Realtime synchronizace v ASP.NET Core s využitím SignalR

Daniel Jurča

Bakalářská práce
2023

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

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

Akademický rok: 2022/2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Daniel Jurča**
Osobní číslo:	**A20307**
Studijní program:	**B0613A140020 Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Realtime synchronizace v ASP.NET Core s využitím SignalR**
Téma práce anglicky:	**Real-time Synchronization in ASP.NET Core Using SignalR**

Zásady pro vypracování

1. Vypracujte literární rešerši na zadané téma.
2. Navrhněte aplikaci, která bude demonstrovat vhodné využití knihovny SignalR.
3. Naprogramujte aplikaci dle návrhu.
4. Věnujte pozornost zabezpečení aplikace.
5. Aplikaci vhodně popište.
6. Vytvořte prezentaci na téma SignalR, která bude shrnovat teoretické poznatky o této knihovně.

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

Seznam doporučené literatury:

1. SAZANAVETS, Fiodar. SignalR on .NET 6 – the Complete Guide: The easiest way to enable real-time two-way HTTP communication on .NET 6. USA: Independently published, 2022. ISBN 979-8796211779.
2. VEMULA, Rami. Real-Time Web Application Development: With ASP.NET Core, SignalR, Docker, and Azure. Berkeley, USA: Apress, 2017. ISBN 978-1-4842-3269-9.
3. Real-time web apps with SignalR [online]. Redmond, USA: Microsoft, 2022 [cit. 2022-12-01]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/signalr/introduction?view=aspnetcore-7.0.
4. APONTE, Michele. Building Single Page Applications in .NET Core 3: Jumpstart Coding Using Blazor and C#. Berkeley, USA: Apress, 2020. ISBN 978-1-4842-5746-3.
5. GIRETTI, Anthony. Beginning gRPC with ASP.NET Core 6: Build Applications using ASP.NET Core Razor Pages, Angular, and Best Practices in .NET 6. Berkeley, USA: Apress, 2022. ISBN 978-1-4842-8007-2.
6. SOH, Julian, Marshall COPELAND, Anthony PUCA a Micheleen HARRIS. Microsoft Azure: Planning, Deploying, and Managing the Cloud. Berkeley, USA: Apress, 2020. ISBN 978-1-4842-5957-3.

Vedoucí bakalářské práce: **Ing. Tomáš Vogeltanz, Ph.D.**
Ústav počítačových a komunikačních systémů

Datum zadání bakalářské práce: **2. prosince 2022**
Termín odevzdání bakalářské práce: **26. května 2023**

L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

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

Ve Zlíně dne 7. prosince 2022

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 bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- 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 – 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 …………………….
podpis studenta

ABSTRAKT

Tato bakalářská práce se zabývá využitím realtime synchronizace ve webových technologiích. Konkrétně se zaměřuje na technologii ASP.NET Core a knihovnu SignalR. V práci budou shrnuty možnosti knihovny SignalR a bude vypracována vhodná ukázka jejího použití. Hlavní části kódu dané ukázky budou dostatečně zdokumentovány. Práce také shrne jednotlivé případy, kdy je nutné ve webových technologiích aplikovat realtime synchronizaci. Součástí práce bude vytvoření prezentace na téma SignalR pro výukové účely v předmětu pokročilé webové technologie.

Klíčová slova:

Realtime synchronizace, SignalR, ASP.NET Core, webové aplikace, Angular

ABSTRACT

This bachelor thesis deals with the use of real-time synchronization in web technologies. Specifically, it focuses on the ASP.NET Core technology and the SignalR library. The thesis will summarize the capabilities of the SignalR library and develop a suitable demonstration of its use. The main parts of the code of the given demonstration will be sufficiently documented. The thesis will also summarize the different cases where real-time synchronization needs to be applied in web technologies. This thesis will include the creation of a presentation on SignalR for teaching purposes in the Advanced Web Technologies course.

Keywords:

Real-time synchronization, SignalR, ASP.NET Core, web applications, Angular

Na tomto místě bych rád poděkoval především vedoucímu práce Ing. Tomáši Vogeltanzovi, Ph.D., za jeho odborné vedení, cenné rady, trpělivost a ochotu, kterou mi v průběhu zpracování bakalářské práce věnoval. Dále chci poděkovat své rodině a přátelům za jejich podporu a povzbuzení během celého studia. Děkuji také svým kolegům ze společnosti Develogi za mentoring v oblasti vývoje webových aplikací.

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

OBSAH

ÚVOD ... **10**
I TEORETICKÁ ČÁST ... **11**
1 WEBOVÉ APLIKACE ... **12**
1.1 PROTOKOLY ... 12
1.1.1 HTTP ... 12
1.1.1.1 Metody ... 13
1.1.1.2 Požadavky ... 14
1.1.1.3 Odpovědi ... 14
1.1.1.4 HTTPS ... 15
1.1.2 WebSocket ... 15
1.1.2.1 Handshake – navázání spojení ... 16
1.1.2.2 Data framing ... 17
1.1.2.3 Přenos dat ... 17
1.1.2.4 Ukončení spojení ... 18
1.1.3 RTP ... 18
1.2 VÝVOJ WEBOVÝCH APLIKACÍ ... 19
1.2.1 Statické webové stránky ... 19
1.2.2 Dynamické stránky a AJAX ... 19
1.2.3 Jednostránkové webové aplikace ... 20
2 REALTIME SYNCHRONIZACE VE WEBOVÝCH APLIKACÍCH ... **22**
2.1 VYUŽITÍ ... 22
2.1.1 Aplikace pro sdílení dokumentů a prezentací ... 22
2.1.2 Instant messaging ... 23
2.1.3 Aplikace pro správu projektů a úkolů ... 23
2.1.4 Platformy pro online obchodování a finance ... 24
2.1.5 Monitorovací a řídicí aplikace ... 24
2.1.6 Online hry ... 25
2.2 TECHNIKY REALTIME KOMUNIKACE ... 25
2.2.1 Short polling ... 25
2.2.1.1 Implementace ... 26
2.2.2 Long polling ... 26
2.2.2.1 Implementace ... 27
2.2.3 WebSocket API ... 28
2.2.3.1 Implementace ... 28
2.2.4 Push API ... 28
2.2.4.1 Implementace ... 29
2.2.5 SSE ... 30
2.2.5.1 Implementace ... 30
3 SIGNALR ... **31**
3.1 SIGNALR HUB ... 31
3.1.1 Vytvoření ... 32
3.1.2 Konfigurace ... 32
3.1.3 Vlastnosti ... 32
3.1.3.1 Context ... 32
3.1.3.2 Clients ... 33

3.1.3.3 Groups ... 34
3.1.4 Eventy ... 34
3.2 PROPOJENÍ S JAVASCRIPT KLIENTEM ... 35
3.2.1 Instalace balíčku ... 35
3.2.1.1 Node Package Manager (NPM) ... 35
3.2.1.2 Content Delivery Network (CDN) ... 36
3.2.2 Připojení k hubu ... 36
3.3 KOMUNIKACE ... 36
3.3.1 Zasílání zpráv ze serveru ... 36
3.3.2 Příjem zpráv u klienta ... 37
3.3.3 Zasílání zpráv od klienta ... 37
3.4 STREAMOVÁNÍ ... 37
3.4.1 Streamování server-klient ... 37
3.4.2 Streamování klient-server ... 38
3.5 ALTERNATIVY ... 38
4 POUŽITÉ TECHNOLOGIE ... 40
4.1 BACKEND ... 40
4.1.1 ASP.NET Core ... 40
4.2 FRONTEND ... 40
4.2.1 Angular ... 41
4.2.2 PrimeNG ... 42
4.2.3 NGXS ... 42
4.3 HOSTING ... 42
4.3.1 Microsoft Azure ... 42
4.4 POUŽITÝ SOFTWARE ... 43
4.4.1 JetBrains Rider ... 43
4.4.2 Visual Studio Code ... 43
4.4.3 Figma ... 43
II PRAKTICKÁ ČÁST ... 45
5 NÁVRH APLIKACE ... 46
5.1 PRINCIP HRY ... 46
5.2 SYNCHRONIZACE ... 46
5.3 NÁVRH UŽIVATELSKÉHO ROZHRANÍ ... 46
5.3.1 Horní lišta ... 47
5.3.2 Levý panel ... 47
5.3.3 Pravý panel ... 47
5.3.4 Herní plocha ... 47
6 BACKEND ... 49
6.1 VYTVOŘENÍ PROJEKTU ... 49
6.2 LOGIKA HRY ... 49
6.2.1 Třídy ... 49
6.2.1.1 Player ... 49
6.2.1.2 Answer ... 50
6.2.1.3 LogRecord ... 51
6.2.1.4 Word ... 52

6.2.1.5 Game 53
6.2.2 Stavy hry 54
6.2.3 GameManager 56
6.2.4 PlayerManager 58
6.3 WEB API 59
6.3.1 LobbyController 59
6.3.2 RoomController 60
6.3.3 Data Transfer Object 61
6.4 SIGNALR HUB 62
6.4.1 Názvy metod 63
6.4.2 Implementace metod 63
6.4.2.1 Přihlášení 63
6.4.2.2 Připojení do hry 64
6.4.2.3 Odpojení ze hry 64
6.4.2.4 Synchronizace kresby 65
6.4.2.5 Hádání slova 67
7 FRONTEND 70
7.1 VYTVOŘENÍ PROJEKTU 70
7.1.1 Instalace balíčků 70
7.1.2 Propojení s API 70
7.2 STAV APLIKACE 70
7.2.1 AppState 71
7.2.2 RoomState 72
7.2.3 CanvasState 73
7.3 VZHLED 73
7.3.1 Rozložení 73
7.3.2 Design 75
7.4 SLUŽBA API A SIGNALR 75
7.4.1 API 75
7.4.2 SignalR 76
7.4.2.1 Vytvoření spojení 76
7.4.2.2 Přihlášení 77
7.4.2.3 Aktualizace dostupných her 78
7.4.2.4 Připojení do hry 78
7.4.2.5 Zprávy chatu 79
7.5 KOMPONENTY 79
7.5.1 Přihlašovací stránka 80
7.5.1.1 Šablona 80
7.5.1.2 Třída 81
7.5.2 Lobby 82
7.5.2.1 Šablona 82
7.5.2.2 Třída 83
7.5.3 Herní místnost 85
7.5.3.1 Šablona 85
7.5.3.2 Třída 86
7.5.3.3 Seznam hráčů 88
7.5.3.4 Záznamy aktivit 88

7.5.3.5 Kreslicí plátno 88
7.5.3.6 Status hry 91
7.5.3.7 Nástroje pro kreslení 92
7.5.3.8 Chat 92
8 OPTIMALIZACE A VYSTAVENÍ 93
8.1 OPTIMALIZACE SIGNALR 93
8.2 VYSTAVENÍ DO MICROSOFT AZURE 95
8.2.1 Backend 95
8.2.2 Frontend 96
8.3 ZABEZPEČENÍ APLIKACE 97
8.3.1 ConnectionId 97
8.3.2 Hádané slovo 98
8.3.3 CORS 98
ZÁVĚR 99
SEZNAM POUŽITÉ LITERATURY 100
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK 107
SEZNAM OBRÁZKŮ 109
SEZNAM TABULEK 110
SEZNAM PŘÍLOH 111

ÚVOD

V dnešní digitální době se webové aplikace stávají stále důležitějšími nástroji pro mnoho oblastí lidské činnosti, od komunikace a vzdělávání až po obchod a zábavu. S rostoucí popularitou a potřebou efektivních webových aplikací se vyvíjejí také technologie, které podporují jejich vývoj. Jednou z takových technologií je ASP.NET Core, který umožňuje vytvářet výkonné a škálovatelné webové aplikace a služby.

Jednou z klíčových funkcí, které mohou webové aplikace poskytnout, je schopnost synchronizovat data v reálném čase mezi různými uživateli nebo relacemi. Tato funkce je nezbytná pro mnoho typů aplikací, včetně těch, které podporují spolupráci, sdílení informací, streamování médií a interaktivní zábavu. SignalR je knihovna, která umožňuje vývojářům snadno přidávat funkce synchronizace v reálném čase do jejich ASP.NET Core aplikací.

Tato bakalářská práce se zaměřuje na realtime synchronizaci v ASP.NET Core s využitím SignalR. V teoretické části práce budou zkoumány webové aplikace, jejich vývoj a využití realtime synchronizace ve webových aplikacích. Dále bude podrobně prozkoumána knihovna SignalR, její možnosti a funkce. Praktická část se pak zaměří na vytvoření konkrétní aplikace s využitím výše zmíněných technologií.

Tato práce má za cíl nejenom poskytnout teoretický přehled o realtime synchronizaci a knihovně SignalR, ale také demonstrovat praktické použití této technologie v kontextu moderního vývoje webových aplikací.

I. TEORETICKÁ ČÁST

1 WEBOVÉ APLIKACE

Webové aplikace představují jeden z nejdůležitějších způsobů, jakým lidé komunikují, získávají informace a pracují v digitálním světě. Od svého vzniku v 90. letech 20. století prošly řadou vývojových fází, které zahrnovaly různé technologie a architektury. [1] Dnes je vývoj webových aplikací klíčovou disciplínou v oblasti softwarového inženýrství, a to zejména díky rostoucí potřebě interakce, spolupráce a sdílení dat v reálném čase. [2]

Webové aplikace jsou v principu programy, které běží v internetovém prohlížeči a umožňují uživatelům interagovat s rozhraním prostřednictvím webových stránek. [3] Tyto aplikace se skládají ze dvou hlavních částí: frontendu, klientské části týkající se vizuálních a interaktivních prvků aplikace, a backendu, který zajišťuje zpracování dat a komunikaci s databázemi nebo jinými serverovými službami. [4] [5]

1.1 Protokoly

Protokoly představují základní stavební kameny komunikace v internetovém prostředí a hrají klíčovou roli ve fungování webových aplikací. Jsou to soubory pravidel a konvencí, které umožňují bezproblémový přenos informací mezi různými systémy a zařízeními. V kontextu webových aplikací se jedná o komunikační vrstvu, která zajišťuje, že data jsou správně formátována, přenášena, přijímána a zpracovávána mezi klienty a servery.

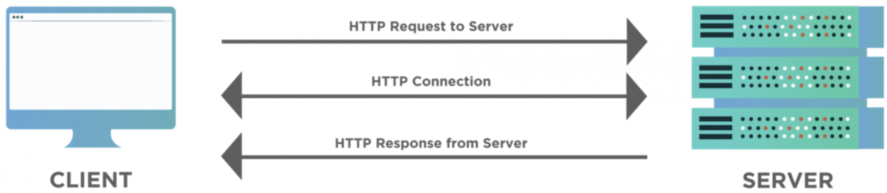
Existuje několik různých protokolů, které se používají v rámci webových aplikací, a každý z nich má své specifické vlastnosti, výhody a nevýhody. Některé protokoly jsou navrženy pro přenos hypertextových dokumentů, zatímco jiné se zaměřují na přenos multimediálního obsahu nebo na komunikaci v reálném čase. [6] [7] Výběr vhodného protokolu pro konkrétní aplikaci závisí na požadavcích na rychlost, spolehlivost, bezpečnost a funkčnost. [8]

V následujících podkapitolách jsou podrobněji rozebrány některé z nejběžnějších protokolů používaných ve webových aplikacích a zejména komunikaci v reálném čase, jako jsou HTTP, RTP nebo WebSocket.

1.1.1 HTTP

Základním protokolem pro datovou komunikaci na webu je Hypertext Transfer Protocol a umožňuje výměnu dat ve formě hypertextových dokumentů, obrázků, videí nebo skriptů. Využívá port s číslem 80. [9]

Protokol je neustále vyvíjen a aktuálně je ve verzi HTTP/3, ovšem nejpoužívanější jsou stále verze HTTP/1.1 a HTTP/2. [10] [11] Nové verze přinášejí zlepšení výkonu, zabezpečení a použitelnosti, a zahrnují trvalá připojení, prioritizaci zdrojů, kompresi, ukládání do mezipaměti, ověřování, šifrování nebo multiplexing. [12]



Obrázek 1. Schéma architektury klient-server [13]

Jedná se o protokol typu klient-server, jenž umožňuje klientům, typicky webovým prohlížečům, iniciovat požadavky na stranu webového serveru. Ten požadavek (request) zpracuje a klientovi pošle zpět odpověď (response) ve formě zprávy.

HTTP je bezstavový protokol, což znamená, že každý požadavek je nezávislý na předchozích. Aplikace však někdy vyžadují stavové interakce, proto k udržení relací aplikace napříč více požadavky a odpověďmi nabízí různé mechanismy, jako jsou soubory cookie, přepisování adres URL nebo skrytá pole formulářů. [14]

1.1.1.1 Metody

HTTP definuje několik dotazovacích metod, které určují akci, jež se má na serveru provést. Mezi základní patří:

- GET – požadavek na získání určitých dat
- POST – odeslání dat na server, typicky obsah formuláře
- PUT – nahrání dat na server
- DELETE – smazání dat ze serveru

Dále existují metody HEAD, TRACE, OPTIONS nebo CONNECT. [15] Metody sloužící pouze pro čtení dat, ne jejich úpravu, jsou nazývány bezpečné. [16] Implementace akcí pro jednotlivé metody na serveru však není striktně dána a je závislá na vývojáři, jejich popis tedy není vždy zárukou toho, co metoda provede.

1.1.1.2 Požadavky

Požadavky jsou zprávy odeslané klientem na server, aby provedl určitou akci na webovém zdroji.

```
GET /hello.html HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
Host: www.example.com
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

Skládají se z metody, URI (Uniform Resource Identifier), verze HTTP protokolu klienta, hlaviček poskytující další informace a těla zprávy. URI identifikuje cílový prostředek.

Hlavičky poskytují další informace o požadavku nebo klientovi, například řetězec user-agent identifikující klientský software, akceptované typy obsahu a soubory cookies. Hlavičky jsou nepovinné a každá z nich se zapisuje na nový řádek.

Tělo požadavku je rovněž nepovinné a obsahuje všechna data, která je potřeba odeslat na server. Používá se u metod jako POST nebo PUT, které odesílají data na server. Například při odesílání formuláře jsou zadaná data obsažena v těle požadavku. [17]

1.1.1.3 Odpovědi

Odpovědi jsou zprávy zaslané serverem zpět klientovi jako reakce na jeho požadavky.

```
HTTP/1.1 200 OK
Date: Fri, 05 May 2023 12:34:56 GMT
Server: Apache/2.4.49 (Ubuntu)
Last-Modified: Mon, 02 May 2023 10:00:00 GMT
Content-Length: 88
Content-Type: text/html
Connection: Closed

<html>
 <body>
  <h1>Hello, World!</h1>
 </body>
</html>
```

Na prvním řádku odpovědi se nachází stavový řádek obsahující verzi protokolu HTTP, stavový kód a text stavu. Stavový kód je třímístné číslo, jehož první číslice určuje třídu odpovědi. Například stavové kódy 2xx představují úspěšný požadavek, 3xx přesměrování, 4xx chyby klienta a 5xx představují chyby serveru. Stavový text je stručná, lidsky čitelná zpráva, která popisuje stavový kód.

Hlavičky v odpovědi poskytují další informace o odpovědi nebo serveru, například délku obsahu a jeho typ, informace o serveru nebo nastavené soubory cookies. V těle odpovědi umístěném za hlavičkami jsou pak samotná data (například HTML, JSON, obrázek atd.), která server odešle zpět jakožto výsledek požadavku. [18]

1.1.1.4 HTTPS

Hypertext Transfer Protocol Secure je šifrovaná verze protokolu HTTP, která chrání uživatelská data vyměňovaná mezi prohlížečem a webovou stránkou. Používá protokol TLS (Transport Layer Security) nebo dříve SSL (Secure Sockets Layer) k vytvoření zabezpečeného kanálu pomocí veřejných a soukromých klíčů. Běží na portu 443.

Použití protokolu HTTPS zabraňuje útočníkům zachytit, upravit nebo ukrást přenášená data, která mohou obsahovat citlivé informace, jako jsou přihlašovací nebo osobní údaje. Ověřuje také pravost webových stránek a chrání před phishingovými útoky nebo škodlivými webovými stránkami.

Pro aktivaci protokolu HTTPS musí majitelé webových stránek získat digitální certifikát od certifikační autority, nainstalovat jej na svůj webový server a nakonfigurovat web tak, aby používal protokol HTTPS. Prohlížeče pak před navázáním zabezpečeného spojení s webovou stránkou ověří pravost certifikátu. [19]

1.1.2 WebSocket

WebSocket byl coby protokol představen jako součást specifikace HTML5 fungující na principu vytváření trvalého spojení mezi prohlížečem a serverem. Slouží jako nezávislý protokol založený na protokolu TCP, který se svým designem a funkcí liší od protokolu HTTP. Protokol TCP zajišťuje spolehlivý přenos dat mezi aplikacemi na různých počítačích. [20]

Sdílí však spojení s HTTP v tom smyslu, že jeho handshake je HTTP servery interpretován jako požadavek na aktualizaci. To umožňuje zachovat kompatibilitu se stávající infrastrukturou HTTP a bezproblémově navazovat WebSocket spojení. Ve výchozím nastavení používá WebSocket port 80 pro běžná spojení, zatímco port 443 se používá pro spojení přes

protokol TLS. Toto uspořádání zajišťuje, že WebSocket spojení mohou existovat současně se standardním webovým provozem a využívat stejné porty k zajištění obousměrné komunikace v reálném čase v různých aplikacích.

Velkou výhodou je schopnost plně duplexní komunikace, což znamená, že obě strany mohou odesílat a přijímat data současně, aniž by čekaly na odpověď. Na druhou stranu jej však nepodporují všechny webové prohlížeče a servery, což může vyžadovat záložní řešení nebo polyfill (kód přidávající moderní funkce pro starší prohlížeče [21]). Rovněž s sebou nese některá bezpečnostní rizika, jako je například cross-site WebSocket hijacking, kdy útočník může pomocí škodlivé webové stránky získat přístup k WebSocket spojení navázanému jinou webovou stránkou. [22] Protokol taktéž vyžaduje více prostředků a větší šířku pásma než požadavky a odpovědi HTTP, což může mít vliv na škálovatelnost a výkon webových aplikací.

1.1.2.1 Handshake – navázání spojení

Pro navázání WebSocket spojení zahájí klient handshake odesláním požadavku HTTP na server se specifickými hlavičkami. Požadavek musí obsahovat hlavičku "Upgrade" nastavenou na "websocket" a hlavičku "Connection" nastavenou na "Upgrade", která označuje záměr provést upgrade na protokol WebSocket. Kromě toho musí požadavek obsahovat hlavičku "Sec-WebSocket-Key" obsahující náhodnou hodnotu kódovanou do base64 a hlavičku "Sec-WebSocket-Version" označující verzi protokolu podporovanou klientem.

