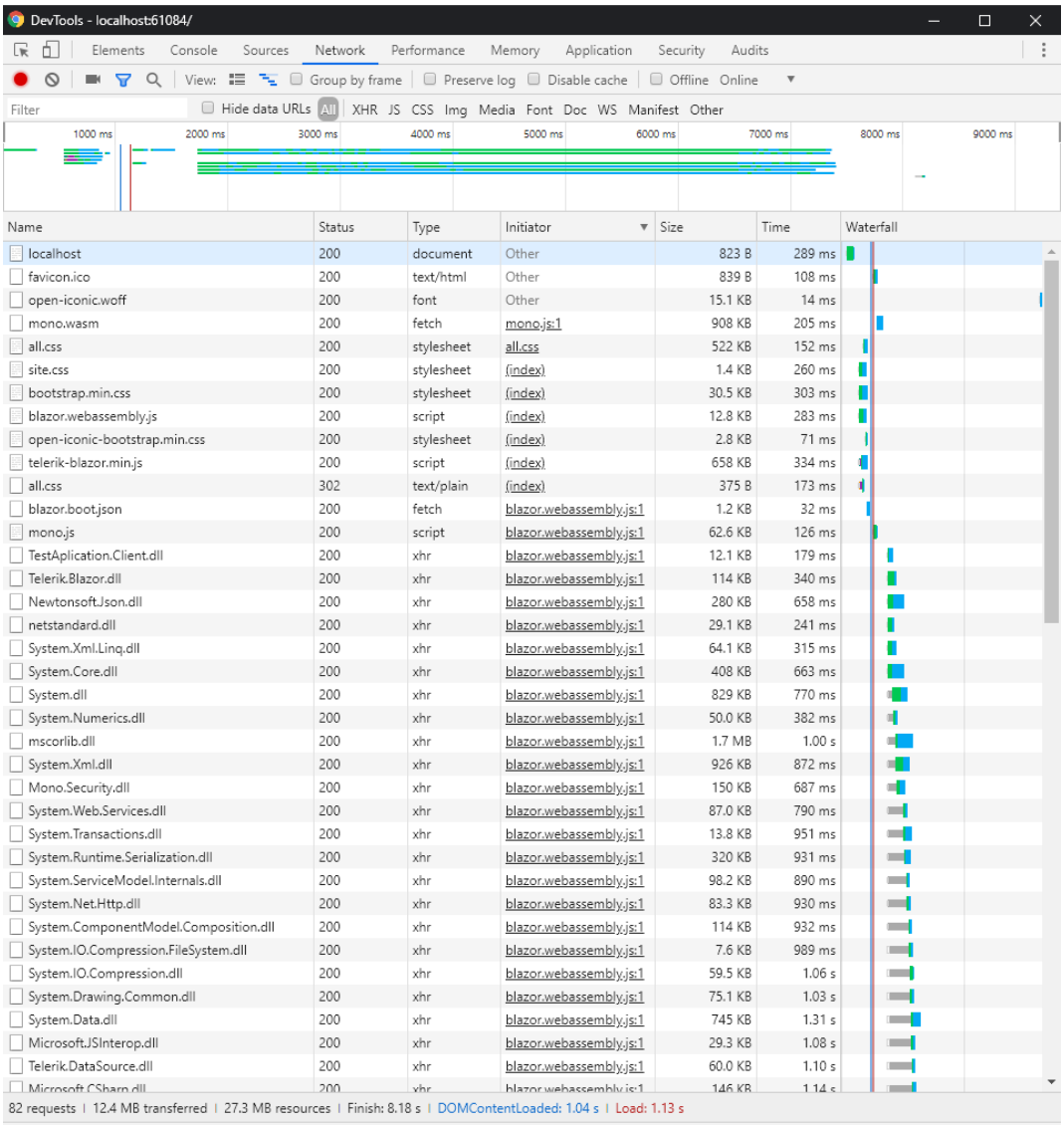
Obrázek 16 – Výsledná Telerik UI komponenta

7 VELIKOST PROSTŘEDKŮ STAHOVANÝCH DO PROHLÍŽEČE

Při otevření aplikace se do prohlížeče stáhnou:

- Html,Css a JavaScript soubory stránky
- Mono.wasm – Mono běhové prostředí přeložení do WebAssembly
- .NET knihovny aplikace

Při prvním navštívení autorovy webové aplikace bylo staženo do prohlížeče 12,4 MB prostředků (viz Obrázek číslo 17). Jelikož se většina prostředků ukládá do vyrovnávací paměti prohlížeče, byla po opakovaném navštívení stránky velikost prostředků jen 32,5 KB. Největší soubor, který se stahuje je mscorlib.dll. Jelikož se jedná o Preview verzi frameworku Blazoru, lze očekávat, že se velikost souborů bude měnit s lepší optimalizací kódu.



Obrázek 17 – Přenesené soubory do prohlížeče

ZÁVĚR

Výstupem praktické části je aplikace demonstrující možnosti webového frameworku Blazor. Výhodou frameworku Blazor je programování client-side i server-side části aplikace v jednotném jazyce C#. Čtenář by měl být schopen založit nový projekt ve webovém frameworku Blazor a vytvářet jednoduché aplikace.

V teoretické části bakalářské práce byl popsaný webový standard WebAssembly a jeho související technologie. Dále byl zde popsaný webový framework Blazor a jeho možnosti nasazení.

V praktické části bakalářské práce byl popsaný postup vytvoření nového ASP .NET Core Hosted projektu ve frameworku Blazor. Dále byla popsána základní struktura projektu, včetně detailního popisu jednotlivých podprojektů. Následoval popis postupu vytvoření tabulky s možností editace prvků, kde jednotlivé podkapitoly popisovaly postup vytvoření API na straně serveru a vytvoření komponenty na straně klienta. Další kapitola se zabývala popisem postupu vytvoření tabulky pomocí Telerik komponent. Nejdříve byl zde popsaný postup přidání Telerik NuGet balíčků do projektu a následně byl popsaný postup vytvoření TelerikGrid komponenty. Poslední kapitola se zabývala soubory, které se stahují do prohlížeče při přechodu na webovou stránku. Je zde popsána velikost souborů autorovy webové aplikace.

Jelikož v době psaní bakalářské nebyl vydán framework Blazor v produkční verzi, musel autor často upravovat strukturu projektu kvůli měnícím se Preview verzím Blazor frameworku. Dá se očekávat, že v budoucnu bude Blazor více optimalizovaný a bude implementována lepší podpora pro autentizace uživatele. Na základě zkušeností s vývojem ukázkové aplikace a podporou Blazoru ze strany dodavatelů komponent uživatelského rozhraní je pravděpodobné, že Blazor může v budoucnu z části nahradit JavaScript SPA aplikace.

SEZNAM POUŽITÉ LITERATURY

[1] *What is WebAssembly* [online]. [cit. 2019-04-25]. Dostupné z: https://www.infoworld.com/article/3291780/what-is-webassembly-the-next-generation-web-platform-explained.html

[2] *C to wasm* [online]. 2019 [cit. 2019-04-25]. Dostupné z: https://developer.mozilla.org/en-US/docs/WebAssembly/C_to_wasm

[3] Going public launch bug. *Github.com* [online]. 2015 [cit. 2019-04-29]. Dostupné z: https://github.com/WebAssembly/design/issues/150

[4] WAGNER, Luke. *A WebAssembly Milestone: Experimental Support in Multiple Browsers* [online]. 2016 [cit. 2019-04-29]. Dostupné z: https://hacks.mozilla.org/2016/03/a-webassembly-milestone/

[5] WebAssembly support now shipping in all major browsers [online]. 2017 [cit. 2019-04-29]. Dostupné z: https://blog.mozilla.org/blog/2017/11/13/webassembly-in-browsers/

[6] *WEBASSEMBLY FIRST PUBLIC WORKING DRAFTS* [online]. 2018 [cit. 2019-04-29]. Dostupné z: https://www.w3.org/blog/news/archives/6838

[7] *WebAssembly 1.0 has shipped in 4 major browser engines* [online]. [cit. 2019-04-29]. Dostupné z: https://webassembly.org/roadmap/

[8] *Text Format* [online]. [cit. 2019-04-29]. Dostupné z: https://webassembly.org/docs/text-format/

[9] MEHRABANI, Afshin. *Understanding asm.js* [online]. 2014 [cit. 2019-04-29]. Dostupné z: https://www.sitepoint.com/understanding-asm-js/

[10] RESIG, John. *Asm.js: The JavaScript Compile Target* [online]. 2013 [cit. 2019-04-30]. Dostupné z: https://johnresig.com/blog/asmjs-javascript-compile-target/