```
GET /websocket-endpoint HTTP/1.1
Host: example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Version: 13
Origin: http://example.com
```

Pokud server podporuje protokol WebSocket a souhlasí s upgradem připojení, odpoví stavovým kódem HTTP 101 (přepínání protokolů). Server musí v odpovědi uvést hlavičku "Upgrade" nastavenou na "websocket" a hlavičku "Connection" nastavenou na "Upgrade". Také musí zahrnout hlavičku "Sec-WebSocket-Accept", která je generována hashováním klientova klíče "Sec-WebSocket-Key" s předem definovaným GUID a následným kódováním do base64.

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
```

Jakmile klient obdrží od serveru odpověď s kódem 101 se správnými hlavičkami, je handshake považován za dokončený a spojení je převedeno na protokol WebSocket. Od tohoto okamžiku si klient i server mohou vyměňovat data ve formě rámců WebSocket prostřednictvím stejného spojení. [23]

1.1.2.2 Data framing

Data framing označuje strukturu datových rámců vyměňovaných mezi klientem a serverem. Rámec WebSocket protokolu se skládá ze záhlaví a přenášených dat, tzv. payload. Záhlaví obsahuje metadata, mezi něž patří příznak konečného rámce (FIN), rezervované bity (RSV1, RSV2, RSV3), opcode určující typ dat, příznak masky, délku payloadu a volitelný maskovací klíč.

Payload obsahuje skutečnou přenášenou zprávu. U textových rámců jsou tato data kódována jako UTF-8. Data jsou maskována, pokud je příznak maskování v záhlaví rámce nastaven na hodnotu 1. V takovém případě musí přijímající koncový bod data odmaskovat pomocí funkce XOR s maskovacím klíčem obsaženým v záhlaví rámce.

1.1.2.3 Přenos dat

Když chce klient nebo server odeslat zprávu, vytvoří rámec WebSocket obsahující data zprávy a metadata v záhlaví. Pokud je zpráva příliš velká na to, aby se vešla do jednoho rámce, může být rozdělena do více rámců. V takových případech je poslední rámec zprávy v záhlaví označen příznakem FIN nastaveným na hodnotu 1. Pokud data odesílá klient, je payload z bezpečnostních důvodů maskován pomocí maskovacího klíče uvedeného v záhlaví.

Při přijmutí zprávy si klient nebo server nejprve přečte záhlaví rámce, aby zjistil jeho metadata. Je-li rámec pokračováním předchozí zprávy, jsou data připojena k datům dané zprávy. Pokud jde o poslední rámec zprávy (indikováno příznakem FIN nastaveným na 1), považuje se za přijatou celá zpráva. V případě maskovaných dat odeslaných klientem musí příjemce tato data odmaskovat pomocí maskovacího klíče uvedeného v záhlaví, až poté je lze zpracovat.

1.1.2.4 Ukončení spojení

Uzavření spojení může iniciovat buď klient nebo server. To se provede odesláním rámce s kódem 0x8, jenž označuje rámec pro uzavření. Iniciátor může do payloadu zahrnout také stavový kód, který označuje důvod uzavření, není to ovšem povinné. Mezi běžné stavové kódy patří kódy 1000 (indikuje normální uzavření) a 1001 (indikuje, že koncový bod odchází, například vypnutí serveru nebo odchod prohlížeče ze stránky).

Po přijetí uzavíracího rámce se očekává, že příjemce pošle zpět obdobný rámec jako odpověď. Tím potvrdí, že zprávu obdržel a je připraven spojení také uzavřít. Po odeslání tohoto uzavíracího rámce by pak měl příjemce TCP spojení uzavřít. Pokud však příjemce zpětný uzavírací rámec neodešle, může iniciátor po uplynutí přiměřeného časového limitu vynutit ukončení spojení. [24]

1.1.3 RTP

Protokol RTP (Real-time Transport Protocol) je síťový standard určený pro konzistentní a včasný přenos zvukových nebo obrazových dat prostřednictvím sítí IP. Má široké využití v aplikacích, které zahrnují streamování médií, jako je VoIP, videokonference, webová komunikace a televizní služby. Protokol RTP může doručovat data do více destinací pomocí vícesměrového vysílání IP a dokáže se přizpůsobit různým podmínkám v síti a formátům médií. [25]

RTP spolupracuje s protokolem RTCP (RTP Control Protocol), který poskytuje zpětnou vazbu o kvalitě a výkonu přenosu médií. RTCP může sledovat statistiky, jako je ztráta paketů, zpoždění a využití šířky pásma, a může pomoci synchronizovat více mediálních proudů. Lze jej také použít pro účely signalizace a řízení, například pro identifikaci účastníků a zdrojů mediální relace. [26] Na jiné protokoly se spoléhá i v jiných oblastech. Sám například nespecifikuje způsob navázání nebo správy mediální relace ani nedefinuje kodeky nebo formáty mediálních dat. To přenechává protokolu SIP (Session Initiation Protocol), jenž tyto aspekty řeší. [27]

RTP může podporovat různé typy mediálních kodeků, například G.711 pro zvuk, H.264 pro video nebo MP3 pro hudbu. [28] Podporuje také šifrování; šifrovaný protokol se nazývá SRTP (Secure Real-time Transport Protocol) a přidává do protokolu RTP funkce zabezpečení a ochrany soukromí. [29]

Díky flexibilitě a přizpůsobivosti různým typům mediálních aplikací a scénářů je RTP jedním z hlavních protokolů pro multimediální komunikaci v reálném čase. [30]

1.2 Vývoj webových aplikací

Vývoj webových aplikací prošel od svého vzniku v 90. letech 20. století řadou změn a evoluce. Tyto změny byly výsledkem pokroku v technologiích, nástrojích a konceptech používaných ve webovém vývoji, stejně jako rostoucích požadavků na funkcionalitu, interakci a použitelnost aplikací. Díky tomuto vývoji se webové aplikace stávají stále složitějšími, s větším důrazem na uživatelské zkušenosti a dynamický obsah.

1.2.1 Statické webové stránky

Na počátku a v polovině 90. let byly webové aplikace postaveny především na statických stránkách HTML, přičemž každá interakce uživatele vyžadovala načtení celé nové stránky ze serveru. Tento přístup byl nevhodný hned z několika důvodu.

Neustálé načítání webových stránek vedlo k pomalému a nepřehlednému uživatelskému prostředí, protože veškerý obsah zmizel dříve, než se objevila nová stránka. Pokaždé, když uživatel požádal o novou stránku, musel server zpracovat a odeslat celý obsah, i když se změnila jen malá část informací. Závislost na úplném načtení stránky i pro drobné aktualizace obsahu spotřebovávala značnou šířku pásma, což z ní činilo omezující faktor výkonu webových stránek. [1] [31]

1.2.2 Dynamické stránky a AJAX

Postupem času se vývoj webových aplikací vyvíjel tak, aby omezení statických stránek odstranil a připravil tak cestu pro dynamičtější a interaktivnější aplikace.

Zavedení jazyka JavaScript umožnilo vývojářům vytvářet interaktivnější webové stránky pomocí manipulace s objektovým modelem dokumentu (DOM) a reagovat na události uživatele, aniž by bylo nutné znovu načítat celou stránku. Stále se však jednalo o statické stránky, neboť bylo nutné ze serveru zaslat veškerý obsah stránky, který JavaScript následně upravoval.

Koncem 90. let představil Microsoft objekt XMLHttpRequest (XHR), sloužící původně pro použití s aplikací Outlook Web Access. Ten umožnil asynchronní odesílání požadavků a přijímání odpovědí pomocí HTTP protokolu. Asynchronní požadavky jsou z klienta na server a neblokují klienta v pokračování provádění kódu při čekání na odpověď.

To znamená, že klient může pokračovat v interakci se stránkou, zatímco se čeká na odpověď serveru. Přestože XMLHttpRequest ve svém názvu nese XML, není omezen pouze na data ve formátu Extensible Markup Language, a mnohem častěji jsou data přenášena ve formátu JSON, který je rychlejší na zpracování a podporuje pole dat. [32]

Rozšíření XMLHttpRequest znamenalo výrazný odklon od do té doby tradičního přístupu, kdy se při každé interakci s uživatelem používalo nové načtení stránky. Díky XHR ve spojení s JavaScriptem mohli vývojáři aktualizovat části webové stránky a zobrazovat nová data, aniž by vyžadovali kompletní obnovení, což zlepšilo celkový uživatelský zážitek a snížilo zatížení serveru. Tato technika je dnes známá pod zkratkou AJAX (Asynchronous JavaScript and XML). AJAX se stal základem při vytváření moderních interaktivních webových aplikací, což vedlo k vývoji různých knihoven a frameworků, které zjednodušují proces vytváření aplikací založených na této technice. [33]

1.2.3 Jednostránkové webové aplikace

S růstem výkonu prohlížečů, zařízení uživatelů a zavedením nových webových standardů začaly webové aplikace nabývat na složitosti a interaktivitě. Jedním z konceptů, který výrazně změnil způsob, jakým jsou dnes webové aplikace navrhovány a vyvíjeny, jsou Single Page Applications (SPA).

SPA jsou založeny na myšlence, že místo načítání nových stránek nebo aktualizací částí stránky prostřednictvím AJAX, se veškerý obsah a interakce odehrává na pouze jedné stránce. Když uživatel interaguje s aplikací, stránka je dynamicky aktualizována pomocí JavaScriptu, který manipuluje s DOM a načítá nový obsah z API na pozadí. [34]

Existuje řada populárních JavaScriptových frameworků a knihoven, které usnadňují vytváření SPA. Mezi nejznámější patří React vyvinutý společností Facebook, Angular od společností Google nebo Vue.js. Tyto frameworky a knihovny poskytují vývojářům nástroje a abstrakce potřebné pro rychlé a efektivní vytváření moderních a škálovatelných jednostránkových aplikací.

Je důležité mít na paměti, že tento typ aplikací s sebou přináší taktéž určité nevýhody. SPA může mít pomalejší prvotní načítání, protože je nutné stáhnout veškerý kód aplikace a zdroje na začátku. To může být problém zejména pro uživatele s pomalejším internetovým připojením. Avšak optimalizační techniky jako lazy-loading mohou tento problém zmírnit. Mimo to dynamické generování obsahu znesnadňuje optimalizaci pro webové vyhledávače (SEO),

které mohou mít problém s indexací. I přes tyto výzvy SPA představuje významný posun v moderním webovém vývoji, který umožňuje vytvářet rychlejší, plynulejší a více interaktivní webové aplikace. [35]

2 REALTIME SYNCHRONIZACE VE WEBOVÝCH APLIKACÍCH

V dnešní době se neustále zvyšuje míra, v jaké webové aplikace nabízejí pokročilé funkce a vyšší úroveň interaktivity. S tím, jak se webové aplikace stávají sofistikovanějšími, roste i důraz na zobrazování aktuálních dat a jejich synchronizaci v reálném čase.

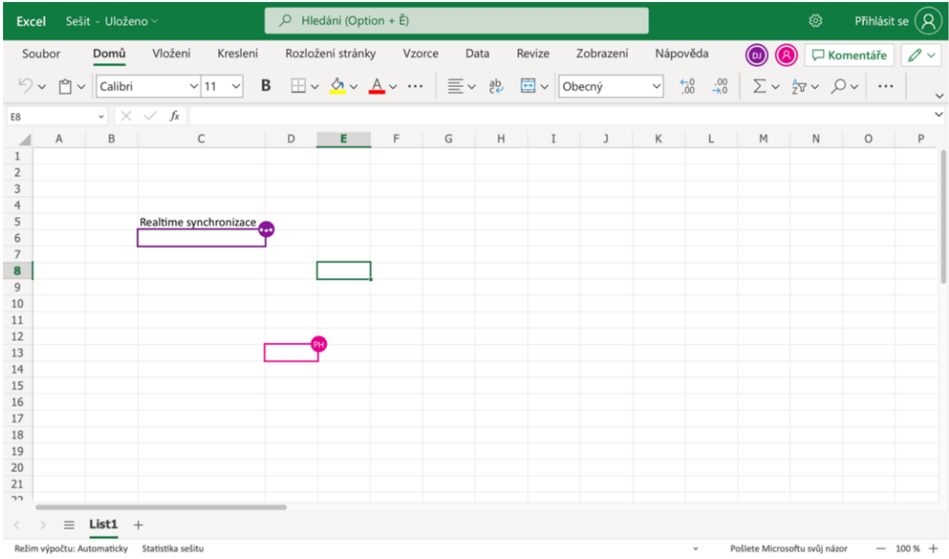
Tato schopnost přináší uživatelům aktuální informace bez nutnosti manuálně aktualizovat stránku nebo čekat na načítání nových dat. Změny přitom nemusí vyvolávat pouze server, ale také ostatní uživatelé. To znamená, že když jeden uživatel aktualizuje data, všichni ostatní uživatelé připojeni k aplikaci obratem obdrží aktualizovaná data.

2.1 Využití

Realtime synchronizace se stává zásadním aspektem v mnohých typech aplikací, kde schopnost poskytovat uživatelům aktuální informace a plynulou interakci s daty je základem pro funkčnost nebo zlepšení uživatelského zážitku.

2.1.1 Aplikace pro sdílení dokumentů a prezentací

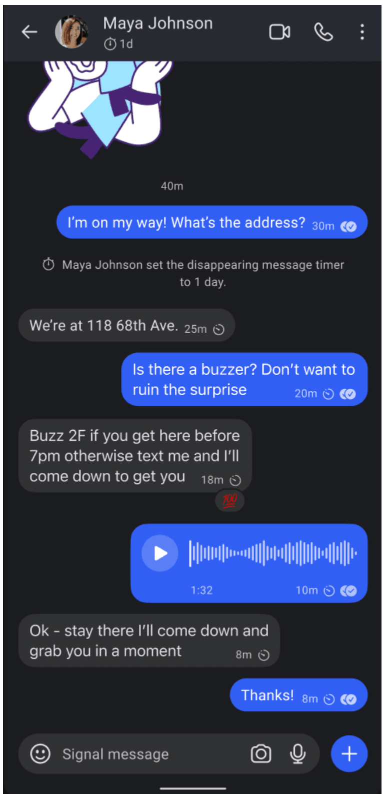
Mezi takové aplikace patří například aplikace jako Google Docs nebo online verze nástrojů Microsoft Office. Umožňují uživatelům sdílet a upravovat dokumenty a prezentace s ostatními a synchronizace v reálném čase zajišťuje, že změny provedené jedním uživatelem jsou okamžitě viditelné pro ostatní, což podporuje efektivní spolupráci a koordinaci.



Obrázek 2. Microsoft Excel v prohlížeči

2.1.2 Instant messaging

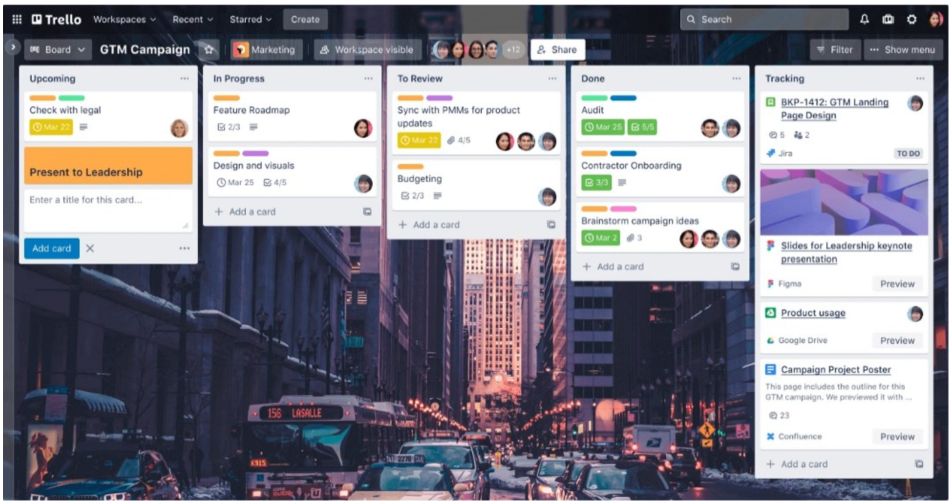
Dalším typickým zástupcem jsou služby pro okamžité zasílání zpráv. Umožňují uživatelům komunikovat v reálném čase pomocí textových zpráv, hlasových hovorů nebo videohovorů. Některé z nich také podporují přenos souborů, sdílení obrazovky nebo skupinové konverzace. Na rozdíl od e-mailu druhá strana ví, zda je příjemce aktuálně připojený a dostupný. Instant messaging aplikace jsou populární pro osobní i profesionální účely, protože nabízejí rychlý a snadný způsob, jak se spojit s lidmi po celém světě. Mezi nejznámější aplikace patří Facebook Messenger, WhatsApp, Discord, Signal nebo Telegram.



Obrázek 3. Aplikace Signal [36]

2.1.3 Aplikace pro správu projektů a úkolů

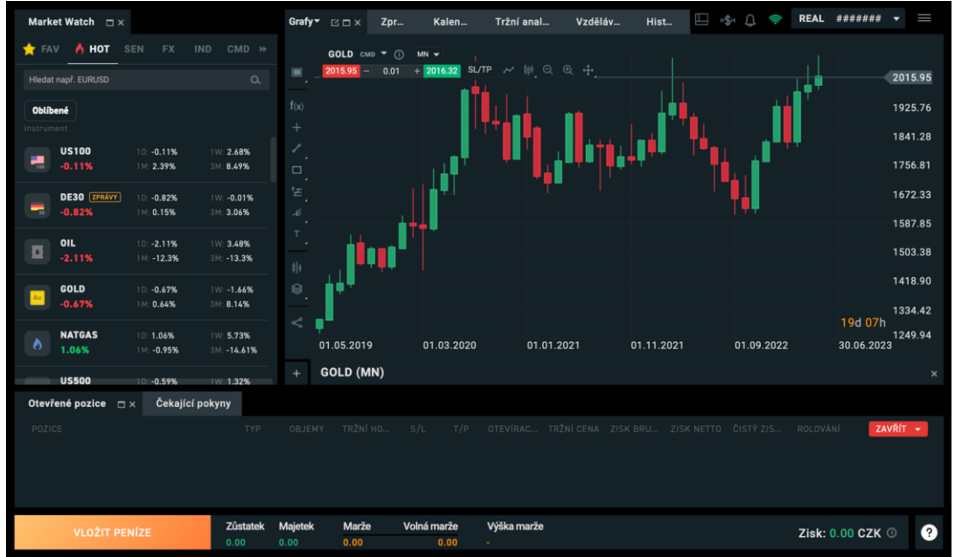
Aplikace dovolující uživatelům plánovat, organizovat a sledovat své projekty a úkoly v reálném čase. Například aplikace Trello umožňuje vytvářet a spravovat nástěnky s kartami, které reprezentují projekty a úkoly, a synchronizovat je mezi další zařízeními.



Obrázek 4. Aplikace Trello [37]

2.1.4 Platformy pro online obchodování a finance

Při obchodování na burze či sledování finančních trhů je důležité mít okamžitý přístup k aktuálním datům. Realtime synchronizace v těchto aplikacích umožňuje aktualizovat cenu například každou sekundu, což dovoluje obchodníkům sledovat pohyb trhu a okamžitě reagovat na změny.



Obrázek 5. Obchodovací platforma xStation 5

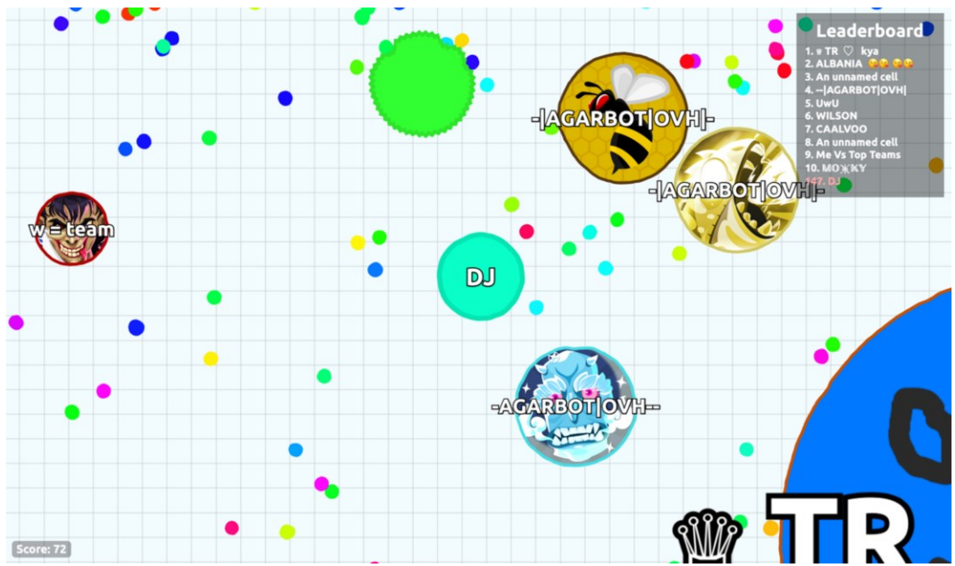
2.1.5 Monitorovací a řídicí aplikace

Dalším typem jsou aplikace pro sledování provozu, chytrou domácnost nebo řízení průmyslových systémů, vyžadující nepřetržité aktualizace stavu zařízení a procesů. Realtime

synchronizace zde poskytuje uživatelům přesné informace o stavu systémů a umožňuje rychle přizpůsobit nastavení v reakci na změny.

2.1.6 Online hry

V neposlední řadě nalezne realtime synchronizace využití v online hrách, které umožňují hrát společně nebo proti sobě v reálném čase s minimálním zpožděním. [38]



Obrázek 6. Hra Agar.io

2.2 Techniky realtime komunikace

Technologie, která stojí za synchronizací v reálném čase, se v průběhu let výrazně vyvinula a byly vytvořeny nové protokoly a API rozhraní, které usnadňují komunikaci mezi klienty a servery v reálném čase. I přes technologický pokrok však zůstává implementace synchronizace ve webových aplikacích poměrně náročným úkolem.