[11] YEGULALP, Serdar. *What is LLVM? The power behind Swift, Rust, Clang, and more* [online]. 2018 [cit. 2019-04-30]. Dostupné z: https://www.infoworld.com/article/3247799/what-is-llvm-the-power-behind-swift-rust-clang-and-more.html

[12] URBAN, Hynek. *WebAssembly a cesta k němu*[online]. 2015 [cit. 2019-05-15]. Dostupné z: https://fragaria.cz/blog/2015/07/23/webassembly-cesta-k-nemu/

[13] *Enscripten* [online]. 2015 [cit. 2019-04-30]. Dostupné z: https://emscripten.org/

[14] *Download and install* [online]. [cit. 2019-05-15]. Dostupné z: https://emscripten.org/docs/getting_started/downloads.html

[15] *About Emscripten* [online]. [cit. 2019-05-15]. Dostupné z: https://emscripten.org/docs/introducing_emscripten/about_emscripten.html

[16] *What is Blazor and why is it so exciting?* [online]. 2018 [cit. 2019-05-15]. Dostupné z: https://chrissainty.com/what-is-blazor-and-why-is-it-so-exciting/

[17] *Blazor: Build client web apps with C#* [online]. [cit. 2019-05-02]. Dostupné z: https://dotnet.microsoft.com/apps/aspnet/web-apps/client

[18] *Blazor: License* [online]. 2018 [cit. 2019-05-02]. Dostupné z: https://github.com/aspnet/Blazor/blob/master/LICENSE.txt

[19] *Blazor now in official preview!* [online]. 2019 [cit. 2019-05-02]. Dostupné z: https://devblogs.microsoft.com/aspnet/blazor-now-in-official-preview/

[20] *TextDecoder* [online]. [cit. 2019-05-02]. Dostupné z: https://developer.mozilla.org/cs/docs/Web/API/TextDecoder

[21] NELSON, Javier, Daniel ROTH a Luke LATHAM. *Blazor JavaScript interop* [online]. 2019 [cit. 2019-05-02]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/blazor/javascript-interop?view=aspnetcore-3.0

[22] ROTH, Daniel. *Debug Blazor* [online]. 2019 [cit. 2019-05-02]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/blazor/debug?view=aspnetcore-3.0

[23] ROTH, Daniel a Luke LATHAM. *Introduction to Blazor* [online]. 2019 [cit. 2019-05-02]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-3.0

[24] *Mono: Cross platform, open source .NET framework* [online]. 2019 [cit. 2019-05-02]. Dostupné z: https://www.mono-project.com/

[25] *Blazor: a technical introduction: Deeper technical details about Blazor* [online]. 2018 [cit. 2019-05-02]. Dostupné z: http://blog.stevensanderson.com/2018/02/06/blazor-intro/

[26] *Hello WebAssembly* [online]. 2017 [cit. 2019-05-02]. Dostupné z: https://www.mono-project.com/news/2017/08/09/hello-webassembly/

[27] FITZMACKEN, Tom. *Introduction to ASP.NET Web Programming Using the Razor Syntax (C#)* [online]. 2014 [cit. 2019-05-02]. Dostupné z:

https://docs.microsoft.com/en-us/aspnet/web-pages/overview/getting-started/introducing-razor-syntax-c

[28] ROTH, Daniel. *Blazor hosting models* [online]. 2019 [cit. 2019-05-03]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/blazor/hosting-models?view=aspnetcore-3.0

[29] THAKUR, DINESH. *FEATURES OF C#* [online]. [cit. 2019-05-05]. Dostupné z: http://ecomputernotes.com/csharp/cs/features-of-c

[30] ARH, Damir. *New C# 8 Features in Visual Studio 2019* [online]. 2019 [cit. 2019-05-05]. Dostupné z: https://www.dotnetcurry.com/csharp/1489/csharp-8-visual-studio-2019

[31] *Standard ECMA-334: C# Language Specification* [online]. [cit. 2019-05-05]. Dostupné z: https://www.ecma-international.org/publications/standards/Ecma-334.htm

[32] SRIVASTAVA, Bikesh. *History Of C# Programming Language* [online]. 2017 [cit. 2019-05-05]. Dostupné z: https://www.c-sharpcorner.com/blogs/history-of-c-sharp-programming-language