V této části práce jsou popsány různé metody a technologie, které mohou vývojáři využít pro realizaci realtime synchronizace v jejich projektech. Pro lepší znázornění jsou uvedeny příklady implementace vybraných metod na straně klienta v jazyce JavaScript.

2.2.1 Short polling

Krátká synchronizace dotazováním (short polling) je technika pro aktualizaci stavu distribuovaného systému periodickým odesíláním požadavků mezi klientem a serverem. Při krátkém dotazování klient opakovaně odesílá požadavky na server v určitém intervalu, například každých několik sekund, aby zkontroloval, zda došlo k aktualizaci dat. Pokud aktualizace

existují, server je odešle zpět klientovi. Pokud žádné aktualizace nejsou, server odpoví zprávou, že nemá žádná nová data.

Krátké dotazování je jednoduchá a snadno implementovatelná technika, má však určitá omezení. Může například vést k velkému množství zbytečných požadavků a při práci s velkým množstvím dat může být neefektivní. Může také vést k latenci a zpoždění, protože klient musí čekat na odpověď serveru, i když nedochází k žádným aktualizacím. [39]

Navzdory těmto omezením může být krátké dotazování v určitých scénářích užitečné, například když jsou synchronizovaná data relativně malá nebo když aktualizace v reálném čase nejsou kritické. Rovněž je-li vyžadována vysoká kompatibilita, jelikož tento typ dotazování spoléhá na standardní HTTP požadavky a je tím pádem podporován de facto všemi internetovými prohlížeči a servery. V jiných případech mohou být vhodnější jiné synchronizační techniky, například dlouhé dotazování nebo WebSocket. [40]

2.2.1.1 Implementace

```
async function shortPoll() {
  try {
      const response = await fetch('http://example.com/api/data')
      const data = await response.json()
      console.log(data)
      setTimeout(shortPoll, 5000)
  } catch (error) {
      console.error('Error:', error)
  }
}
```

Funkce shortPoll periodicky stahuje data z endpointu (koncového bodu) na serveru v pevném intervalu 5 sekund. Toto periodické stahování dat se provádí pomocí funkce setTimeout, která každých 5 sekund rekurzivně volá funkci shortPoll.

2.2.2 Long polling

Long polling, v překladu dlouhé dotazování, je technika synchronizace, která umožňuje webovému serveru odesílat aktualizace klientovi bez čekání na požadavek. Používá se k implementaci aplikací pracujících v reálném čase jako je chat, informační kanály nebo oznámení.

Základní myšlenka synchronizace pomocí dlouhého dotazováním spočívá v tom, že klient odešle serveru požadavek a čeká na odpověď. Server neodešle odpověď okamžitě, ale místo

toho ponechá spojení otevřené, dokud není třeba odeslat aktualizaci. Jakmile má server k dispozici aktualizaci, odešle ji klientovi a spojení uzavře. Klient pak aktualizaci zpracuje a odešle serveru další požadavek, čímž se celý proces opakuje. [41]

Použití synchronizace s dlouhým dotazováním má oproti tradičním technikám dotazování nebo streamování několik výhod. Zaprvé snižuje režii sítě a latenci, protože klient nemusí posílat časté požadavky nebo přijímat zbytečná data. Zadruhé umožňuje serveru řídit frekvenci a načasování aktualizací, což může zlepšit výkon a škálovatelnost aplikace. Zatřetí zjednodušuje logiku na straně klienta, protože klient musí v daném okamžiku zpracovávat pouze jeden požadavek. [40]

Synchronizace s dlouhým dotazováním má však také některé nevýhody a problémy. Jednou z nich je fakt, že vyžaduje, aby server udržoval otevřené spojení pro každého klienta, což může spotřebovávat mnoho prostředků a omezovat počet současně pracujících uživatelů. Rovněž nemusí dobře fungovat s některými proxy servery nebo firewally, které mohou uzavírat nečinná připojení nebo narušovat přenos dat. Třetí nevýhodou je možná nekompatibilita s některými prohlížeči nebo zařízeními, která nepodporují dlouhé dotazování nebo mají omezenou výdrž baterie. [42]

2.2.2.1 Implementace

```
async function longPoll() {
  try {
      const response = await fetch('http://example.com/api/data')
      const data = await response.json()
      console.log(data)
      longPoll()
  } catch (error) {
      console.error('Error:', error)
  }
}
```

Implementace dlouhého dotazování na straně klienta je v podstatě shodná s implementací krátkého dotazování, ale funkce se volá znovu ihned po obdržení odpovědi. Rozdíl této techniky je na straně serveru, který musí být schopen udržet požadavek aktivní, dokud nejsou k dispozici nová data, která následně odešle jako odpověď.

2.2.3 WebSocket API

Další možností pro komunikaci v reálném čase je využití protokolu WebSocket pomocí WebSocket API, jež je součástí specifikace HTML5. Poskytuje vývojářům rozhraní pro interakci s protokolem v prostředí webového prohlížeče. Je implementováno v jazyce JavaScript. [43]

Oproti dlouhému dotazování nabízí plně duplexní komunikaci, tudíž si klient a server mohou posílat zprávy současně, aniž by čekali na odpověď na předchozí požadavek. Díky tomu, že WebSocket udržuje nepřetržité spojení mezi klientem a serverem, snižuje režii opakovaného otevírání a zavírání spojení jako je tomu při dlouhém dotazování. To vede k větší efektivitě z hlediska využití sítě a výkonu a nižší latenci, zejména u aplikací, které vyžadují časté aktualizace. [44]

2.2.3.1 Implementace

Pro použití je nejprve nutné vytvořit WebSocket spojení zavoláním konstruktoru s URL adresou serveru, na který chceme navázat připojení.

```
const socket = new WebSocket("wss://example.com")
```

Následně můžeme naslouchat na eventy, které objekt WebSocket vyvolá. Prostřednictvím eventů lze reagovat na otevření a uzavření spojení, chyby nebo příchozí zprávy. K odeslání dat na server lze použít metodu send.

```
socket.addEventListener("open", (event) => {
  socket.send("Hello Server!")
})

socket.addEventListener("message", (event) => {
  console.log("Message from server ", event.data)
})
```

K ukončení spojení slouží metoda close. [45]

```
socket.close()
```

2.2.4 Push API

Push API je webový standard, který umožňuje asynchronní komunikaci mezi webovou aplikací a uživatelským agentem, typicky webovým prohlížečem. Jednou z jedinečných vlastností tohoto rozhraní je jeho schopnost doručovat zprávy, i když webová aplikace není právě

aktivní nebo otevřená v okně prohlížeče. To je užitečné zejména v situacích, kdy uživatel může aplikaci zavřít, ale přesto může využít toho, že se aplikace znovu spustí, jakmile přijde nová zpráva. Zpráva push by například mohla uživatele upozornit na příchozí hovor.

Rozhraní podporuje také doručování zpráv, když je uživatelský agent dočasně offline. Pokud není uživatelský agent v okamžiku odeslání zprávy dostupný, služba push zprávu uloží a doručí ji, jakmile bude uživatelský agent opět dostupný. Tato funkce je klíčová pro zajištění toho, aby uživatelé obdrželi důležité aktualizace nebo oznámení, která byla odeslána v době, kdy byli offline.

Push API lze využívat i v momentě, kdy uživatel aplikaci aktivně používá. Zprávy mají ale obvykle vyšší latenci než přímá komunikace a velikost a počet zpráv může být omezen službou Push. Využití nalezne v případě zřídka se vyskytujících nebo důležitých zpráv, případně pokud je zbytečné udržovat neustálé spojení se serverem. [46]

2.2.4.1 Implementace

Nejprve je potřeba zaregistrovat tzv. Service Worker, což je soubor JavaScriptu, který může řídit webovou stránku, ke které je přidružen, zachycovat a upravovat požadavky na navigaci a zdroje, a ukládat zdroje do mezipaměti, aby bylo možné provádět úlohy na pozadí, jako je příjem push notifikací. [47]

Registrace se provede následovně:

```
if ('serviceWorker' in navigator) {
  navigator.serviceWorker.register("/serviceworker.js")
}
```

V souboru serviceworker.js je poté potřeba přihlásit se k odběru notifikací z push služby přidáním posluchače (event listeneru).

```
self.addEventListener('push', function(event) {
  event.waitUntil(
      self.registration.showNotification('Notification')
  )
})
```

Funkce event.waitUntil je použita proto, aby prohlížeč ponechal service worker aktivní až do doby, kdy je notifikace uživateli zobrazena. [48]

2.2.5 SSE

Server-Sent Events (SSE) je standard, který umožňuje webovému serveru odesílat aktualizace klientovi prostřednictvím protokolu HTTP. Stejně jako WebSocket API je součástí specifikace HTML5 a poskytuje serverům způsob, jak jednoduchým způsobem odesílat aktualizace v reálném čase do webových prohlížečů.

SSE používá jedno dlouhodobé spojení HTTP od klienta k serveru. Toto spojení může následně server používat k odesílání aktualizací klientovi, jakmile jsou k dispozici. Aktualizace jsou odesílány jako textová data a každá aktualizace je oddělena dvojicí znaků nového řádku. [49]

Jednou z klíčových vlastností SSE je schopnost automaticky obnovit spojení, pokud dojde k jeho uzavření, a pokračovat v doručování událostí od místa, kde bylo přerušeno. To je největší výhodou oproti WebSocketům. Naopak však nenabízí plně duplexní komunikaci, ale pouze jednosměrnou ze serveru ke klientovi. [50] Přestože SSE si pro své fungování vystačí s protokolem HTTP, nevyrovná se WebSocketům v oblasti podpory webovými prohlížeči. [51] [52]

2.2.5.1 Implementace

Jak bylo zmíněno, komunikace pomocí SSE je pouze jednosměrná a klient tak může zprávy pouze přijímat. To se děje vytvořením objektu EventSource s adresou serveru a vytvořením funkce, která se spustí v momentě obdržení zprávy ze serveru.

```
let source = new EventSource('http://example.com/sse')
source.onmessage = (event) => {
    console.log(event.data)
}
```

3 SIGNALR

Jednou z výzev při vývoji realtime webových aplikací je zvolit správnou techniku synchronizace pro daný scénář. Pro ASP.NET byla proto vyvinuta open-source knihovna s názvem SignalR, která poskytuje abstrakci nad různými technikami. Zjednodušuje přidávání synchronizačních funkcí do webových aplikací, což umožňuje kódu na straně serveru okamžitě odesílat obsah klientům.

Kódování je stejně jednoduché jako vytváření vzdálených volání procedur. Standardní metody jazyka C# slouží jako metody koncového bodu na serveru. Zadáním stejnojmenných metod na straně klienta a zadáním předpokládaných parametrů se naváže spojení a metoda se aktivuje. Podobně to funguje i v opačném směru, kde klient disponuje posluchači událostí (event listenery). Zachováním přesného názvu posluchače v kódu také na straně serveru a zavedením předpokládaných datových typů jako parametrů se aktivuje klientský posluchač.

SignalR staví na třech technikách realtime komunikace:

- WebSocket API
- Události odesílané serverem (SSE)
- Dlouhé dotazování (long polling)

Automaticky vybírá nejlepší dostupnou metodu na základě možností serveru a klienta. Primárně je využíváno spojení skrze WebSocket API, avšak v případě, že jej z nějakého důvodu použít nelze, bude synchronizace probíhat jinou metodou. Tím je zaručena široká kompatibilita napříč různými platformami a zařízeními. [53]

Podobně jako SSE obsahuje SignalR funkci automatického obnovení spojení, která umožňuje klientovi automaticky obnovit spojení při jeho ztrátě bez nutnosti ručního zásahu. Při ztrátě spojení se klient SignalR automaticky pokusí o obnovení spojení. Pokud se obnovení spojení podaří, klient začne přijímat všechny nové zprávy odeslané serverem. [54]

3.1 SignalR Hub

Středobodem realtime komunikace pomocí SignalR je na straně serveru tzv. hub. Je to třída sdružující metody, které jsou klienty následně vzdáleně volány. Hub se rovněž stará o všechny detaily správy připojení, jako je připojení klienta, události odpojení nebo seskupování připojení.

Hub nemá pouze jednu instanci, ale při každé operaci (například volání metody) prováděné z klienta je vytvořena nová instance třídy. Není tudíž možné ve třídě udržovat stav objektů, protože při každé operaci bude třída vytvořena znovu. Pokud chceme zachovat stav napříč různými huby, je nutné využít jiných mechanismů, jako je databáze, cache paměť nebo paměť na straně serveru (statická proměnná nebo singleton).

3.1.1 Vytvoření

K vytvoření SignalR hubu stačí vytvořit novou třídu, která bude dědit ze třídy Hub, jež je součástí Microsoft.AspNetCore.SignalR.

```
public class MyHub : Hub
{}
```

3.1.2 Konfigurace

Aby však bylo možné vytvořený hub používat, je nutné nejdříve registrovat middleware SignalR v souboru Program.cs zavoláním metody AddSignalR.

```
builder.Services.AddSignalR();
```

Dále je nutné namapovat hub na specifickou cestu URL adresy. Pomocí tohoto endpointu se může klient připojit k hubu prostřednictvím protokolu HTTP. Následný upgrade připojení na protokol WebSocket, případně využití jiné techniky komunikace, se děje na pozadí a my jakožto programátor se nemusíme o nic starat.

```
app.MapHub<MyHub>("/hub");
```

3.1.3 Vlastnosti

Hub obsahuje tři klíčové vlastnosti (property) – Context, Clients a Groups – poskytující funkce a informace ohledně připojení.

3.1.3.1 Context

Objekt Context umožňuje přístup k informacím o připojení, klientovi, požadovaném rozbočovači a volané metodě.

Objekt má k dispozici několik vlastností a metod:

Tabulka 1. Vlastnosti objektu Context

Vlastnosti	**Popis**
ConnectionId	Získá ID aktuálního připojení

UserIdentifier	Získá identifikátor konkrétního uživatele
User	Získá ClaimsPrincipal konkrétního uživatele
Items	Získá kolekci klíčů a hodnot, kterou lze použít ke sdílení dat v rámci připojení
ConnectionAborted	Získá CancellationToken, který informuje o ukončení připojení

Tabulka 2. Metody objektu Context

Metody	**Popis**
GetHttpContext	Vrátí HttpContext připojení
Abort	Ukončí připojení

3.1.3.2 Clients

Clients je dynamický objekt, který umožňuje přístup ke všem připojeným klientům, určitým skupinám klientů nebo i jednotlivým klientům.

Obsahuje následující vlastnosti a metody:

Tabulka 3. Vlastnosti objektu Clients

Vlastnosti	**Popis**
All	Volá metodu na všech připojených klientech
Caller	Volá metodu u klienta, který akci vyvolal
Others	Volá metodu na všech klientech kromě toho, který akci vyvolal

Tabulka 4. Metody objektu Clients

Metody	**Popis**
AllExcept	Volá metodu na všech připojených klientech kromě specifikovaných připojení
Client	Volá metodu na konkrétním připojeném klientovi
Clients	Volá metodu na specifikovaných klientech
Group	Volá metodu na skupině klientů
GroupExcept	Volá metodu na skupině klientů kromě specifikovaných připojení
Groups	Volá metodu na více skupinách
OthersInGroup	Volá metodu pro skupinu připojení s vyloučením klienta, který vyvolal akci

User	Volá metodu pro všechna připojení spojená s konkrétním uživatelem
Users	Volá metodu pro všechna připojení spojená s konkrétními uživateli

3.1.3.3 Groups

Uživatel může navázat několik připojení k dané aplikaci. K tomu může dojít například tehdy, když se uživatel připojí prostřednictvím počítače i mobilního telefonu. Každé z těchto zařízení udržuje nezávislé SignalR připojení k aplikaci, přesto se všechna propojují s jedním uživatelem. Když je zpráva směřována na uživatele, obdrží ji všechna připojení navázaná na tohoto uživatele. Identifikátor uživatele vztahující se ke spojení lze získat pomocí vlastnosti Context.UserIdentifier v rámci hubu.

Skupina je kolekce připojení přidružených k názvu. Skupiny umožňují odesílat zprávy určitým podmnožinám připojených klientů. To je užitečné zejména tehdy, když chceme odeslat zprávu určité skupině uživatelů nebo připojení, ale ne všem připojeným uživatelům. [55]

3.1.4 Eventy

API SignalR hubu poskytuje dvě metody pro správu událostí připojení. Tou první je OnConnectedAsync, která se zavolá při navázání nového spojení s hubem. Naopak při ukončení připojení k hubu je volána metoda OnDisconnectedAsync. V rámci ukončení spojení je připojení také odstraněno ze skupin.

Pokud chceme provádět při těchto událostech vlastní akce, jako třeba přidat připojení do skupiny při navázání spojení, můžeme metody přepsat pomocí override. [56]

```
public override async Task OnConnectedAsync()
{
    await Groups.AddToGroupAsync(Context.ConnectionId, "Users");
    await base.OnConnectedAsync();
}

public override async Task OnDisconnectedAsync(Exception? ex-ception)
{
    await base.OnDisconnectedAsync(exception);
}
```

3.2 Propojení s JavaScript klientem

Microsoft poskytuje balíčky pro Java klienty, .NET a JavaScript.

Tabulka 5. SignalR balíčky pro klienty

Klient	Balíček
.NET	Microsoft.AspNetCore.SignalR.Client
Java	com.microsoft.signalr:signalr
JavaScript	@microsoft/signalr

To umožňuje SignalR komunikovat s Android aplikacemi v Javě, Xamarin, WPF a Win-Form aplikacemi v .NET nebo webovými aplikacemi pomocí JavaScriptu. Ať už jde o Javu, .NET nebo JavaScript, integrace je velmi podobná, proto bude v téhle práci rozebráno pouze propojení s JavaScript klientem, tedy de facto klientem ve webovém prohlížeči.

3.2.1 Instalace balíčku

3.2.1.1 Node Package Manager (NPM)

Správce balíčků NPM byl původně navržen k integraci externích knihoven do aplikací Node.js, který umožňuje spouštění JavaScriptu přímo na hostitelském počítači, nejen v prohlížeči. Node.js se převážně využívá k vytváření aplikací, kde JavaScript zahrnuje kód na straně serveru i klienta. NPM se však vyvinul nad rámec svého původního účelu správy knihoven pro Node.js, protože se ukázalo, že je vhodným prostředkem pro získávání externích závislostí i pro aplikace na straně klienta.

Pro instalaci balíčků do aplikací využívající NPM slouží terminálový příkaz *npm install*. Pomocí něj lze nainstalovat balíček @microsoft/signalr.

```
npm install @microsoft/signalr
```

Tato akce vytvoří adresář s názvem node_modules. V adresáři bude uložen kód všech knihoven používaných původními knihovnami SignalR. Ty pro nás však nejsou podstatné. Důležité je z adresáře node_modules@microsoft/signalr/dist/browser zkopírovat soubor signalr.js nebo signalr.min.js do adresáře lib v projektu. Následně je nutné přidat referenci na daný skript do hlavního html souboru aplikace.

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

3.2.1.2 Content Delivery Network (CDN)

V projektech, které nevyužívají NPM, lze přidat skript hostovaný v rámci CDN.

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/microsoft-sig-
nalr/6.0.1/signalr.js"></script>
```

3.2.2 Připojení k hubu

```
const connection = new signalR.HubConnectionBuilder()
  .withUrl("/myhub")
  .build();
connection.start();
```

Třída HubConnectionBuilder vytváří připojení k SignalR hubu na relativní adrese URL /my-hub. Jakmile je spojení spuštěno, lze jej použít k odesílání a přijímání aktualizací v reálném čase. [54]

3.3 Komunikace

3.3.1 Zasílání zpráv ze serveru

Pro zasílání zpráv klientům ze strany serveru slouží funkce SendAsync nad objekty Clients a Groups uvnitř SignalR hubu. Prvním parametrem funkce je název metody u klienta, která bude na zprávu reagovat. Další parametry jsou přenášená data.

Zaslání zprávy všem uživatelům:

```
public async Task SendMessage(string user, string message)
    => await Clients.All.SendAsync("ReceiveMessage", user, message);
```

Zaslání zprávy pouze uživateli, který akci vyvolal:

```
public async Task SendMessageToCaller(string user, string message)
    => await Clients.Caller.SendAsync("ReceiveMessage", user,
message);
```

Zaslání zprávy skupině uživatelů s názvem „SignalR Users“:

```
public async Task SendMessageToGroup(string user, string message)
    => await Clients.Group("SignalR Users").SendAsync("Recei-
veMessage", user, message);
```

Stejným způsobem se volají metody pro konkrétní uživatele, vybrané skupiny a podobně.

3.3.2 Příjem zpráv u klienta

K příjmu zpráv u klienta je třeba definovat funkci on s názvem a parametry, které bude přijímat.

V následujícím příkladu je funkce s názvem „ReceiveMessage" a přijímá parametry user a message, se kterými následně pracuje.

```
connection.on("ReceiveMessage", (user, message) => {
  console.log("Message from " + user + ": " + message)
})
```

3.3.3 Zasílání zpráv od klienta

Klienti volají veřejné metody hubu prostřednictvím metod invoke a send rozhraní HubConnection. Parametry funkce jsou shodné s voláním metod ze serveru. První parametr (v příkladu "SendMessage") určuje název vzdálené metody a ostatní slouží jako přenášená data.

```
try {
  await connection.invoke("SendMessage", user, message)
} catch (err) {
  console.error(err)
}
```

3.4 Streamování

Streamování je funkce, která umožňuje odesílat nepřetržitý proud dat namísto odesílání dat v jedné odpovědi. To je výhodné ve scénářích, kdy jsou data příliš velká na to, aby byla odeslána v jedné odpovědi, nebo kdy server potřebuje průběžně odesílat data klientovi, jakmile jsou k dispozici.

Při typickém volání metody SignalR vrací metoda serveru jeden kompletní výsledek. Naproti tomu streamovací metoda v SignalR může v průběhu času vracet více výsledků. Například místo čekání na dokončení rozsáhlého databázového dotazu a následného vrácení všech výsledků najednou může streamovací metoda vracet výsledky, jakmile jsou dostupné.

3.4.1 Streamování server-klient

Metoda v hubu se automaticky stává streamovací metodou, pokud vrací IAsyncEnumerable<T>, ChannelReader<T>, Task<IAsyncEnumerable<T>> nebo Task<ChannelReader<T>>.

```
public async IAsyncEnumerable<int> Counter(
int count,
int delay,
[EnumeratorCancellation]
CancellationToken cancellationToken)
{
    for (var i = 0; i < count; i++)
    {
        cancellationToken.ThrowIfCancellationRequested();

        yield return i;
        await Task.Delay(delay, cancellationToken);
    }
}
```

3.4.2 Streamování klient-server

V tomto typu streamování je komunikační stream iniciován ze strany klienta. Klient pokračuje v přenosu zpráv prostřednictvím tohoto streamu, dokud nejsou k dispozici další zprávy k odeslání nebo dokud není stream ukončen. Server tyto zprávy přijímá v přesném pořadí, v jakém byly odeslány. [57] [54]

```
const subject = new signalR.Subject()
yield connection.send("UploadStream", subject)
var iteration = 0;
const intervalHandle = setInterval(() => {
  iteration++
  subject.next(iteration.toString())
  if (iteration === 10) {
    clearInterval(intervalHandle)
    subject.complete()
  }
}, 500)
```

3.5 Alternativy

SignalR je oblíbenou volbou pro komunikaci v reálném čase v aplikacích ASP.NET Core, ale není to jediná možnost.

Jako jednu z alternativ je možné využít WebSocket API. Jak bylo zmíněno, výhodou SignalR je, že není vázán na konkrétní přenosový protokol. Ve výchozím nastavení používá WebSocket, ale pokud klient nebo server WebSocket nepodporuje, automaticky se vrátí k jiným

protokolům, jako jsou SSE nebo long polling. Tato abstrakce umožňuje vývojářům psát funkce v reálném čase, aniž by se museli starat o základní transportní protokol. Implementace samotného WebSocket API by tak byla zbytečně náročnější a bez přidaných hodnot. [58]

Obdobnou možností nahrazení SignalR se jeví protokol SSE. Je méně složitý než WebSocket, ale podporuje pouze jednosměrnou komunikaci (ze serveru na klienta). Také navíc podporuje obnovení spojení, které je ovšem zahrnuto i v SignalR. Rovněž tím pádem nenabízí pádné důvody, proč jej upřednostnit před knihovnou SignalR.

Pokud tedy hledáme nejlepší řešení pro realtime synchronizaci ve webových aplikací postavených na ASP.NET Core, SignalR poskytuje komplexní balíček funkcí, které těžko hledají alternativu. Jelikož je produktem společnosti Microsoft, je bezproblémově integrován do celého ekosystému .NET Core a využívá tak výhod výkonnosti, zabezpečení a škálovatelnosti ASP.NET Core. [59]

4 POUŽITÉ TECHNOLOGIE

4.1 Backend

Podle zadání byl k tvorbě serverové části aplikace využit framework ASP.NET Core s knihovnou SignalR.

4.1.1 ASP.NET Core

.NET Core je bezplatný multiplatformní framework vyvíjený společností Microsoft s otevřeným zdrojovým kódem pro vytváření různých typů aplikací. Jedná se o modulární framework, který běží jak na plném prostředí .NET Framework v systému Windows, tak na multiplatformním prostředí .NET Core na systémech Linux nebo macOS.

Jedním z hlavních rysů .NET Core je jeho modularita. Funkce jsou poskytovány prostřednictvím balíčků NuGet, což umožňuje vývojářům přidávat do svých aplikací pouze ty funkce, které skutečně potřebují. To vede k větší efektivitě a menším požadavkům na výkon. Výhodou jsou také různé možnosti nasazení – může být nasazen na Windows, Linux nebo macOS a podporuje kontejnery Docker, což ulehčuje škálování a distribuci aplikací.

ASP.NET Core je framework pro vývoj webových aplikací, který běží na platformě .NET Core a poskytuje způsob vytváření webových aplikací a rozhraní API v jazyce C#. Je následovníkem ASP.NET, ale je kompletně přepracovaný a vylepšený s ohledem na současné požadavky webového vývoje. Má v sobě integrovanou podporu pro vývoj REST API a také podporuje vytváření realtime webových aplikací pomocí SignalR. [60] [61]

4.2 Frontend

Pro tvorbu klientské části se nabízí spousta možností. Nejjednodušší se jeví využití frameworku Blazor, který je rovněž součástí ASP.NET Core a umožňuje vytváření webových aplikací pomocí jazyka C#. Celá webová aplikace by tak mohla být zpracována v rámci jednoho projektu. Dělí se na dva modely – Blazor Server a Blazor WebAssembly.

Blazor Server aplikace běží na serveru, který zároveň zpracovává všechny interakce uživatelského rozhraní prostřednictvím připojení SignalR. To znamená, že všechny interakce s uživatelem, aktualizace DOM, zpracování událostí a interop volání JavaScriptu se posílají po síti a vlastní provádění akce probíhá na serveru. Tento model tudíž nevyžaduje

dodatečnou implementaci SignalR, na kterém rovnou staví, proto by nebyl pro tuto práci vhodnou ukázkou implementování knihovny SignalR.

Druhý model WebAssembly do prohlížeče stáhne Blazor aplikaci, její závislosti a runtime prostředí .NET. Aplikace je spuštěna přímo v bajtovém kódu WebAssembly prohlížeče. To umožňuje spuštění aplikace offline a snižuje zatížení serveru, ale je spojeno s větší velikostí stahování, což má za následek pomalejší počáteční načítání. [62] [63] Pro integraci SignalR je tedy jako příklad vhodnější.

V téhle práci však dostal přednost framework Angular, který je v porovnání s Blazor WebAssembly dospělejší, jelikož je na trhu mnohem delší dobu, což v praxi znamená, že má větší komunitu, více knihoven třetích stran a více dostupných výukových zdrojů. Aplikace v něm jsou obvykle menší a rychlejší než aplikace Blazor WebAssembly, protože Angular používá JavaScript, který se spouští přímo v prohlížeči, zatímco Blazor WebAssembly musí spolu s aplikací stáhnout runtime prostředí .NET. Také je díky JavaScriptu univerzálnější. [64]

4.2.1 Angular

Angular je open-source framework pro vývoj webových aplikací, který byl vyvinut a je udržován společností Google. Byl poprvé představen v roce 2010 jako AngularJS a od verze 2 je znám jednoduše jako Angular. V této práci byla využita verze 14.

Angular je navržen pro vývoj jednostránkových aplikací (SPA) a poskytuje kompletní řešení pro klientskou stranu aplikace, včetně šablon, datové vazby, testování, modulární architektury a AJAX manipulace. Také poskytuje silnou podporu pro moderní webové technologie, jako je komponentový model pro rozhraní a TypeScript pro psaní kódu.

Komponenty v Angularu jsou základními stavebními bloky webových aplikací vytvořených v tomto frameworku. Každá komponenta se skládá ze tří částí: programové třídy, HTML šablony a CSS stylů. Programová třída definuje data a logiku komponenty, HTML šablona určuje její obsah a CSS styly upravují její vizuální prezentaci. Komponenty mohou být vnořené do sebe a tvořit tak hierarchickou strukturu aplikace. Komponenty také mohou využívat služby, které poskytují další funkce, například komunikaci se serverem nebo sdílení dat mezi komponentami.

K organizaci a strukturování kódu aplikace se používají moduly. Jsou to logické celky, které usnadňují správu kódu a jeho znovupoužitelnost. Moduly mohou být importovány nebo exportovány mezi sebou a vytvářet tak hierarchii a závislosti. [65] [66]

4.2.2 PrimeNG

Pro usnadnění práce využitím předvytvořených komponent byla využita knihovna PrimeNG. Je to open-source knihovna vyvinutá firmou PrimeTek s cílem poskytnout kompletní sadu komponent pro tvorbu bohatých a interaktivních webových aplikací v Angularu. Obsahuje velké množství komponent, včetně formulářů, tlačítek, datových tabulek, dialogů, panelů, menu a mnoho dalšího. [67]

4.2.3 NGXS

NGXS je moderní knihovna pro správu stavu v Angular aplikacích. Je inspirována Reduxem, populárním nástrojem pro správu stavu v React aplikacích, ale je specificky navržena tak, aby dobře zapadla do Angularu a využívala jeho specifické funkce a konvence.

NGXS usnadňuje udržování a manipulaci se stavem aplikace tím, že poskytuje jednoduché a jednotné rozhraní pro interakci se stavem. Zahrnuje koncepty, jako jsou akce, stavy a selektory, které umožňují vytvářet přehledný a snadno testovatelný kód. Je oblíbena pro svou jednoduchost a přímočarost, která je často kontrastem k jiným, složitějším řešením pro správu stavu. NGXS se snaží minimalizovat boilerplate kód (často opakovaný kód s minimální nebo žádnou změnou) a poskytnout intuitivní API pro vývojáře. [68]

4.3 Hosting

4.3.1 Microsoft Azure

Pro publikaci a hostování vytvořené aplikace byl zvolen Microsoft Azure. Je to cloudová platforma, která nabízí řadu služeb a produktů pro různé potřeby a scénáře. Některé z nich jsou zdarma. Například, pokud chceme vytvořit a testovat jednoduchou webovou aplikaci, můžeme využít bezplatný účet Azure, který nám poskytne na 12 měsíců zdarma některé populární služby, jako jsou Azure App Service, Azure Functions nebo Azure SQL Database. Také je k dispozici 200 USD kreditu na prvních 30 dní, který lze použít na jakoukoli službu. Po uplynutí těchto období jsou využívané služby zpoplatněny. [69]

4.4 Použitý software

4.4.1 JetBrains Rider

Při tvorbě backendu aplikace byl uplatněn software Rider. Je to integrované vývojové prostředí (IDE) vyvinuté společností JetBrains. Kromě jazyka C# použitý pro vývoj v ASP.NET Core podporuje různé frameworky a jazyky – .NET, Xamarin, JavaScript, TypeScript a další. Je postaven na platformě IntelliJ IDEA, která pohání další populární IDE od JetBrains, jako je samotné IntelliJ IDEA pro Javu, PyCharm (Python) nebo WebStorm (JavaScript).

Rider obsahuje mnoho funkcí nástroje ReSharper, dalšího produktu společnosti JetBrains, který je oblíbeným doplňkem pro Visual Studio. ReSharper poskytuje řadu funkcí pro zvýšení produktivity vývojářů v prostředí .NET, například kontrolu kódu, refaktorizaci, vylepšení navigace a spoustu dalšího. Výhodou je taktéž integrace systému pro správu verzí Git nebo integrace s cloudovou platformou Azure. [70]

4.4.2 Visual Studio Code

Pro vývoj frontendové části aplikace bylo využito Visual Studio Code. Jedná se o vývojové prostředí určené pro psaní kódu a tvorbu softwaru. Je vybaveno množstvím užitečných nástrojů a rozšíření, které usnadňují programování a správu rozsáhlých projektů.

Díky řadě výhod je momentálně VS Code jedním z nejčastěji používaných editorů mezi programátory. [71] Jednou z nejvýznamnějších je jeho rozšiřitelnost. VS Code podporuje mnoho rozšíření, které mohou uživatelé snadno instalovat a tím přizpůsobit editor svým individuálním potřebám a stylu práce. Tato rozšíření zahrnují různé jazykové balíčky, ladicí nástroje, nástroje pro správu kódu, formátování kódu a mnoho dalších. [72]

Pro tuto práci byly použity následující rozšíření:

- Angular Language Service
- Azure App Service

4.4.3 Figma

Návrh uživatelského rozhraní a vzhledu aplikace byl vytvořen v cloudové designové platformě Figma, která uživatelům umožňuje vytvářet, spolupracovat a sdílet interaktivní prototypy. Podporuje různé funkce návrhu, ať už jde o vektorovou grafiku, wireframe modely, komponenty, styly nebo animace. Figma také nabízí nástroje pro spolupráci v reálném čase,

které umožňují více uživatelům pracovat na stejném projektu současně a komunikovat prostřednictvím komentářů a zpětné vazby. [73]

II. PRAKTICKÁ ČÁST

5 NÁVRH APLIKACE

Pro demonstraci implementace synchronizace v reálném čase v ASP.NET Core s využitím knihovny SignalR byla zvolena webová aplikace, respektive online hra, typu Draw & Guess. Cílem této aplikace není konkurovat stávajícím hrám na stejné téma, které již existují. Zmínit lze například skribbl.io, gartic.io nebo drawize.com. Hra tohoto typu byla zvolena z důvodu, že využívá realtime synchronizaci na hned několika místech k synchronizaci různých dat a je pro ni klíčová.

5.1 Princip hry

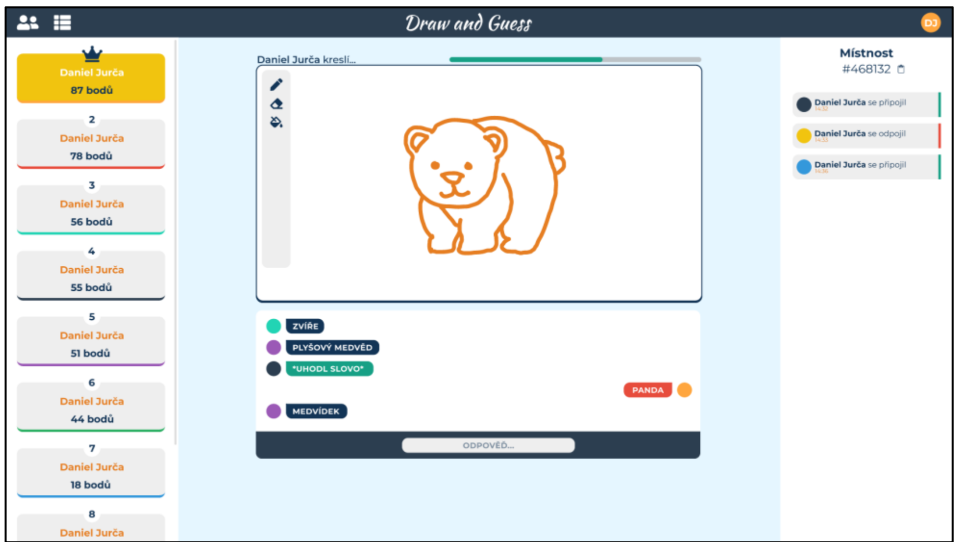
Princip hry je poměrně jednoduchý. Jak už název napovídá, jde o kreslení obrázků a jeho hádání. Hra se hraje v minimálním počtu dvou hráčů, kdy jeden z nich má za úkol nakreslit obrázek podle zadání, které zná pouze on, a ostatní hráči mají za cíl uhodnout, o co konkrétně se jedná. Pokud některý hráč zadané slovo uhodne, připíše si body společně s hráčem, který kreslí (ten však v nižším počtu). Výše získaných bodů pro hádající hráče se odvíjí od pořadí, ve kterém uhodnou. Pokud tedy hráč slovo uhodne jako první, získává maximální počet bodů. Hraje se na několik kol a role kreslíře se spravedlivě střídá. Výhercem hry je hráč, který jako první dosáhne cílového skóre (například 100 bodů).

5.2 Synchronizace

Středobodem je synchronizace samotné kresby, která musí být okamžitě k dispozici všem hráčům připojeným v dané hře. Kromě toho je potřeba aktualizovat seznam připojených hráčů, jejich skóre a roli, odpovědi, či samotné akce serveru jako začátek nového kola. V neposlední řadě by měl být synchronizovaný seznam aktivních her, do kterých se může hráč připojit.

5.3 Návrh uživatelského rozhraní

V aplikaci Figma bylo navrženo rozhraní aplikace, které se skládá ze čtyřech hlavních částí – horní lišty, dvou bočních panelů a samotné herní plochy.



Obrázek 7. Návrh rozhraní v aplikaci Figma

5.3.1 Horní lišta

V levé části lišty se nachází dvě tlačítka pro zobrazení nebo skrytí postranních panelů aplikace. Uprostřed je logo, respektive název hry. V pravé části je pak ikona přihlášeného uživatele.

5.3.2 Levý panel

Na levém panelu je umístěna výsledková tabulka. Jsou zde zobrazeni všichni hráči připojeni v dané místnosti (hře). Každý hráč je rozlišen barevným spodním okrajem, a kromě jména je u něj uveden i počet bodů (skóre) a jeho pozice, podle které jsou hráči v seznamu seřazeni. Hráč na první pozici je barevně zvýrazněn a namísto čísla pozice je vyobrazena ikona koruny.

5.3.3 Pravý panel

Obsahem pravého panelu je název a identifikační číslo místnosti, které bude možné tlačítkem vedle zkopírovat. Níže se nachází historie místnosti, kde se eviduje připojení a odpojení hráčů. Každý z těchto záznamů má uvedený čas a typ akce je barevně odlišen pravým okrajem.

5.3.4 Herní plocha

Součástí herní plochy jsou tři prvky – status bar, kreslicí plátno a chat pro odpovědi. Status bar slouží pro jednoduché zobrazení aktuálního stavu hry a zbývajícího času skrze progres bar. Hned pod ním následuje kreslicí plátno s nástroji pro kreslení – pero, guma a kyblík.

Posledním prvkem je chat, který slouží pro hádání kresleného obrázku. Vlevo jsou zarovnány zprávy ostatních hráčů a napravo zprávy vlastní. Správnost tipu symbolizuje barva bubliny zprávy. Pokud hráč slovo uhodne, ostatním hráčům je zobrazen pouze placeholder „*uhodl slovo*“.

6 BACKEND

6.1 Vytvoření projektu

Projekt byl vytvořen na šabloně ASP.NET Core Web API, která je v IDE Rider v základu k dispozici. Verze .NET frameworku byla zvolena 7.0. Šablona Web API v projektu vytvoří ukázkové rozhraní WeatherForecast s podporou Swagger, který pro API generuje dokumentaci na adrese /swagger/index.html.

Pro vnitřní uspořádání projektu byly v kořenové složce vytvořeny adresáře Hubs, Models, Services a Resources.

V adresáři Hubs byla založena nová třída DrawHub, která dědí ze třídy Hub ze SignalR. V Program.cs byla následně přidána služba SignalR a zaregistrován vytvořený hub.

```
builder.Services.AddSignalR();
app.MapHub<DrawHub>("/hub");
```

Byla nakonfigurována služba CORS a povolena adresa pro klientskou aplikaci.

```
builder.Services.AddCors(options =>
{
    options.AddPolicy("CorsPolicy", policy =>
    {
        policy.AllowAnyHeader()
            .AllowAnyMethod()
            .WithOrigins("http://localhost:4200")
            .AllowCredentials();
    });
});
```

6.2 Logika hry

6.2.1 Třídy

6.2.1.1 Player

Třída Player modeluje hráče hry. Vlastnostmi třídy jsou unikátní identifikátor, ID SignalR spojení, jméno, barva a skóre. Třída má jak konstruktor bez parametrů, tak konstruktor s parametry, který inicializuje vlastnosti. Dále má metodu AddPoints pro přičtení bodů ke skóre hráče a metodu ResetScore pro vynulování skóre.

```
public class Player
{
    public Guid Id { get; set; }
    public string ConnectionId { get; set; }
    public string Username { get; set; }
    public string Color { get; set; }
    public int Score { get; set; }

    public Player() {}
    public Player(string connectionId, string username, string color)
    {
        ConnectionId = connectionId;
        Id = Guid.NewGuid();
        Username = username;
        Color = color;
        Score = 0;
    }

    public void AddPoints(int points)
    {
        Score += points;
    }

    public void ResetScore()
    {
        Score = 0;
    }
}
```

6.2.1.2 Answer

Pokud hráč odešle na server zprávu se slovem při hádání obrázku, je vytvořen objekt z modelu třídy Answer. Ta kromě unikátního ID a hádaného textu obsahuje informace o hráči a pravdivostní hodnotu, zda je tip správný. K inicializaci objektu slouží parametrický konstruktor. Třída záměrně neuchovává objekt hráče, aby i při odpojení a odstranění hráče byly údaje o něm k dispozici. Pokud se tedy do hry připojí nový hráč a načte si historii odpovědí, uvidí i odpovědi hráčů, kteří již ve hře nejsou.

```
public class Answer
{
    public Guid Id { get; set; }
```

```
    public string Username { get; set; }
    public string Color { get; set; }
    public string Message { get; set; }
    public bool IsCorrect { get; set; }

    public Answer(Guid id, string username, string color, string
message, bool isCorrect)
    {
        Id = id;
        Username = username;
        Color = color;
        Message = message;
        IsCorrect = isCorrect;
    }
}
```

6.2.1.3 LogRecord

Třída LogRecord modeluje záznamy pro historii hry, jako je připojení nebo odpojení hráče. Podobně jako třída Answer neobsahuje referenci na objekt hráče, ale jeho údaje ukládá přímo do vlastností. Akci záznamu specifikuje výčtový typ LogEvent. Ten obsahuje dvě hodnoty:

- Connected – připojení hráče
- Disconnected – odpojení hráče

```
public class LogRecord
{
    public Guid UserId { get; set; }
    public string Username { get; set; }
    public string Color { get; set; }
    public EventType LogEvent { get; set; }
    public DateTime Time { get; set; }

    public LogRecord(Guid userId, string username, string color,
EventType logEvent)
    {
        UserId = userId;
        Username = username;
        Color = color;
        LogEvent = logEvent;
        Time = DateTime.Now;
```

```
        }
    }

    public enum EventType
    {
        Connected,
        Disconnected
    }
```

6.2.1.4 Word

Třída Word slouží pro výběr slova, které bude hráč, který je právě na řadě, kreslit. Slova (případně slovní spojení) jsou uložena v poli v souboru WordList.json, odkud jsou pomocí StreamReaderu vytažena a následně deserializována do listu řetězců. K deserializaci byl vy-užit JsonSerializer.