[33] RUBENS, Arden. *The History of JavaScript [INFOGRAPHIC]* [online]. 2018 [cit. 2019-05-05]. Dostupné z: https://www.checkmarx.com/blog/javascript-history-infographic/

[34] *JavaScript* [online]. [cit. 2019-05-05]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript

[35] *JavaScriptCore* [online]. [cit. 2019-05-05]. Dostupné z: https://developer.apple.com/documentation/javascriptcore

[36] *What is V8?* [online]. [cit. 2019-05-05]. Dostupné z: https://v8.dev/

[37] *ChakraCore* [online]. [cit. 2019-05-05]. Dostupné z: https://github.com/Microsoft/ChakraCore

[38] *SpiderMonkey* [online]. [cit. 2019-05-05]. Dostupné z: https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey

[39] *.NET architectural components* [online]. 2017 [cit. 2019-05-07]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/standard/components

[40] *What is .NET?* [online]. [cit. 2019-05-08]. Dostupné z: https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet

[41] Play Awesome Indie Games Directly in Firefox Including the Award-Winning FTL. *Mozzila* [online]. 2014 [cit. 2019-05-08]. Dostupné z: https://blog.mozilla.org/blog/2014/10/14/play-awesome-indie-games-directly-in-firefox-including-the-award-winning-ftl/

[42] *BROWSER COMPATIBILITY TESTING OF JAVASCRIPT ASM.JS* [online]. [cit. 2019-05-08]. Dostupné z: https://www.lambdatest.com/asm-js

[43] *Telerik UI for Blazor* [online]. [cit. 2019-05-14]. Dostupné z: https://www.telerik.com/blazor-ui

[44] TURNER, Aaron. *WebAssembly Is Fast: A Real-World Benchmark of WebAssembly vs. ES6* [online]. 2018 [cit. 2019-05-15]. Dostupné z: https://medium.com/@torch2424/webassembly-is-fast-a-real-world-benchmark-of-webassembly-vs-es6-d85a23f8e193

[45] JANGDA, Abhinav, Bobby POWERS, Arjun GUHA a Emery BERGER. *Mind the Gap: Analyzing the Performance of WebAssembly vs. Native Code* [online]. 2019 [cit. 2019-05-15]. Dostupné z: https://arxiv.org/abs/1901.09056

[46] HERERA, Omer. *Writing your first WebAssembly Project* [online]. 2019 [cit. 2019-05-16]. Dostupné z: https://medium.com/appsflyer/writing-your-first-webassembly-project-2a2d86b20e8f

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

API	Application Programming Interface
SDK	Software development kit
HTML	Hypertext Markup Language
IR	Intermediate Representation
CIL	Common intermediate Language
JSON	JavaScript Object Notation
AOT	Ahead Of Time
CSS	Cascading Style Sheets
SPA	Single Page Application
UI	User Interface
IoT	Internet of Things

SEZNAM OBRÁZKŮ

Obrázek 1 – Blazor oficiální logo [19] 11
Obrázek 2 - Ukázka WebAssembly formátu kódu [46] 12
Obrázek 3 - Otevření výsledné stránky 13
Obrázek 4 – Proces kompilace C/C++ kódu na asm.js[10] 14
Obrázek 5 - Kompilace jazyků do binární formy pomocí LLVM [12] 14
Obrázek 6 - Kompilace jazyků do JavaScriptu pomocí Emscripten [12] 15
Obrázek 7 – Mono logo [24] 21
Obrázek 8 - Interpreted mód [25] 21
Obrázek 9 – AOT mód [25] 22
Obrázek 10 – Client-side 24
Obrázek 11 – Server-side [28] 25
Obrázek 12 – Struktura projektu Client 31
Obrázek 13 – Struktura projektu Server 32
Obrázek 14 – Validace v dialogu 37
Obrázek 15 – Výsledná tabulka s funkcemi CRUD 38
Obrázek 16 – Výsledná Telerik UI komponenta 41
Obrázek 17 – Přenesené soubory do prohlížeče 42

SEZNAM PŘÍLOH

P1 CD disk

PŘÍLOHA P I: CD

Přiložené CD obsahuje:

- Bakalářskou práci ve formátu docx: fulltext.docx
- Bakalářskou práci ve formátu pdf: fulltext.pdf
- Zdrojové kódy: priloha.zip