```
    public class Word
    {
        public string Value { get; set; }
        private List<string> _words;

        public Word()
        {
            using (StreamReader r = new StreamReader("Re-
    sources/WordList.json"))
            {
                string json = r.ReadToEnd();
                _words = JsonSerializer.Deserialize<List<string>>(json);
            }
            Value = _words[Random.Shared.Next(_words.Count)];
        }

        public Word Next() => new() { Value = _words[Random.Sha-
    red.Next(_words.Count)] };
    }
```

Obrázek 8. Seznam slov v souboru WordList.json

6.2.1.5 Game

Hlavní třídou pro logiku hry je třída Game. Ta reprezentuje jednotlivé hry vytvořené uživateli. Obsahuje vlastnosti, které se nastavují na základě uživatelského vstupu – jméno a hodnotu IsPrivate, která značí, zda bude hra skryta v seznamu dostupných her a jediná možnost pro připojení bude pomocí odkazu nebo ID. Tyto hodnoty se k vytvoření objektu předávají v konstruktoru. Dále obsahuje seznam připojených hráčů, list záznamů LogRecords, referenci na aktuálně kreslícího hráče, aktuální kreslené slovo, cílové skóre pro konec hry nebo aktuální bodový zisk pro hráče, který slovo uhodne. Pravdivostní hodnota IsInProgress označuje, zda právě probíhá herní kolo.

GUID uživatelů, kteří v aktuálním kole již slovo uhodli, se ukládá do zvláštního listu. Hráč, který kreslí, dostává bod za každé uhodnutí ostatních hráčů. Úkolem tohoto listu je zamezit zisku bodu od hráče, který uhodne slovo, odpojí se ze hry, znovu se do ní připojí a uhodne znovu slovo, které již zná, v rámci stejného kola. Poslední vlastností je seznam odpovědí.

Třída obsahuje také dva privátní fieldy: _roundTimer, což je časovač, který se používá k řízení délky každého kola, a _roundStartTime, představující čas zahájení časovače.

Tato třída také obsahuje několik eventů: NewRoundEvent, RoundEndEvent, BreakEvent, WaitingForPlayersEvent, EndOfGameEvent. Tyto události se spouštějí v různých okamžicích hry, například když začíná nebo končí nové kolo nebo když hra čeká na připojení hráčů.

```
using System.Timers;

public class Game : IDisposable
{
    public int Id { get; }
    public string Name { get; set; }
    public bool IsPrivate { get; set; }
```

```
    public IList<Player> Players { get; }
    public IList<LogRecord> LogRecords { get; set; }
    public Player CurrentDrawer { get; set; }
    public Word CurrentWord { get; set; }
    public int ScoreTarget { get; set; }
    public int ScoreGain { get; set; }
    public bool IsInProgress { get; set; }
    public IList<Guid> PlayersGuessed { get; set; }
    public Status Status { get; set; }
    public IList<Answer> Answers { get; set; }

    private Timer _roundTimer;
    private DateTime _roundStartTime;


    public event EventHandler<GameEventArgs> NewRoundEvent;
    public event EventHandler<GameEventArgs> RoundEndEvent;
    public event EventHandler<GameEventArgs> BreakEvent;
    public event EventHandler<GameEventArgs> WaitingForPlayersEvent;
    public event EventHandler<GameEventArgs> EndOfGameEvent;

    public Game(int id, string name, bool isPrivate = false)
    {
        Id = id;
        Name = name;
        IsPrivate = isPrivate;
        ScoreTarget = 120;
        Players = new List<Player>();
        LogRecords = new List<LogRecord>();
        PlayersGuessed = new List<Guid>();
        Status = Status.WaitingForPlayers;
        Answers = new List<Answer>();
    }
```

Ve třídě byly implementovány metody pro přidání a odebrání hráče ze seznamu, výběr hráče pro kreslení nebo metody pro řízení průběhu a stavu hry.

6.2.2 Stavy hry

Hra má čtyři stavy:

- Čekání na hráče
- Hra
- Pauza
- Konec hry

Výchozí stav při vytvoření hry je vyčkávání na hráče. Pokud se do hry připojí alespoň 2 hráči, spustí se pětisekundová pauza před započetím herního kola.

```
private void StartBreak()
{
    Status = Status.Break;

    if (_roundTimer == null)
    {
        _roundTimer = new Timer();
        _roundTimer.Elapsed += OnRoundTimerElapsed;
    }

    _roundTimer.Interval = 5 * 1000;
    _roundTimer.Start();
    _roundStartTime = DateTime.Now;

    BreakEvent?.Invoke(this, new GameEventArgs(this));
}
```

Po vypršení intervalu je zavolána funkce, která na základě posledního stavu hry vybere stav nový. Pokud se právě dohrálo kolo, spustí se pauza, případně konec hry, pokud některý z hráčů dosáhl cílového skóre. V ostatních případech se spustí nové kolo.

```
private void OnRoundTimerElapsed(object sender, ElapsedEventArgs e)
{
    _roundTimer.Stop();
    EndRound();
    RoundEndEvent?.Invoke(this, new GameEventArgs(this));
}

private void EndRound()
{
    IsInProgress = false;
    PlayersGuessed = new List<Guid>();
```

```
        if (Status == Status.Drawing)
        {
            if (Players.Any(p => p.Score >= ScoreTarget))
            {
                EndGame();
                return;
            }

            StartBreak();
            return;
        }

        StartNewRound();
    }
```

6.2.3 GameManager

GameManager je třída registrovaná jako singleton sloužící ke správě všech existujících her, které má uloženy v kolekci. Obsahuje metody pro vytváření a mazání her, připojování, získání listu hráčů nebo záznamů a další, které jsou následně využívány ve webové API a SignalR hubu.

```
    public class GameManager
    {
        private readonly ConcurrentDictionary<int, Game> _games = new();

        private readonly IHubContext<DrawHub> _hubContext;
        private readonly IMapper _mapper;

        public GameManager(IHubContext<DrawHub> hubContext, IMapper map-
    per)
        {
            _hubContext = hubContext;
            _mapper = mapper;
        }

        public Game CreateGame(string name, bool isPrivate)
        {
            if (name == null || name == "") name = "Draw & Guess";
            var game = new Game(GenerateId(), name, isPrivate);
```

```
        _games.TryAdd(game.Id, game);

        game.NewRoundEvent += OnNewRoundEvent;
        game.RoundEndEvent += OnRoundEndEvent;
        game.EndOfGameEvent += OnEndOfGameEvent;
        game.BreakEvent += OnBreakEvent;
        game.WaitingForPlayersEvent += OnWaitingForPlayersEvent;

        return game;
    }

    public Game FindGameById(int gameId)
    {
        _games.TryGetValue(gameId, out var game);
        return game;
    }

    public bool JoinGame(int gameId, Player player)
    {
        var game = FindGameById(gameId);

        if (game == null) return false;
        if (game.Players.Count >= 8) return false;

        return game.AddPlayer(player);
    }
```

Při vytvoření nové hry jsou zaregistrovány také obslužné funkce, které se spouští v případě vyvolání eventů změny stavu. Díky dependency injection může GameManager využívat kontext SignalR hubu skrze instanci IHubContext<DrawHub> a zasílat aktualizace uživatelům. Toho využívají právě obslužné funkce eventů pro zaslání nového stavu hry nebo aktualizaci seznamu hráčů.

```
private void OnNewRoundEvent(object? sender, Game.GameEventArgs e)
{
    var status = e.Game.GetStatusDto();
    _hubContext.Clients.Group(e.Game.Id.ToString()).SendAsync(Sig-
nalRMethods.Status, status);
    _hubContext.Clients.Group(e.Game.Id.ToString()).SendAsync(Sig-
nalRMethods.Drawer, e.Game.CurrentDrawer.Id);
```

```
    _hubContext.Clients.Client(e.Game.CurrentDrawer.Connectio-
nId).SendAsync(SignalRMethods.Word, e.Game.CurrentWord.Value);


    var players = _mapper.Map<List<PlayerDto>>(GetPlay-
ers(e.Game.Id).OrderByDescending(p => p.Score).ToList());

    _hubContext.Clients.Groups(e.Game.Id.ToString()).SendAsync(Sig-
nalRMethods.UpdatePlayers, players);

}


private void OnWaitingForPlayersEvent(object? sender, Game.GameEven-
tArgs e)

{

    var status = e.Game.GetStatusDto();

    _hubContext.Clients.Group(e.Game.Id.ToString()).SendAsync(Sig-
nalRMethods.Status, status);

    _hubContext.Clients.Group(e.Game.Id.ToString()).SendAsync(Sig-
nalRMethods.Drawer, "");

}
```

6.2.4 PlayerManager

Třída PlayerManager je obdobou třídy GameManager, ovšem namísto správy her se stará o hráče.

```
public class PlayerManager

{

    private readonly ConcurrentDictionary<string, Player> _players =
new();


    public Player CreatePlayer(string connectionId, string username,
string color)

    {

        if (_players.ContainsKey(connectionId))

        {

            return _players[connectionId];

        }


        var player = new Player(connectionId, username, color);

        _players.TryAdd(player.ConnectionId, player);

        return player;

    }


    public Player GetPlayer(string connectionId)

    {
```

```
            _players.TryGetValue(connectionId, out var player);
            return player;
        }

        public bool DeletePlayer(string connectionId)
        {
            return _players.TryRemove(connectionId, out _);
        }
    }
```

6.3 Web API

Aplikace využívající SignalR ke komunikaci mezi serverem a klientem nemusí využívat pouze tuto knihovnu, ale je možné ji kombinovat společně s rozhraním REST API. Proto i zde byly vytvořeny kontrolery, které definují metody pro požadavky, které není nutné provádět skrze SignalR.

6.3.1 LobbyController

Tento kontroler je určen k vytvoření API metod pro stránku se seznamem her na straně klienta.

[ApiController], *[Produces("application/json")]*, *[Route("api/games")]* jsou atributy, které určují, že tato třída je kontroler, její metody budou vytvářet odpovědi JSON a jsou dostupné na adrese api/games.

Jsou zde dvě HTTP metody. Metoda GET slouží k vyžádání seznamu her a metoda POST pro vytvoření nové hry. POST metoda se nachází na endpointu api/games/create.

```
    [ApiController]
    [Produces("application/json")]
    [Route("api/games")]
    public class LobbyController : ControllerBase
    {
        private readonly GameManager _gameManager;
        private readonly IMapper _mapper;

        public LobbyController(GameManager gameManager, IMapper mapper)
        {
            _gameManager = gameManager;
            _mapper = mapper;
```

```
        }

        [HttpGet(Name = "GetGames")]
        public IEnumerable<GameLobbyDto> GetGames()
        {
            return _mapper.Map<List<GameLobbyDto>>(
                _gameManager.GetGames()
                    .Where(g => g.IsPrivate == false)
                    .OrderByDescending(g => g.Players.Count)
                    .ToList()
            );
        }

        [HttpPost("create")]
        public int CreateGame(CreateGameRequest request)
        {
            var game = _gameManager.CreateGame(request.Name, request.Pri-
    vate);
            return game.Id;
        }
    }
```

6.3.2 RoomController

Druhý kontroler byl vytvořen pro definování metod pro konkrétní hru s adresou api/game. Obsahuje metodu GetGameInfo typu GET, jež slouží k získání dat ze hry, která je specifikována pomocí ID zaslaným v adrese požadavku.

```
    [ApiController]
    [Produces("application/json")]
    [Route("api/game")]
    public class RoomController : ControllerBase
    {
        private readonly GameManager _gameManager;
        private readonly IMapper _mapper;

        public RoomController(GameManager gameManager, IMapper mapper)
        {
            _gameManager = gameManager;
            _mapper = mapper;
        }
```

```
        [HttpGet("{id:int}")]
        public IActionResult GetGameInfo(int id)
        {
            var gameInfo = _mapper.Map<GameInfoDto>(_gameManager.FindGa-
    meById(id));

            var status = _gameManager.GetStatusDto(id);
            gameInfo.Status = status;
            return Ok(gameInfo);
        }
    }
```

6.3.3 Data Transfer Object

Doménové modely často obsahují více dat, než chceme odeslat ke klientovi. DTO (Data Transfer Object) jsou modely, které obsahují pouze ty vlastnosti, které chceme odeslat. Příkladem může být metoda pro získání dostupných her, kde je nevhodné posílat celé objekty, ale stačí pouze ID, název a počet hráčů. Model pro tento požadavek pouze s těmito vlastnostmi byl vytvořen ve třídě s názvem GameLobbyDto a podobně byly vytvořeny DTO modely pro ostatní třídy.

```
    public class GameLobbyDto
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public int PlayerCount { get; set; }
    }
```

Pro zjednodušení transformace z jednoho objektu na druhý, jako je potřeba provést zde z doménového objektu na DTO, byla využita knihovna AutoMapper. AutoMapper automaticky zpracovává kopírování sdílených vlastností z jednoho objektu do druhého, čímž snižuje množství ručně psaného kódu. Tyto transformace byly definovány ve třídě MappingProfile.

```
    public class MappingProfile : Profile
    {
        public MappingProfile()
        {
            CreateMap<Game, GameLobbyDto>()
                .ForMember(
                    dest => dest.PlayerCount,
```

```
                opt => opt.MapFrom(src => src.Players.Count)
            );

        CreateMap<Game, GameInfoDto>()
            .ForMember(
                dest => dest.DrawerId,
                opt => opt.MapFrom(src => src.CurrentDrawer.Id)
            )
            .ForMember(
                dest => dest.Status,
                opt => opt.Ignore()
            );

        CreateMap<Player, PlayerDto>();

        CreateMap<LogRecord, LogRecordDto>();

        CreateMap<Answer, AnswerDto>();
    }
}
```

6.4 SignalR Hub

Aby v hubu bylo možné spravovat uživatele a jednotlivé hry, jsou pomocí dependency injection přidány služby PlayerManager a GameManager. Rovněž je využíván AutoMapper pro mapování doménových objektů na DTO.

```
public class DrawHub : Hub
{
    private readonly GameManager _gameManager;
    private readonly PlayerManager _playerManager;
    private readonly IMapper _mapper;

    private string ConnectionId => Context.ConnectionId;

    public DrawHub(GameManager gameManager, PlayerManager playerMa-
nager, IMapper mapper)
    {
        _gameManager = gameManager;
        _playerManager = playerManager;
        _mapper = mapper;
```

```
        }
    }
```

6.4.1 Názvy metod

Názvy metod, které jsou implementovány na straně klienta, jsou definovány v konstantních proměnných, aby se zamezilo překlepům v názvech při ručním psaní do jednotlivých SignalR funkcí.

```
public class SignalRMethods
{
    public const string UpdatePlayers = "UpdatePlayers";
    public const string UpdateLobby = "UpdateLobby";
    public const string NewLogMessage = "NewLogMessage";
    public const string BeginPath = "BeginPath";
    public const string Draw = "Draw";
    public const string Answer = "Answer";
    public const string GetDrawing = "GetDrawing";
    public const string SendDrawing = "SendDrawing";
    public const string Status = "Status";
    public const string Drawer = "Drawer";
    public const string Word = "Word";
}
```

6.4.2 Implementace metod

6.4.2.1 Přihlášení

Vzhledem k faktu, že aplikace nevyžaduje složité přihlašování heslem a uložení citlivých údajů, je přihlašování řešeno jednoduchou SignalR metodou, která vytvoří nového uživatele a uloží v něm ID jeho připojení. Pro přihlášení musí klient zavolat metodu Login s jeho uživatelským jménem a zvolenou barvou charakteru. Jako odpověď mu je zaslán objekt jeho hráče.

```
public Task<PlayerDto> Login(string name, string color)
{
    var player = _playerManager.CreatePlayer(ConnectionId, name,
color);
    var playerDto = _mapper.Map<PlayerDto>(player);
    return Task.FromResult(playerDto);
}
```

6.4.2.2 Připojení do hry

Pro připojení do hry klient volá metodu JoinRoom s parametrem ID hry. Ze seznamu hráčů je pomocí ID připojení získán objekt hráče, který metodu vyvolal. Tento objekt pak společně s ID hry slouží jako parametry pro metodu JoinGame objektu _gameManager.

Pokud se připojení nezdaří, je uživateli vrácena hodnota false. Pokud zdaří, je spojení uživatele přidáno do skupiny, jejíž název tvoří ID hry převedeno do textového řetězce. Následně je vytvořen nový záznam v podobě objektu LogRecord a zaslán uživatelům připojeným v dané hře. Stejně tak je těmto uživatelům zaslán aktualizovaný seznam připojených hráčů, který již nově obsahuje připojovaného hráče. Nakonec je všem uživatelům zaslána notifikace, která je informuje o změně v seznamu her, jelikož se změnil počet hráčů v jedné z nich. V případě úspěchu vrací funkce hodnotu true.

```
public async Task<bool> JoinRoom(int gameId)
{
    var player = _playerManager.GetPlayer(ConnectionId);
    var joined = _gameManager.JoinGame(gameId, player);
    if (!joined) return false;

    await Groups.AddToGroupAsync(ConnectionId, gameId.ToString());

    var logRecord = new LogRecord(player.Id, player.Username, pla-
yer.Color, EventType.Connected);
    _gameManager.AddLogRecord(gameId, logRecord);
    var logRecordDto = _mapper.Map<LogRecordDto>(logRecord);
    await Clients.Group(gameId.ToString()).SendAsync(SignalRMet-
hods.NewLogMessage, logRecordDto);

    var players = GetPlayerList(gameId);
    await Clients.OthersInGroup(gameId.ToString()).SendAsync(SignalR-
Methods.UpdatePlayers, players);

    await Clients.All.SendAsync(SignalRMethods.UpdateLobby);

    return true;
}
```

6.4.2.3 Odpojení ze hry

Velmi podobně funguje také funkce pro opuštění hry. Narozdíl od připojení do hry je však uživatel ze hry i ze skupiny odebrán.

```
public async Task LeaveRoom(int gameId)
{
    var player = _playerManager.GetPlayer(ConnectionId);
    var left = _gameManager.LeaveGame(gameId, player);
    if (!left) return;

    await Groups.RemoveFromGroupAsync(ConnectionId, ga-
meId.ToString());

    var logRecord = new LogRecord(player.Id, player.Username, pla-
yer.Color, EventType.Disconnected);
    _gameManager.AddLogRecord(gameId, logRecord);
    var logRecordDto = _mapper.Map<LogRecordDto>(logRecord);
    await Clients.Group(gameId.ToString()).SendAsync(SignalRMet-
hods.NewLogMessage, logRecordDto);

    var players = GetPlayerList(gameId);
    await Clients.OthersInGroup(gameId.ToString()).SendAsync(SignalR-
Methods.UpdatePlayers, players);

    await Clients.All.SendAsync(SignalRMethods.UpdateLobby);
}
```

6.4.2.4 Synchronizace kresby

Při připojení do probíhající hry je nutné uživateli zaslat dosavadní kresbu, jinak by viděl pouze obsah nakreslený až po jeho příchodu. K tomu byly vytvořeny dvě následující metody.

```
public async Task GetDrawing(int gameId)
{
    var player = _playerManager.GetPlayer(ConnectionId);
    var game = _gameManager.FindGameById(gameId);
    if (!game.IsInProgress) return;

    var drawer = game.CurrentDrawer;
    await Clients.Client(drawer.ConnectionId).SendAsync(SignalRMet-
hods.SendDrawing, player.Id);
}

public async Task SendDrawing(string drawing, int gameId, Guid play-
erId)
{
    var game = _gameManager.FindGameById(gameId);
```

```
    var player = game.Players.First(p => p.Id == playerId);
    await Clients.Client(player.ConnectionId).SendAsync(SignalRMet-
hods.GetDrawing, drawing);
}
```

První metodu GetDrawing volá nově připojený hráč pouze s ID hry. Ta zavolá klientskou funkci u konkrétního hráče, jímž je hráč, který aktuálně kreslí. Nově připojený hráč si tak vyžádá aktuální kontext plátna od kreslícího hráče, u něhož je jistota, že má přesnou podobu kresby.

U kreslícího hráče je spuštěna funkce SendDrawing, která uloženou kresbu odesílá na stejnojmennou funkci na serveru. Kreslící hráč s výzvou pro zaslání kresby obdrží také ID hráče, který má kresbu dostat. Tím lze následně na serveru identifikovat cílového hráče a skrze ID jeho spojení mu kontext plátna zaslat.

Při kreslení se však už neposílá celá kresba, ale pouze jednotlivé akce. Při kreslení linky se nejprve zavolá metoda BeginPath a následné tahy myší vyvolávají metodu Draw. Tyto metody následně u klientů vyvolávají stejné akce nad plátnem, jako provedl kreslící hráč.

```
public async Task BeginPath(BeginPathDto req)
{
    await Clients.GroupExcept(req.RoomId.ToString(), Connectio-
nId).SendAsync(SignalRMethods.BeginPath, req);
}

public async Task Draw(DrawDto req)
{
    await Clients.GroupExcept(req.RoomId.ToString(), Connectio-
nId).SendAsync(SignalRMethods.Draw, req);
}
```

Přenášeným obsahem metody BeginPath je objekt BeginPathDto obsahující například barvu a šířku linky. Metoda Draw posílá objekt DrawDto.

```
public class BeginPathDto
{
    public int RoomId { get; set; }
    public int OffsetX { get; set; }
    public int OffsetY { get; set; }
    public string Color { get; set; } = "#000000";
    public int Width { get; set; } = 1;
}
```

6.4.2.5 Hádání slova

U metod pro kreslení se akce nevolají pro hráče, který metodu zavolal, protože již kresbu zobrazenou má. Při hádání slov psaním do chatu se však zpráva odešle na server a aby se zobrazila hráči, který ji napsal, musí ji obdržet ze serveru. Ten ji nejdřív zpracuje a ověří, zda hráč slovo uhodl. Poté mu je zaslán celý objekt Answer, respektive AnswerDto. Pokud je odpověď správná, je ostatním uživatelům kvůli bezpečnosti zaslán objekt bez uhodnutého slova. Pokud je nesprávná, je uživatelům zaslán kompletní objekt.

```
public async Task Answer(int gameId, string message)
{
    var player = _playerManager.GetPlayer(ConnectionId);

    var answer = _gameManager.GuessWord(gameId, player, message);
    if (answer == null) return;

    var answerDto = _mapper.Map<AnswerDto>(answer);
    await Clients.Caller.SendAsync(SignalRMethods.Answer, answerDto);

    if (answer.IsCorrect)
    {
        var players = GetPlayerList(gameId);
        await Clients.Groups(gameId.ToString()).SendAsync(SignalRMet-
hods.UpdatePlayers, players);
        answerDto.Message = string.Empty;
        await Clients.OthersInGroup(gameId.ToString()).SendAsync(Sig-
nalRMethods.Answer, answerDto);
    }
    else
    {
        await Clients.OthersInGroup(gameId.ToString()).SendAsync(Sig-
nalRMethods.Answer, answerDto);
    }
}
```

Ve třídě Game je metoda GuessWord implementována pro následovně:

```
public bool GuessWord(Player player, string answer)
{
    var word = CurrentWord.Value.ToLower();
    answer = answer.ToLower();
```

```
        if (!word.Equals(answer)) return false;

        player.AddPoints(ScoreGain);
        CurrentDrawer.AddPoints(1);

        PlayersGuessed.Add(player.Id);
        ScoreGain += ScoreGain > 1 ? -1 : 0;

        var allGuessed = true;
        foreach (var player1 in Players)
        {
            if (!PlayersGuessed.Contains(player1.Id))
            {
                if (player1 != CurrentDrawer)
                {
                    allGuessed = false;
                    break;
                }
            }
        }
        if (allGuessed) EndRound();

        return true;
    }
```

Každým hráčem, který v aktuálním kole uhodne slovo, se snižuje bodový zisk pro následující hráče. Pokud všichni hráči ve hře uhodli, je kolo ukončeno.

Ve třídě GameManager, která metodu třídy Game volá, jsou k funkci implementovány ještě dodatečné akce a ověření. Nemůže například hádat kreslící hráč, nebo hráč, který již uhodl.

```
    public Answer GuessWord(int gameId, Player player, string message)
    {
        var game = FindGameById(gameId);
        if (game == null || !game.IsInProgress) return null;
        if (player.Id == game.CurrentDrawer.Id) return null;
        if (game.PlayersGuessed.Contains(player.Id)) return null;

        var isCorrect = game.GuessWord(player, message);
        Answer answer = new(player.Id, player.Username, player.Color,
    message, isCorrect);
```

```
        game.Answers.Add(answer);

        return answer;
    }
```

7 FRONTEND

7.1 Vytvoření projektu

Angular projekt byl vytvořen v terminálu příkazem *ng new*. Při zadání příkazu jsou na výběr různé volby nastavení projektu. Bylo zvoleno využití Angular routing a styly CSS.

Ke spuštění vytvořeného projektu slouží příkaz *ng serve*.

7.1.1 Instalace balíčků

Dálším krokem byla instalace balíčků. Byly nainstalovány balíčky NGXS, PrimeNG, SignalR a NGX Cookie Service pomocí správce balíčků NPM.

```
npm install @ngxs/store --save
npm install @microsoft/signalr
npm i ngx-cookie-service
```

Knihovna PrimeNG se rovněž instaluje skrze NPM.

```
npm install primeng
```

Poté je však nutné importovat nezbytné soubory se styly do hlavního CSS souboru.

```
@import "primeng/resources/themes/lara-light-blue/theme.css";
@import "primeng/resources/primeng.css";
```

Do soboru index.html byla přidána knihovna ikon FontAwesome.

```
<script src="https://kit.fontawesome.com/827bd95576.js" crossori-
gin="anonymous"></script>
```

7.1.2 Propojení s API

Adresa serveru byla přidána jako proměnná prostředí v environment.ts:

```
export const environment = {
  production: false,
  baseURL: "https://localhost:7151/"
};
```

7.2 Stav aplikace

K udržení stavu aplikace napříč komponentami je využíván NGXS. Stav udržuje úložiště (store). Ty byly v aplikaci vytvořeny celkem tři.

7.2.1 AppState

V úložišti AppState jsou uchovány hodnoty, které jsou využívány v rámci všech komponent. Jedná se o objekt aktuálního hráče a hodnoty, zda je hráč přihlášen a zda je ve hře.

```
export class AppStateModel {
    isLoggedIn: boolean
    isInRoom: boolean
    user: Player
}

@State<AppStateModel>({
    name: 'app',
    defaults: {
        isLoggedIn: false,
        isInRoom: false,
        user: {
            id: "",
            username: "",
            color: "",
            score: 0
        }
    }
})
```

Pro získání hodnot slouží selektory. Ty byly vytvořeny pomocí dekorátoru @Selector.

```
@Selector()
static getUser(state: AppStateModel) {
    return state.user
}
```

Pro změnu stavu se využívají akce. Ty jsou definovány v samostatném souboru app.actions.ts.

```
export class SetUser {
    static readonly type = "[App] Set user"

    constructor(public user: Player) {}
}
```

V konstruktorech jsou uvedeny parametry, které akce přijímají a které slouží pro následnou úpravu stavu. O změnu stavu se starají handlery – obslužné funkce. Ty se definují dekorátorem @Action. Pomocí funkce patchState je stav zvolených vlastností aktualizován novou hodnotou.

```
@Action(SetUser)
setUser({ patchState }: StateContext<AppStateModel>, { user }: SetU-
ser) {
    patchState({
        user: user
    })
}
```

7.2.2 RoomState

RoomState obsahuje stav herní místnosti. Jedná se o hodnoty získané ze serveru, čemuž odpovídají i objekty, které jsou shodné s DTO modely na straně serveru.

```
export class RoomStateModel {
    roomId: number
    roomName: string
    players: Player[]
    showPlayerList: boolean
    showRoomLog: boolean
    drawerId: string
    status: Status
    logRecords: LogRecord[]
    answers: Answer[]
    word: string
}

@State<RoomStateModel>({
    name: 'room'
})
```

Při změně stavu pole je nutné jej nahradit novým, pouhá modifikace existujícího pole totiž nevyvolá změnu stavu. Při přidání prvku do pole je proto nutné vytvořit pole nové, naplnit jej hodnotami z aktuálního stavu pole, který se získá metodou getState, a do stavu uložit pole nové.

```
@Action(AppendAnswer)
```

```
appendAnswer({ getState, patchState }: StateContext<RoomStateModel>,
{ answer }: AppendAnswer) {
    const state = getState()
    const currentAnswers = state.answers || []
    const updatedAnswers = [...currentAnswers, answer]

    patchState({
        answers: updatedAnswers,
    })
}
```

7.2.3 CanvasState

V tomto úložišti jsou uloženy informace o vybraném nástroji pro kreslení a barvě a šířce kreslené linky.

```
export class CanvasStateModel {
    activeTool: ToolType
    color: string
    width: number
}

@State<CanvasStateModel>({
    name: 'canvas',
    defaults: {
        activeTool: ToolType.Pencil,
        color: "#123456",
        width: 5
    }
})
```

7.3 Vzhled

7.3.1 Rozložení

Hlavní komponentou aplikace je kořenová komponenta root, kterou nalezneme v souborech app.component. Její HTML kód obsahuje pouze header komponentu a router-outlet, což je zástupný prvek, do kterého se bude vykreslovat obsah různých komponent na základě aktuálního směrování.

```
<app-header></app-header>
<router-outlet></router-outlet>
```

Komponenta header slouží jako záhlaví stránky a obsahuje horní panel, který je zobrazen v celé aplikaci.

Ve třídě komponenty jsou selektory pro výběr stavů z úložiště. Jedná se o typy Observable, již umožňují přihlásit se k odběru změn konkrétního stavu a přijímat aktualizace, kdykoliv se stav změní.

```
@Select(AppState.getIsLoggedIn) isLoggedIn$: Observable<boolean>
@Select(AppState.getUserName) userName$: Observable<string>
@Select(AppState.getUserColor) userColor$: Observable<string>
@Select(AppState.getIsInRoom) isInRoom$: Observable<boolean>
@Select(RoomState.getShowPlayerList) showPlayerList$: Obser-
vable<boolean>
@Select(RoomState.getShowRoomLog) showRoomLog$: Observable<boolean>
```

Toho je využito v metodě ngOnInit, která se volá při vytvoření komponenty.

```
ngOnInit(): void {
  this.userNameSubscription = this.userName$.subscribe(name => {
    this.userName = name
  })
  this.userColorSubscription = this.userColor$?.subscribe(color => {
    this.userColor = color
  })
}
```

Při každé změně stavu některé z hodnot se nová hodnota uloží do lokální proměnné, se kterou pak lze pracovat.

V šabloně neboli HTML části komponenty lze s Observable hodnotami pracovat i napřímo, a to pomocí pipe metody async. Tato funkcionalita umožňuje přihlásit se k asynchronnímu zdroji dat přímo v šabloně bez nutnosti starat se o přihlášení a odhlášení odběru.

```
<div class="item buttons">
    <i *ngIf="(isInRoom$ | async) === true"
        (click)="leaveRoom()"
        class="fa-solid fa-right-from-bracket fa-rotate-180 leave">
    </i>
    <i *ngIf="(isInRoom$ | async) === true"
        (click)="showPlayerList()"
        [ngClass]="(showPlayerList$ | async) === true ? 'fa-solid' :
'fa-regular'"
```

```
            class="fa-user">
        </i>
        <i *ngIf="(isInRoom$ | async) === true"
            (click)="showRoomLog()"
            [ngClass]="(showRoomLog$ | async) === true ? 'fa-solid' :
'fa-regular'"
            class="fa-rectangle-list">
        </i>
    </div>
```

Na panelu se nachází tlačítka, která jsou viditelná jen pokud je hráč připojený ve hře, logo a hráčův avatar, pokud je přihlášený. Oproti původnímu návrhu bylo přidáno tlačítko pro opuštění hry.

7.3.2 Design

Původním plánem bylo postavení většiny aplikace na komponentách z knihovny PrimeNG. Přestože jsou tyto komponenty navrženy tak, aby byly snadno konfigurovatelné, v aplikaci byly nakonec využity pouze minimálně, a to tam, kde jejich naprogramované funkce převažovaly nevýhodu náročnější změny vzhledu oproti tvorbě nových komponent. Navržený design aplikace se totiž od vzhledu prvků z knihovny výrazně liší a vlastní vytvoření a nastylování prvku je tak často jednodušší.

Globální styly jsou definovány v souboru style.css. Většina designových úprav je však prováděna v CSS souborech jednotlivých komponent. Při využívání PrimeNG komponent je nutné pro změnu jejich vzhledu použít selektor ::ng-deep, který umožňuje použít styly na podřízené komponenty nebo prvky v rámci šablony komponenty.

```
::ng-deep .p-inputswitch.p-inputswitch-checked .p-inputswitch-slider
{
    background-color: var(--green);
}
```

7.4 Služba API a SignalR

7.4.1 API

API metody jsou definovány ve třídě ApiService. Třídy tohoto typu se vytvářejí příkazem *ng generate service*.

```
export class ApiService {
  apiUrl = environment.baseURL + "api/"
```

```
  constructor(private http: HttpClient) { }

  getRooms(): Observable<GameLobby[]> {
    return this.http.get<GameLobby[]>(this.apiUrl + "games")
  }

  createGame(request: CreateGameRequest): Observable<number> {
    return this.http.post<number>(this.apiUrl + "games/create",
request)
  }

  getGameInfo(id: number): Observable<GameInfo> {
    return this.http.get<GameInfo>(this.apiUrl + "game/" + id)
  }
}
```

7.4.2 SignalR

Pro SignalR funkce byla vytvořena třída SignalRService. Skrze dependency injection využívá úložiště NGXS Store a směrovač Router.

```
export class SignalRService {
  private _hubConnection!: signalR.HubConnection
  private _connectionStatus = new BehaviorSubject<boolean>(false)
  public connectionStatus$: Observable<boolean> = this._connection-
Status.asObservable()

  constructor(private store: Store, private router: Router) { }
```

7.4.2.1 Vytvoření spojení

K navázání SignalR spojení se serverem byla vytvořena metoda startConnection.

```
public startConnection = () => {
  this._hubConnection = new signalR.HubConnectionBuilder()
    .withUrl(environment.baseURL + 'hub')
    .build()

  this._hubConnection.start()
    .then(() => this.setConnected(true))
    .catch(err => console.log('Error while starting connection: ' +
err))
}
```

```
private setConnected(status: boolean): void {
  this._connectionStatus.next(status)
}
```

Tato metoda je volána v kořenové komponentě root při spuštění aplikace.

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
  title = 'Draw and Guess';

  constructor(
    private signalR: SignalRService,
    private store: Store
  ) { }

  ngOnInit(): void {
    this.signalR.startConnection()
    this.store.dispatch(new SwitchShowPlayerList())
    this.store.dispatch(new SwitchShowRoomLog())
  }
}
```

7.4.2.2 Přihlášení

Funkce login je asynchronní funkce, která se pokouší provést operaci přihlášení na serveru pomocí připojení SignalR. Jako parametry přijímá jméno a barvu uživatele, se kterými vyvolá metodu SignalR hubu na serveru a čeká na odpověď. Pokud je přihlášení úspěšné, vrátí objekt hráče. Pokud se během přihlašování vyskytnou nějaké chyby, jsou zachyceny a zaznamenány do konzole a je vrácena hodnota undefined. V případě, že by se nejednalo o metodu hubu, u které očekáváme odpověď, bylo by možné využít metodu send namísto invoke.

```
public async login(name: string, color: string): Promise<Player | un-
defined> {
  try {
    const player = await this._hubConnection.invoke("login", name,
color)
    return player
```

```
    } catch (error) {
      console.log(error)
      return undefined
    }
  }
```

7.4.2.3 Aktualizace dostupných her

Pro přijmutí nového seznamu dostupných her byly vytvořeny následující funkce.

```
public addUpdateLobbyListener(callback: (data: any) => void): void {
  this._hubConnection.on('UpdateLobby', callback)
}
public removeUpdateLobbyListener(callback: (data: any) => void): void
{
  this._hubConnection.off('UpdateLobby', callback)
}
```

Funkce addUpdateLobbyListener nastaví posluchače události UpdateLobby v připojení k hubu SignalR. Jako parametr přebírá funkci zpětného volání, která bude vyvolána při přijetí události. Funkce removeUpdateLobbyListener odstraní dříve přidaný posluchač. Jako parametr přebírá rovněž funkci zpětného volání, která by měla odpovídat funkci zpětného volání použité při přidání posluchače. Po odebrání již nebude callback funkce vyvolána, když dojde k události.

7.4.2.4 Připojení do hry

K připojení do herní místnosti slouží funkce joinRoom. Pokud metoda vyvolaná na serveru vrátí hodnotu true značící úspěšné spojení, provede se odeslání dvou akcí pomocí NGXS metody store.dispatch. První akce nastaví aktuální roomId do stavu odesláním akce SetRoomId a druhá akce nastaví příznak isInRoom na true odesláním akce SetIsInRoom. V opačném případě je uživatel přesměrován do lobby na výběr her.

```
public async joinRoom(roomId: number): Promise<boolean>  {
  const joined = await this._hubConnection.invoke("joinRoom", roomId)
  if (!joined) {
    this.router.navigate(["/lobby"])
    return false
  }

  this.store.dispatch(new SetRoomId(roomId))
```

```
    this.store.dispatch(new SetIsInRoom(true))

    return true
  }
```

7.4.2.5 Zprávy chatu

Pro odeslání hádaného slova slouží funkce sendAnswer. Pro příjem zpráv byla vytvořena funkce k přidání posluchače události a funkce, která tento posluchač odstraní.

```
public sendAnswer = (roomId: number, answer: string) => {
  this._hubConnection.send("Answer", roomId, answer)
}
public addAnswerListener(callback: (answer: Answer) => void): void {
  this._hubConnection.on("Answer", callback)
}
public removeAnswerListener(callback: (answer: Answer) => void): void
{
  this._hubConnection.off("Answer", callback)
}
```

Stejným způsobem byly implementovány i ostatní SignalR metody.

7.5 Komponenty

Aplikace je rozdělena do třech hlavních částí, které zastupují funkci stránek. Jedná se o komponenty se specifikovanou cestu, přes kterou k nim lze skrze URL adresu přistoupit. Cesty pro směrování jsou definovány v souboru app-routing.module.ts.

```
const routes: Routes = [
  { path: '', redirectTo: 'lobby', pathMatch: 'full' },
  { path: 'lobby', component: LobbyComponent },
  { path: 'login', component: LoginComponent },
  { path: 'room', redirectTo: 'lobby', pathMatch: 'full' },
  { path: 'room/:id', component: RoomComponent },
];
```

Výchozí stránkou aplikace je lobby, jež slouží pro zobrazení aktivních her a připojení k nim.

7.5.1 Přihlašovací stránka

7.5.1.1 Šablona

Obrázek 9. Přihlašovací stránka

Přihlašovací stránka obsahuje jednoduchý formulář s uživatelským vstupem pro jméno a výběr barvy, která slouží pro odlišení hráčů. Nad tlačítkem pro přihlášení se nachází přepínač pro výběr, zda si má aplikace údaje o uživateli zapamatovat.

```
<div class="card">
    <h2>Přihlášení</h2>
    <form (ngSubmit)="login()">
        <div class="name">
            <input type="text" id="name" name="name" [(ngMo-
del)]="name" placeholder="Jméno">
        </div>
        <div class="color">
            <span>Barva avataru</span>
            <p-colorPicker name="avatarColor" [(ngModel)]="avatar-
Color"></p-colorPicker>
        </div>
        <div class="save">
            <span>Zapamatovat</span>
            <p-inputSwitch name="save" [(ngModel)]="remember"></p-
inputSwitch>
        </div>
        <div class="submit">
            <button type="submit" [disabled]="name === ''">Přihlásit
se</button>
```

```
        </div>
    </form>
</div>
```

7.5.1.2 Třída

Tlačítku je přiřazena funkce po kliknutí, která volá metodu login třídy LoginService.

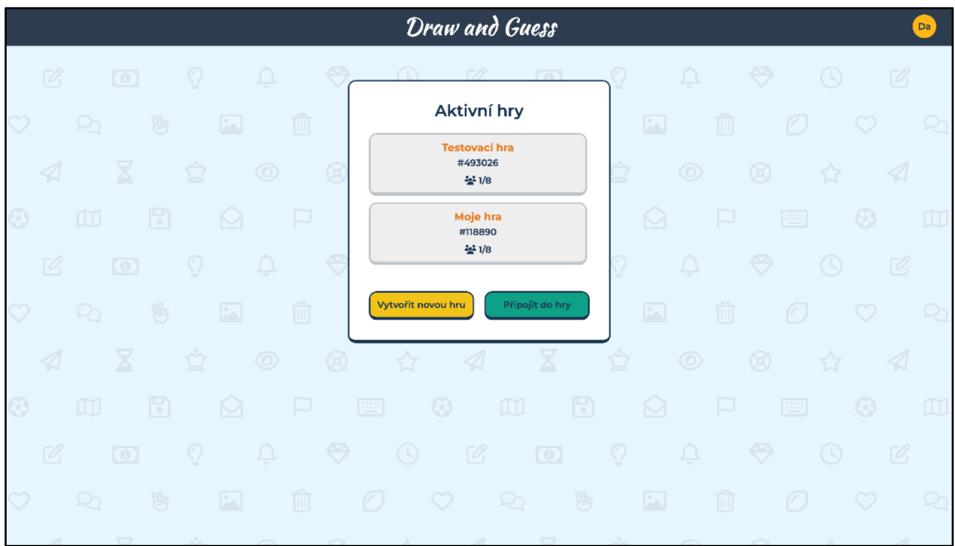
```
async login() {
  if (this.name === "") return
  const logged = await this.loginService.login(this.name, this.ava-
tarColor, this.remember)
  if (logged) this.router.navigate(["/lobby"])
}
```

Tato metoda přihlásí uživatele skrze SignalR a údaje uloží do paměti relace v prohlížeči. Pokud byla zvolena možnost pro zapamatování údajů, jsou rovněž uloženy jako soubory cookies, které mají dobu expirace 10 dní.

```
async login(name: string, color: string, remember: boolean): Pro-
mise<boolean> {
  const player = await this.signalR.login(name, color)
  if (player === undefined) return false

  sessionStorage.setItem("dag.name", name)
  sessionStorage.setItem("dag.color", color)

  this.store.dispatch(new SetUser(player))
  this.store.dispatch(new SetIsLoggedIn(true))

  this.cookieService.delete("dag.name")
  this.cookieService.delete("dag.color")

  if (remember) {
    const expirationDate = new Date(new Date().getTime() + 10 * 24 *
60 * 60 * 1000) // 10 days
    this.cookieService.set("dag.name", name, expirationDate)
    this.cookieService.set("dag.color", color, expirationDate)
  }

  return true
}
```

7.5.2 Lobby

7.5.2.1 Šablona



Obrázek 10. Lobby hry

Lobby slouží jako stránka pro výběr dostupných her. Kliknutím na hru je vyvoláno připojení do hry.

```
<div *ngIf="isLoggedIn$ | async" class="card">
    <h2>Aktivní hry</h2>
    <div class="rooms">
        <div *ngIf="(rooms$ | async)?.length === 0">
            <span class="none">Žádná hra nenalezena</span>
        </div>
        <div *ngFor="let room of rooms$ | async" class="room"
(click)="room.playerCount < 8 && join((room.id).toString())"
[class.full]="room.playerCount >= 8">
            <h3>{{room.name}}</h3>
            <div class="number">#{{room.id}}</div>
            <div class="players"><i class="fa-solid fa-users"></i>
{{room.playerCount}}/8</div>
        </div>
    </div>
    <div class="container">
        <button (click)="showCreateDialog()" class="item">Vytvořit
novou hru</button>
        <button (click)="showJoinDialog()" class="item join">Připojit
do hry</button>
    </div>
```

```
</div>
```

Pod dostupnými hrami se nachází dvě tlačítka pro vytvoření hry nebo připojení do hry pomocí jejího ID. To lze využít v případě, kdy je hra v seznamu skryta, ale uživatel její ID zná.

Tato tlačítka otevírají dialogová okna. Dialogové okno pro vytvoření hry obsahuje vstupní pole pro zadání názvu hry a přepínač určující, zdali má být hra skryta v seznamu. Dialogové okno pro připojení do hry obsahuje vstupní pole pro ID.

7.5.2.2 Třída

Při inicializaci lobby je zavolána funkce loginOrRedirect. Pokud uživatel ještě není přihlášen, funkce se pokusí získat zapamatované údaje ze souborů cookies. Pokud údaje uloženy nejsou, je uživatel přesměrován na stránku přihlášení. Seznam dostupných her je zíakán pomocí API metody getRooms.

```
ngOnInit(): void {
  this.connectionSubscription = this.signalR.connectionStatus$.sub-
scribe(isConnected => {
    if (isConnected) {
      this.login.loginOrRedirect().then(_ => {
        this.rooms$ = this.apiService.getRooms()
      })
    }
  })

  this.signalR.addUpdateLobbyListener(this.updateLobbyListenerCall-
back)
}
```

Funkce loginOrRedirect:

```
async loginOrRedirect(): Promise<boolean> {
  const isLoggedIn = this.store.selectSnapshot(AppState.getIsLogge-
dIn)
  if (isLoggedIn) return true

  let name: any
  let color: any
  let remember = true

  name = this.cookieService.get("dag.name")
```

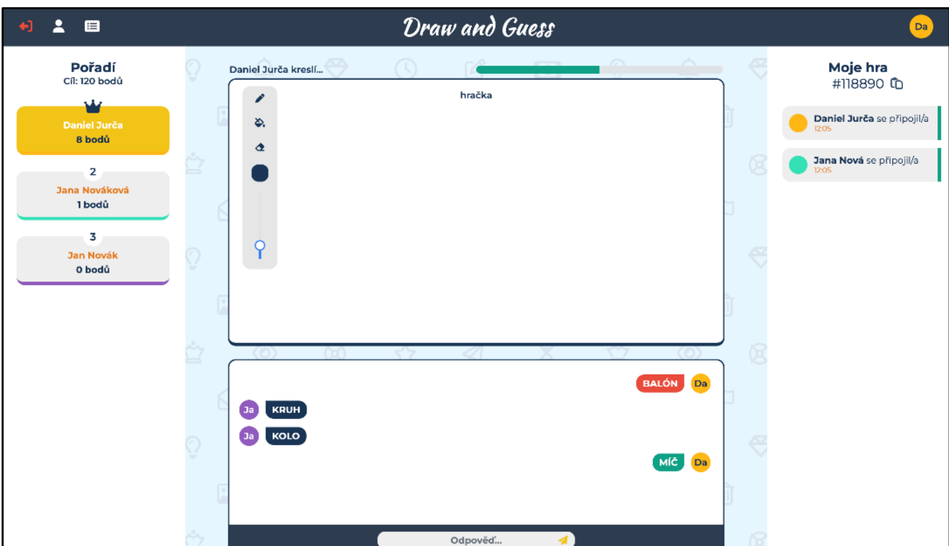
```
    color = this.cookieService.get("dag.color")

    if (name === "" || color === "") {
      name = sessionStorage.getItem("dag.name")
      color = sessionStorage.getItem("dag.color")
      remember = false
    }

    if (name === null || color === null) {
      this.router.navigate(["/login"])
      return false
    }

    const logged = await this.login(name, color, remember)
    if (logged === false) {
      this.router.navigate(["/login"])
      return false
    }

    return true
  }
```

Callback funkce updateLobbyListenerCallback je určena k vytvoření nového API požadavku pro získání dostupných her při obdržení SignalR notifikace na aktualizaci lobby. Tato notifikace je zasílána všem uživatelům aplikace, proto je efektivnější, aby si aktualizovaný seznam vyžádali sami uživatelé nacházející se v lobby, než aby byl přenášen v notifikaci.

```
  updateLobbyListenerCallback = () => {
    this.rooms$ = this.apiService.getRooms();
  }
```

7.5.3 Herní místnost

7.5.3.1 Šablona



Obrázek 11. Herní místnost

Herní místnost je stránka, kde se odehrává samotná hra. Jednotlivé části jsou rozděleny do vlastních komponent. Nachází se zde komponenta pro zobrazování toast notifikací.

```
<p-toast></p-toast>
<div *ngIf="roomLoaded">
    <div class="players">
        <app-player-list [hidden]="!(showPlayerList$ | async)"></app-
player-list>
    </div>
    <div class="container">
        <div class="item">
            <app-status-bar></app-status-bar>
            <app-canvas></app-canvas>
            <app-answer-chat></app-answer-chat>
        </div>
    </div>
    <div class="log">
        <app-room-log [hidden]="!(showRoomLog$ | async)"></app-room-
log>
    </div>
</div>
```

7.5.3.2 Třída

Ve třídě této komponenty byla implementována funkcionalita připojení do místnosti. Z URL adresy je získáno ID hry. Pokud je nevalidní, je uživatel přesměrován do lobby. Díky tomu se lze do her připojovat i pomocí odkazu.

Pokud je navázáno SignalR spojení a uživatel je přihlášený, je zavolána SignalR metoda pro připojení do hry. Po úspěšném připojení jsou skrze API získána data dané hry a uložena do stavu aplikace, odkud si je přebírají další komponenty.

```
ngOnInit(): void {
  this.routeParamsSubscription = this.route.params.subscribe(params
=> {
    const id = +params['id']
    const isNumber = !isNaN(Number(id))
    if (!isNumber) this.router.navigate(["/lobby"])
    this.roomId = id
  })

  this.roomStateIdSubscription = this.roomStateId$.subscribe(id => {
    this.roomStateId = id
  })

  this.connectionSubscription = this.signalR.connectionStatus$.pipe(
    filter(isConnected => isConnected),
    switchMap(() => this.login.loginOrRedirect()),
    filter(logged => logged),
    switchMap(() => this.signalR.joinRoom(this.roomId)),
    switchMap(() => this.isInRoom$.pipe(
      filter(isInRoom => isInRoom && this.roomId === this.roomSta-
teId),
      take(1),
      switchMap(() => this.api.getGameInfo(this.roomId))
    ))
  ).subscribe(gameInfo => {
    if (gameInfo === null) return
    this.store.dispatch(new SetRoomInfo(gameInfo))
    this.roomLoaded = true
  })

  this.signalR.addUpdatePlayersListener(this.updatePlayersCallback)
```

```
    this.signalR.addDrawerListener(this.drawerListenerCallback)
    this.signalR.addStatusListener(this.statusListenerCallback)
    this.signalR.addNewLogRecordListener(this.logRecordCallback)
    this.signalR.addAnswerListener(this.answerCallback)
    this.signalR.addWordListener(this.wordCallback)
  }
```

Jsou zde rovněž vytvořeny callback funkce pro posluchače SignalR metod, které aktualizují stav aplikace. Obdrží-li klient například nový seznam hráčů nebo status hry, uloží se do stavu aplikace a změny se zobrazí v komponentách, které tyto hodnoty ze stavu využívají.

```
  updatePlayersCallback = (players: Player[]) => {
    this.store.dispatch(new SetPlayerList(players))
  }
  drawerListenerCallback = (drawerId: string) => {
    this.store.dispatch(new SetDrawerId(drawerId))
  }
  statusListenerCallback = (status: Status) => {
    this.store.dispatch(new SetStatus(status))
  }
  logRecordCallback = (record: LogRecord) => {
    this.store.dispatch(new AppendLogRecord(record))
  }
  answerCallback = (answer: Answer) => {
    this.store.dispatch(new AppendAnswer(answer))
  }
  wordCallback = (word: string) => {
    this.store.dispatch(new SetWord(word))
  }
```

Při odpojení ze hry je nutné tyto posluchače ukončit. Z toho důvodu komponenta implementuje rozhraní OnDestroy nabízející metodu ngOnDestroy, která se zavolá při zničení objektu komponenty.

```
  ngOnDestroy(): void {
    this.signalR.leaveRoom(this.roomId)
    this.store.dispatch(new ResetRoomState)

    this.signalR.removeUpdatePlayersListener(this.updatePlayersCall-
  back)
    this.signalR.removeDrawerListener(this.drawerListenerCallback)
```

```
  this.signalR.removeStatusListener(this.statusListenerCallback)
  this.signalR.removeNewLogRecordListener(this.logRecordCallback)
  this.signalR.removeAnswerListener(this.answerCallback)
  this.signalR.removeWordListener(this.wordCallback)

  this.routeParamsSubscription.unsubscribe
  this.connectionSubscription.unsubscribe
  this.roomStateIdSubscription.unsubscribe
}
```

7.5.3.3 Seznam hráčů

Pro zobrazení hráčů připojených ve hře a jejich skóre slouží komponenta PlayerList. K hráčům přistupuje ze stavu aplikace. K rozlišení hráče aktuálního uživatele se porovnávají ID objektů.

7.5.3.4 Záznamy aktivit

Pravý panel místnosti obsahuje komponentu RoomLog. Ta zobrazuje záznamy o připojení a odpojení hráčů. Také se zde nachází jméno a ID místnosti, které lze tlačítkem zkopírovat do schránky. Informace o zkopírování je uživateli zobrazena formou toast notifikace.

```
async copyIdToClipboard(): Promise<void> {
  try {
    const roomId = this.store.selectSnapshot(RoomState.getRoomId)
    await navigator.clipboard.writeText(roomId.toString())
    this.messageService.add({ severity: 'success', summary: 'Zkopíro-
váno', detail: 'ID bylo zkopírováno do schránky' })
  } catch (error) {
    this.messageService.add({ severity: 'error', summary: 'Chyba',
detail: 'Nepodařilo se zkopírovat ID' })
  }
}
```

7.5.3.5 Kreslicí plátno

```
<canvas #canvasElement width="750" height="400"></canvas>
```

Jako kreslicí plátno byl využit HTML element canvas, který zvládá vykreslování grafiky, animací a dalšího vizuálního obsahu. Pro kreslení do něj je potřeba ve třídě komponenty získat jeho kontext.

```
@ViewChild('canvasElement') canvasElement: ElementRef | undefined
ngAfterViewInit() {
```

```
    if (this.canvasElement === undefined) return
    this.canvas = this.canvasElement.nativeElement
    this.ctx = this.canvas.getContext('2d')
  }
```

Na eventy elementu byly přidány posluchače. Například pro kreslení linky myší je nejprve zmáčknuto tlačítko myši, které vyvolá event mousedown. Callback funkce k tomuto eventu přidá posluchač na event mousemove a zavolá funkci beginPath, jež je pomocí SignalR provedena i u ostatních hráčů. Synchronizace kresby totiž funguje na provádění stejných akcí, které kreslící hráč provede, na canvas všech hráčů.

```
this.canvas.addEventListener('mousedown', this.mousedownListener)

this.mousedownListener = (event: any) => {
  this.beginPath(event.offsetX, event.offsetY, this.color,
this.width)

  this.canvas.addEventListener('mousemove', this.mousemoveListener)
  this.signalR.invokeBeginPath({
    roomId: this.roomId,
    offsetX: event.offsetX,
    offsetY: event.offsetY,
    color: this.color,
    width: this.width
  })
}
beginPath(offsetX: number, offsetY: number, color: string, width: nu-
mber) {
  this.ctx.beginPath()
  this.ctx.moveTo(offsetX, offsetY)
  this.ctx.strokeStyle = color
  this.ctx.lineWidth = width
  this.ctx.lineCap = "round"
  this.ctx.lineJoin = 'round'
}
```

Zavolání funkce při obdržení notifikace SignalR:

```
this.signalR.addBeginPathListener(this.beginPathListenerCallback)

beginPathListenerCallback = (payload: BeginPathDto) => {
```

```
  this.beginPath(payload.offsetX, payload.offsetY, payload.color, pa-
yload.width)
}
```

Metoda beginPath volá stejnojmennou metodu canvas kontextu. Tím se začne kresba nové linky a nastaví se vlastnosti jako barva nebo šířka. Následné pohyby myší vyvolávají event mousemove a callback funkce na canvas kreslí. Opět je tato akce provedena i u ostatních hráčů.

```
this.canvas.addEventListener('mousemove', this.mousemoveListener)

this.mousemoveListener = (event: any) => {
  this.draw(event.offsetX, event.offsetY)
  this.signalR.invokeDraw({
    roomId: this.roomId,
    offsetX: event.offsetX,
    offsetY: event.offsetY
  })
}

draw(offsetX: number, offsetY: number) {
  this.ctx.lineTo(offsetX, offsetY)
  this.ctx.stroke()
}
```

Po puštění tlačítka myši je vyvolán event mouseup, který kresbu ukončí odebráním posluchače na event mousemove.

```
this.canvas.addEventListener('mouseup', this.mouseupListener)

this.mouseupListener = () => {
  this.canvas.removeEventListener('mousemove', this.mousemoveLis-
tener)
}
```

Pokud skončí kolo, jsou hráči odebrány všechny posluchače na události na plátně.

Synchronizace celé kresby pro nově připojené hráče byla implementována následovně:

```
sendDrawingListenerCallback = (playerId: string) => {
  this.ctx.save()
  const drawing = this.canvas.toDataURL('image/png')
  this.ctx.restore()
```

```
    this.signalR.sendDrawing(drawing, this.roomId, playerId)
  }

  getDrawingListenerCallback = (drawing: string) => {
    const img = new Image()

    img.onload = () => {
      this.ctx.drawImage(img, 0, 0)
    }

    img.src = drawing
  }
```

Hráč, který kreslí, obdrží SignalR notifikaci SendDrawing. Uloží aktuální stav plátna a zavolá funkci toDataURL, která vrací datovou adresu URL obsahující reprezentaci obrázku na plátně. Datová adresa URL představuje obsah plátna jako obrázek PNG kódovaný podle base64. Ta je následně zaslána nově připojenému hráči, který ji převede na objekt Image a vykreslí na canvas.

7.5.3.6 Status hry

Status obsahuje textový popis průběhu hry a progres bar, který zobrazuje zbývající čas kola nebo přestávky.

```
<p-progressBar [value]="remainingTime" [showValue]="false" [styleC-
lass]="getProgressBarClass()"></p-progressBar>
```

Při každém obdržení nového statusu ze serveru je vypočítán interval a vytvořen časovač, který každou sekundu snižuje čas, čímž se mění procentuální vyjádření zbývajícího času, které je zobrazeno v progres baru.

```
ngOnInit(): void {
  this.statusSubscription = this.status$.subscribe(status => {
    clearInterval(this.timer)

    if (status === undefined) return
    this.status = status

    if (status.status === 2) return
    this.remainingTime = 100 / (this.status.time / 1000) * (this.sta-
tus.timeRemaining / 1000)
```

```
      if (status.time === 0 || status.time === null) return
      this.timer = setInterval(this.decreaseTime.bind(this), 1000)
    })
  }

  decreaseTime() {
    if (this.status.time === undefined) clearInterval(this.timer)

    const decrement = 100 / (this.status.time / 1000)
    this.remainingTime -= decrement

    if (this.remainingTime <= 0) {
      clearInterval(this.timer)
      this.remainingTime = 0
    }
  }
```

7.5.3.7 Nástroje pro kreslení

Komponenta DrawingTools umisťuje do kreslicího plátna panel s nástroji pro kreslení a obsahuje funkce sloužící k výběru nástroje a nastavení barvy a šířky linky. Volby se ukládají do CanvasState, aby byly k dispozici při kreslení.

7.5.3.8 Chat

Chat pro zasílání odpovědí na hádané slovo byl vytvořen v komponentě AnswerChat. K posílání zpráv slouží funkce volající SignalR metodu.

```
  sendAnswer(answer: string) {
    if (answer !== "") {
      this.signalR.sendAnswer(this.roomId, answer)
      this.answer = ""
    }
  }
```

8 OPTIMALIZACE A VYSTAVENÍ

8.1 Optimalizace SignalR

Microsoft Azure v bezplatném tarifu limituje denní počet SignalR zpráv na 20 tisíc. Je proto podstatné pokusit se nejvíce vytěžující funkce optimalizovat. V této aplikaci vytváří nejvíce požadavků akce kreslení, kterou vyvolává event mousemove.

```
this.canvas.addEventListener('mousemove', this.mousemoveListener)

this.mousemoveListener = (event: any) => {
  this.draw(event.offsetX, event.offsetY)
  this.signalR.invokeDraw({
    roomId: this.roomId,
    offsetX: event.offsetX,
    offsetY: event.offsetY
  })
}
```

Jednoduchým výpisem času do konzole lze zjistit počet eventů, které funkce vyvolá.

```
console.log(Date())
```

Webový prohlížeč seskupuje shodné zprávy vypsané do konzole, díky čemuž lze lehce vyčíst počet výpisů, které byly v každou sekundu provedeny.

```
56 Mon May 22 2023 09:03:09 GMT+0200 (středoevropský letní čas)
60 Mon May 22 2023 09:03:10 GMT+0200 (středoevropský letní čas)
60 Mon May 22 2023 09:03:11 GMT+0200 (středoevropský letní čas)
60 Mon May 22 2023 09:03:12 GMT+0200 (středoevropský letní čas)
60 Mon May 22 2023 09:03:13 GMT+0200 (středoevropský letní čas)
57 Mon May 22 2023 09:03:14 GMT+0200 (středoevropský letní čas)
60 Mon May 22 2023 09:03:15 GMT+0200 (středoevropský letní čas)
44 Mon May 22 2023 09:03:16 GMT+0200 (středoevropský letní čas)
4 Mon May 22 2023 09:03:22 GMT+0200 (středoevropský letní čas)
60 Mon May 22 2023 09:03:23 GMT+0200 (středoevropský letní čas)
60 Mon May 22 2023 09:03:24 GMT+0200 (středoevropský letní čas)
55 Mon May 22 2023 09:03:25 GMT+0200 (středoevropský letní čas)
60 Mon May 22 2023 09:03:26 GMT+0200 (středoevropský letní čas)
60 Mon May 22 2023 09:03:27 GMT+0200 (středoevropský letní čas)
```

Obrázek 12. Frekvence zpráv eventu mousemove

Z obrázku je zřejmé, že event mousemove při kreslení na canvas je volán až 60x každou sekundu. To znamená, že jen pouhá synchronizace kreslení na server by denní limit v Azure spotřebovala za zhruba 5,5 minuty. Pokud by byly zprávy navíc posílány druhému hráči, čas by byl už jen poloviční.

Je tedy nutné omezit posílání SignalR zpráv při každém vyvolání eventu, ovšem neodeslané eventy uchovat, aby byla synchronizována celá kresba.

Vytvoření bufferu pro seskupení více eventů:

```
buffer: { offsetX: number, offsetY: number }[] = []
isBuffering = false
addToBuffer(offsetX: number, offsetY: number): void {
  this.buffer.push({ offsetX, offsetY })
}

flushBuffer(): void {
  const points = this.buffer.slice()

  this.buffer = []
  this.isBuffering = false

  this.signalR.invokeDraw({
    roomId: this.roomId,
    points
  })
}
```

Zakomponování bufferu do obslužné funkce eventu mousemove:

```
this.mousemoveListener = (event: any) => {
  this.draw(event.offsetX, event.offsetY)

  this.addToBuffer(event.offsetX, event.offsetY)
  if (!this.isBuffering) {
    this.isBuffering = true
    setTimeout(() => this.flushBuffer(), 200)
  }
}
```

Buffer se plní hodnotami z eventů a je vyprázdněn a odeslán na server každých 200 milisekund.

```
4 Mon May 22 2023 13:00:20 GMT+0200 (středoevropský letní čas)
4 Mon May 22 2023 13:00:21 GMT+0200 (středoevropský letní čas)
5 Mon May 22 2023 13:00:22 GMT+0200 (středoevropský letní čas)
4 Mon May 22 2023 13:00:23 GMT+0200 (středoevropský letní čas)
5 Mon May 22 2023 13:00:24 GMT+0200 (středoevropský letní čas)
5 Mon May 22 2023 13:00:25 GMT+0200 (středoevropský letní čas)
4 Mon May 22 2023 13:00:26 GMT+0200 (středoevropský letní čas)
4 Mon May 22 2023 13:00:27 GMT+0200 (středoevropský letní čas)
5 Mon May 22 2023 13:00:28 GMT+0200 (středoevropský letní čas)
4 Mon May 22 2023 13:00:29 GMT+0200 (středoevropský letní čas)
2 Mon May 22 2023 13:00:30 GMT+0200 (středoevropský letní čas)
  Mon May 22 2023 13:00:30 GMT+0200 (středoevropský letní čas)
```

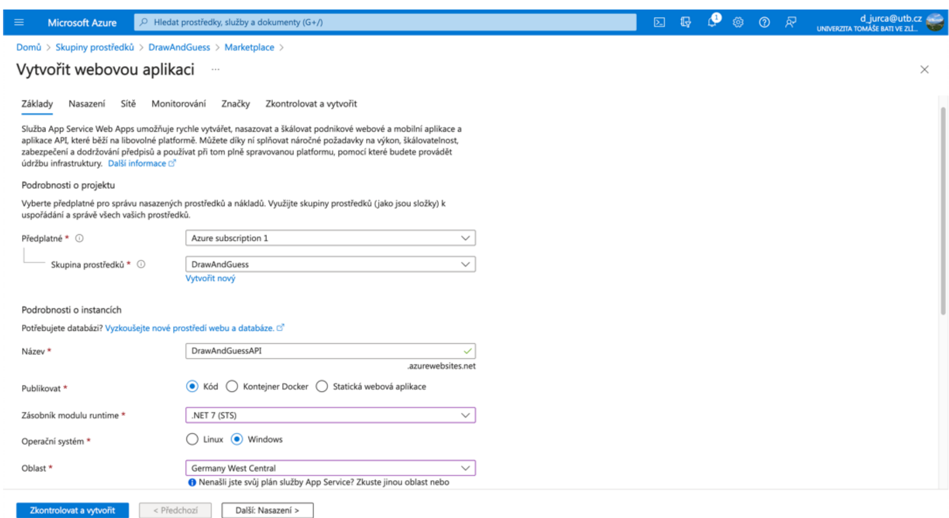
Obrázek 13. Frekvence zpráv optimalizované funkce

Z výpisu času při využití bufferu lze na první pohled vidět výrazné snížení frekvence výpisu a tím i odesílání zpráv na server. Razantně se tak prodlužuje doba, po kterou lze kreslit na plátno, než bude vyčerpán denní limit. Nevýhodou je snížení plynulosti kreslení u ostatních hráčů, kterým jsou data posílána. Je proto vždy potřeba najít vhodný kompromis mezi ušetřením prostředků a příjemným uživatelským zážitkem.

8.2 Vystavení do Microsoft Azure

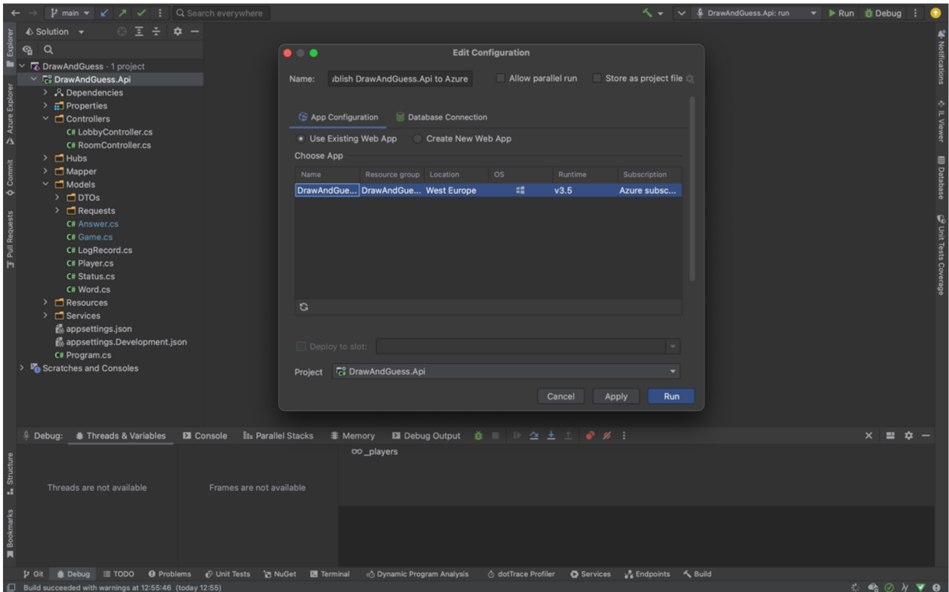
8.2.1 Backend

Pro vystavený projektu do cloudové služby Microsoft Azure je nejprve nutné vytvořit novou skupinu prostředků pro daný projekt. Ve vytvořené skupině byl vytvořen nový prostředek typu webová aplikace. V konfiguraci byl jako název zadán DrawAndGuessAPI a jako runtime zvolen .NET 7. Tarif byl zvolen bezplatný, který pro účely této práce dostačuje.



Obrázek 14. Vytvoření prostředku v Microsoft Azure

Pro vystavení kódu ASP.NET aplikace do tohoto prostředku byl využit vestavěný nástroj pro publikaci do Azure přímo v IDE Rider. Po přihlášení Azure účtu jsou k dispozici dostupné prostředky, do kterých lze aplikaci zveřejnit. Stačí tedy zvolit prostředek a spustit publikování.



Obrázek 15. Publikování aplikace v JetBrains Rider

Aplikace je po nahrání do cloudu k dispozici na adrese https://drawandguessapi.azurewebsites.net/.

8.2.2 Frontend

Pro aplikaci v Angularu je opět potřeba vytvořit nový prostředek. Tentokrát s runtime Node 18 LTS. Aplikace byla nazvána DrawAndGuess.

Před publikací je nutné v klientské aplikaci nastavit proměnnou s novou adresou API v souboru environment.prod.ts, který bude použitý při sestavení produkční verze aplikace.

```
export const environment = {
  production: true,
  baseURL: "https://drawandguessapi.azurewebsites.net/"
};
```

Pro sestavení produkční verze, která je určena pro publikování, slouží následující terminálový příkaz:

```
ng build –configuration=production
```

Sestavenou aplikaci nalezneme v adresáři dist. Pomocí rozšíření Azure App Service ve Visual Studio Code byla vystavena do cloudu.

Aby se aplikace v cloudu spustila, bylo nutné v konfiguraci prostředku přidat následující příkaz do spouštěcích příkazů:

```
pm2 serve /home/site/wwwroot --no-daemon –spa
```

Posledním krokem bylo povolení této aplikace pro komunikaci s API. To se provede v záložce CORS v nastavení DrawAndGuessAPI prostředku přidáním adresy klientské aplikace a povolením Access-Control-Allow-Credentials.

Aplikace je k dispozici na adrese **https://drawandguess.azurewebsites.net/**.

8.3 Zabezpečení aplikace

Při publikaci projektu je vhodné se zamyslet nad zabezpečením aplikace. Výhodou této aplikace je fakt, že nezpracovává a neuchovává žádná citlivá uživatelská data. Proto se není nutné zabývat například zabezpečením dat v databázi. Přesto je zde pár oblastí, které si pozornost zaslouží.

8.3.1 ConnectionId

V SignalR je ConnectionId jedinečný identifikátor přiřazený každému klientskému připojení. Identifikátor je přiřazen při navázání spojení klienta se serverem SignalR a zůstává konstantní po celou dobu trvání tohoto spojení. Vystavení ConnectionId klientům může přinést bezpečnostní riziko – pokud by uživatel získal přístup k identifikátoru jiného klienta, může se za něj vydávat a provádět jeho jménem neoprávněné akce.

Proto se v aplikaci využívá model PlayerDto, pomocí kterého jsou vytvářeny objekty určené k přenosu ke klientům, které ale neobsahují ConnectionId, jež se v doménové třídě Player nachází. K identifikaci jednotlivých hráčů na straně klienta má každý hráč vytvořeno unikátní GUID, u něhož není takové riziko zneužití jako u ConnectionId.

8.3.2 Hádané slovo

Co se týče zabezpečení aplikace proti podvádění, kromě validací na straně klienta jsou implementována různá ověření i na straně serveru, která zabraňují mimo jiné zisku bodů v neoprávněných případech.

Rovněž je chování serveru naprogramováno tak, aby hádané slovo při hře obdržel pouze hráč, který kreslí. Tím je zamezeno získání hádaného slova hráčem, který je v roli hádajícího. To platí i v případě, kdy uživatelé posílají do chatu své tipy a ty se zobrazují ostatním. Pokud uživatel uhodne, je ostatním hráčům zaslána pouze prázdná zpráva, kterou není možné zneužít, a frontend se postará o zobrazení zástupné hodnoty.

8.3.3 CORS

Cross-Origin Resource Sharing je mechanismus, který umožňuje webovým aplikacím komunikovat s jinými zdroji na internetu. Je důležitý pro zabezpečení webových aplikací, protože brání neoprávněnému přístupu k citlivým datům nebo zneužití zdrojů.

Tento mechanismus byl využit v backendové části aplikace. Bylo přidáno oprávnění pro klientskou aplikaci, která je tak jediným zdrojem, který může se serverem komunikovat. Nehrozí tedy zneužití a čerpání cloudových prostředků z jiné aplikace.

ZÁVĚR

Cílem této bakalářské práce bylo analyzovat a implementovat realtime synchronizaci ve webových technologiích pomocí technologie SignalR. Práce se zaměřila na popis protokolů pro webové aplikace, jejich vývoj a techniky realtime komunikace. Dále se zabývala podrobným popisem SignalR frameworku, včetně propojení s JavaScript klientem.

V praktické části práce byla navržena a implementována aplikace sloužící jako online hra, ve které hráči kreslí a hádají slova a jejich akce se synchronizují v reálném čase pomocí SignalR. Backend aplikace byl vyvinut v ASP.NET Core a frontend využívá framework Angular spolu s knihovnami PrimeNG a NGXS. Pro hosting byla využita cloudová platforma Microsoft Azure.

Bylo také popsáno vystavení aplikace do Microsoft Azure a řešení zabezpečení v rámci aplikace. Při nasazování byla provedena optimalizace synchronizace s cílem šetřit prostředky cloudové služby, přičemž bylo nezbytné najít vhodný kompromis mezi efektivním využitím prostředků a poskytováním plynulého a příjemného uživatelského zážitku.

Výsledkem bakalářské práce je funkční aplikace, která demonstruje využití SignalR pro realtime synchronizaci v online hře, ale principy a techniky zde prezentované lze aplikovat i na jiné typy webových aplikací, které vyžadují synchronizaci v reálném čase. Práce přispívá k rozšíření poznatků o technologii SignalR a jejím využití v realtime synchronizaci ve webových aplikacích, k čemuž byla vypracována také prezentace sloužící jako výukový materiál pro předmět pokročilé webové technologie.

SEZNAM POUŽITÉ LITERATURY

[1] VOLLE, Adam. Web application | Definition, History, Development, Examples, Uses, & Facts. In: Britannica [online]. 2022 [cit. 2023-05-15]. Dostupné z: https://www.britannica.com/topic/Web-application

[2] What Is a Web Application? (With Benefits and Jobs). In: *Indeed* [online]. 2023 [cit. 2023-05-15]. Dostupné z: https://www.indeed.com/career-advice/career-development/what-is-web-application

[3] What is a Web Application?. In: *Javatpoint* [online]. 2021 [cit. 2023-05-15]. Dostupné z: https://www.javatpoint.com/web-application

[4] KRAWCZYK, Mikołaj. Frontend vs Backend – What's the Difference? Detailed Explanation. In: Droptica [online]. 2023 [cit. 2023-05-15]. Dostupné z: https://www.droptica.com/blog/frontend-vs-backend-whats-difference-detailed-explanation/

[5] LEMONAKI, Dionysia. Frontend VS Backend – What's the Difference?. In: FreeCodeCamp [online]. 2022 [cit. 2023-05-15]. Dostupné z: https://www.freecodecamp.org/news/frontend-vs-backend-whats-the-difference/

[6] POSPÍŠIL, Vojtěch. Nejčastěji používané síťové protokoly. In: ZonerCloud [online]. 2021 [cit. 2023-05-15]. Dostupné z: https://www.zonercloud.cz/magazin/nejcasteji-pouzivane-sitove-protokoly

[7] What is a protocol? | Network protocol definition. In: Cloudflare [online]. 2023 [cit. 2023-05-15]. Dostupné z: https://www.cloudflare.com/learning/network-layer/what-is-a-protocol/

[8] SHARMA, Ruchika. A Basic Understanding of Web Protocols: HTTP and HTTPS. In: DZone [online]. 2017 [cit. 2023-05-15]. Dostupné z: https://dzone.com/articles/easy-understanding-of-web-protocols-http-and-https

[9] FIELDING, Roy, Mark NOTTINGHAM a Julian RESCHKE. RFC 9110: HTTP Semantics. In: Internet Engineering Task Force [online]. 2022 [cit. 2023-05-15]. Dostupné z: https://datatracker.ietf.org/doc/html/rfc9110

[10] RUAMVIBOONSUK, Vaspol. HTTP. In: Web Almanac [online]. 2022 [cit. 2023-05-15]. Dostupné z: https://almanac.httparchive.org/en/2022/http

[11] HTTP/2 vs. HTTP/3 usage statistics. In: W3Techs [online]. 2023 [cit. 2023-05-15]. Dostupné z: https://w3techs.com/technologies/comparison/ce-http2,ce-http3

[12] ABCOM. HTTP/1.1 vs HTTP/2: What's the Difference?. In: DigitalOcean [online]. 2019 [cit. 2023-05-15]. Dostupné z: https://www.digitalocean.com/community/tutorials/http-1-1-vs-http-2-what-s-the-difference

[13] KAUSHIK, Gunjan. What is Client Server Architecture and HTTP Protocol?. In: Tools QA [online]. 2022 [cit. 2023-05-15]. Dostupné z: https://www.toolsqa.com/client-server/client-server-architecture-and-model

[14] HTTP. In: MDN Web Docs [online]. 2023 [cit. 2023-05-15]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP

[15] HTTP request methods. In: MDN Web Docs [online]. 2023 [cit. 2023-05-15]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

[16] Safe (HTTP Methods). In: MDN Web Docs [online]. 2023 [cit. 2023-05-15]. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/Safe/HTTP

[17] HTTP Messages. In: MDN Web Docs [online]. 2023 [cit. 2023-05-15]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages

[18] HTTP - Responses. In: Tutorials Point [online]. 2023 [cit. 2023-05-15]. Dostupné z: https://www.tutorialspoint.com/http/http_responses.htm

[19] KOĎOUSKOVÁ, Barbora. HTTPS v kostce: co to je, jak funguje a jak na něj přejít. In: Rascasone [online]. 2021 [cit. 2023-05-15]. Dostupné z: https://www.rascasone.com/cs/blog/co-je-https-http-ssl-tls

[20] What is TCP/IP in Networking?. In: Fortinet [online]. 2023 [cit. 2023-05-15]. Dostupné z: https://www.fortinet.com/resources/cyberglossary/tcp-ip

[21] Polyfill. In: MDN Web Docs [online]. 2023 [cit. 2023-05-16]. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/Polyfill

[22] Cross-site WebSocket hijacking. In: PortSwigger [online]. 2023 [cit. 2023-05-20]. Dostupné z: https://portswigger.net/web-security/websockets/cross-site-websocket-hijacking

[23] Writing WebSocket servers. In: MDN Web Docs [online]. 2023 [cit. 2023-05-16]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API/Writing_WebSocket_servers

[24] MELNIKOV, Alexey a Ian FETTE. RFC 6455: The WebSocket Protocol. In: Internet Engineering Task Force [online]. 2011 [cit. 2023-05-16]. Dostupné z: https://datatracker.ietf.org/doc/html/rfc6455

[25] WRIGHT, Gavin. Real-time Transport Protocol (RTP). In: TechTarget [online]. 2021 [cit. 2023-05-16]. Dostupné z: https://www.techtarget.com/searchnetworking/definition/Real-Time-Transport-Protocol

[26] Introduction to the Real-time Transport Protocol (RTP). In: MDN Web Docs [online]. 2023 [cit. 2023-05-16]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Intro_to_RTP

[27] PRUETT, Michael. SIP and RTP/RTCP. In: Fortinet GURU [online]. 2018 [cit. 2023-05-16]. Dostupné z: https://www.fortinetguru.com/2018/10/sip-and-rtp-rtcp/

[28] PERKINS, Colin. RTP: Audio and Video for the Internet. Addison-Wesley Professional, 2003. ISBN 0672322498.

[29] SRTP. In: GoTo [online]. 2023 [cit. 2023-05-16]. Dostupné z: https://www.goto.com/resources/glossary/srtp

[30] BASUMALLICK, Chiradeep. What Is RTP (Real-Time Transport Protocol)? Meaning, Working, and Applications. In: Spiceworks [online]. 2022 [cit. 2023-05-16]. Dostupné z: https://www.spiceworks.com/tech/networking/articles/what-is-rtp/

[31] GARRETT, Jesse James. Ajax: A New Approach to Web Applications. In: University of Washington [online]. AdaptivePath.com, 2005 [cit. 2023-05-16]. Dostupné z: https://courses.cs.washington.edu/courses/cse490h/07sp/readings/ajax_adaptive_path.pdf

[32] SHAMIL, Fazal Rehman. History of AJAX. In: T4Tutorials [online]. 2023 [cit. 2023-05-16]. Dostupné z: https://t4tutorials.com/history-of-ajax/

[33] AWATI, Rahul. AJAX (Asynchronous JavaScript and XML). In: TheServerSide [online]. 2022 [cit. 2023-05-16]. Dostupné z: https://www.theserverside.com/definition/Ajax-Asynchronous-JavaScript-and-XML

[34] WIGMORE, Ivy. Single-page application (SPA). In: TechTarget [online]. 2016 [cit. 2023-05-16]. Dostupné z: https://www.techtarget.com/whatis/definition/single-page-application-SPA

[35] KOĎOUSKOVÁ, Barbora. Co je jednostránková webová aplikace (SPA) a kdy ji využít?. In: Rascasone [online]. 2021 [cit. 2023-05-16]. Dostupné z: https://www.rascasone.com/cs/blog/jednostrankova-webova-aplikace-spa

[36] Signal. In: Signal [online]. 2023 [cit. 2023-05-16]. Dostupné z: https://www.signal.org/

[37] Trello. In: Atlassian [online]. 2023 [cit. 2023-05-16]. Dostupné z: https://www.atlassian.com/software/trello

[38] LUTKEVICH, Ben. Real-time application (RTA). In: TechTarget [online]. 2022 [cit. 2023-05-16]. Dostupné z: https://www.techtarget.com/searchunifiedcommunications/definition/real-time-application-RTA

[39] ZACK. 5 Ways to Sync Web and Server Data in Real Time. In: Bits and Pieces [online]. 2022 [cit. 2023-05-16]. Dostupné z: https://blog.bitsrc.io/5-ways-to-sync-web-and-server-data-in-real-time-3b5748ad2bf7

[40] AGGARWAL, Anuradha. Short Polling vs Long Polling vs Web Sockets. In: DEV Community [online]. 2022 [cit. 2023-05-16]. Dostupné z: https://dev.to/anuradha9712/short-polling-vs-long-polling-vs-web-sockets-3h36

[41] SATWIKSUMAN. What is Long Polling and Short Polling ?. In: GeeksforGeeks [online]. 2022 [cit. 2023-05-16]. Dostupné z: https://www.geeksforgeeks.org/what-is-long-polling-and-short-polling/

[42] LAMONACA, Michael. Long Polling. In: Device42 [online]. 2023 [cit. 2023-05-16]. Dostupné z: https://support.device42.com/hc/en-us/articles/14033101049623-Long-Polling

[43] PANDA, Sandeep. Introduction to the HTML5 WebSockets API. In: SitePoint [online]. 2013 [cit. 2023-05-16]. Dostupné z: https://www.sitepoint.com/introduction-to-the-html5-websockets-api/

[44] DIKSHAMULCHANDANI1. Difference between Long Polling and WebSocket. In: GeeksforGeeks [online]. 2021 [cit. 2023-05-16]. Dostupné z: https://www.geeksforgeeks.org/difference-between-long-polling-and-websocket/

[45] Writing WebSocket client applications. In: MDN Web Docs [online]. 2023 [cit. 2023-05-16]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API/Writing_WebSocket_client_applications

[46] BEVERLOO, Peter, Martin THOMSON a Marcos CACERES, ed. Push API. In: World Wide Web Consortium [online]. 2022 [cit. 2023-05-16]. Dostupné z: https://www.w3.org/TR/push-api/

[47] Using Service Workers. In: MDN Web Docs [online]. 2023 [cit. 2023-05-16]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API/Using_Service_Workers

[48] LIEBEL, Christian. Push API: Advanced Progressive Web Apps – Push Notifications Under Control – Part 2. In: Thinklecture [online]. 2020 [cit. 2023-05-16]. Dostupné z: https://www.thinktecture.com/en/angular/push-api/

[49] Server-sent events. In: Web Hypertext Application Technology Working Group [online]. 2023 [cit. 2023-05-16]. Dostupné z: https://html.spec.whatwg.org/multipage/server-sent-events.html

[50] MARTIN, Eve. WebSockets vs Server-Sent Events: Key differences and which to use. In: Ably [online]. 2023 [cit. 2023-05-16]. Dostupné z: https://ably.com/blog/websockets-vs-sse

[51] The WebSocket API (WebSockets). In: MDN Web Docs [online]. 2023 [cit. 2023-05-16]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

[52] Using server-sent events. In: MDN Web Docs [online]. 2023 [cit. 2023-05-16]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events/Using_server-sent_events

[53] Overview of ASP.NET Core SignalR. In: Microsoft Learn [online]. 2023 [cit. 2023-05-16]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/signalr/introduction

[54] SAZANAVETS, Fiodar. SignalR on .NET 6 - the Complete Guide: The easiest way to enable real-time two-way HTTP communication on .NET 6. USA: Independently published, 2022. ISBN 979-8796211779.

[55] CONROY, Brennan. Manage users and groups in SignalR. In: Microsoft Learn [online]. 2022 [cit. 2023-05-16]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/signalr/groups

[56] APPEL, Rachel a Kevin GRIFFIN. Use hubs in SignalR for ASP.NET Core. In: Microsoft Learn [online]. 2023 [cit. 2023-05-16]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/signalr/hubs

[57] CONROY, Brennan. Use streaming in ASP.NET Core SignalR. In: Microsoft Learn [online]. 2022 [cit. 2023-05-16]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/signalr/streaming

[58] SignalR vs. WebSocket: Which is best for your realtime app?. In: Ably [online]. 2022 [cit. 2023-05-16]. Dostupné z: https://ably.com/topic/signalr-vs-websocket

[59] Real-time ASP.NET with SignalR. In: Microsoft [online]. 2023 [cit. 2023-05-16]. Dostupné z: https://dotnet.microsoft.com/en-us/apps/aspnet/signalr

[60] ROTH, Daniel, Rick ANDERSON a Shaun LUTTIN. Overview of ASP.NET Core. In: Microsoft Learn [online]. 2022 [cit. 2023-05-16]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core

[61] VEMULA, Rami. Real-Time Web Application Development: With ASP.NET Core, SignalR, Docker, and Azure. Berkeley, CA: Apress, 2017. ISBN 978-1-4842-3269-9.

[62] FRANKLIN, Sean. Picking Between Blazor Server And Blazor WebAssembly. In: C# Corner [online]. 2022 [cit. 2023-05-16]. Dostupné z: https://www.c-sharpcorner.com/article/picking-between-blazor-server-and-blazor-webassembly/

[63] SPASOJEVIC, Marinko. Blazor Server vs Blazor WebAssembly, Pros and Cons. In: Code Maze [online]. 2022 [cit. 2023-05-16]. Dostupné z: https://code-maze.com/blazor-webassembly-introduction/

[64] HILTON, Jon. Blazor vs Angular. In: Telerik [online]. 2020 [cit. 2023-05-16]. Dostupné z: https://www.telerik.com/blogs/blazor-vs-angular-web-developers

[65] KOĎOUSKOVÁ, Barbora. Co je Angular, v čem je jiný než AngularJS a proč ho použít?. In: Rascasone [online]. 2021 [cit. 2023-05-16]. Dostupné z: https://www.rascasone.com/cs/blog/co-je-angular-angularjs

[66] Getting started with Angular. In: MDN Web Docs [online]. 2023 [cit. 2023-05-16]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Angular_getting_started

[67] PrimeNG. In: PrimeNG [online]. 2023 [cit. 2023-05-16]. Dostupné z: https://primeng.org/

[68] NGXS. In: NGXS [online]. 2023 [cit. 2023-05-16]. Dostupné z: https://www.ngxs.io/

[69] Microsoft Azure. In: Microsoft [online]. 2023 [cit. 2023-05-16]. Dostupné z: https://azure.microsoft.com/cs-cz/

[70] Rider: The Cross-Platform .NET IDE from JetBrains. In: JetBrains [online]. 2023 [cit. 2023-05-16]. Dostupné z: https://www.jetbrains.com/rider/

[71] THORNDYKE, Kattie. 8 Popular IDEs and Code Editors. In: Codecademy [online]. 2021 [cit. 2023-05-16]. Dostupné z: https://www.codecademy.com/resources/blog/popular-ides-and-code-editors/

[72] Visual Studio Code. In: Visual Studio [online]. 2023 [cit. 2023-05-16]. Dostupné z: https://code.visualstudio.com/

[73] BUCHHOLZ-POWERS, Carrie. What Is Figma? A Complete Introduction for Beginners. In: Skillshare [online]. 2022 [cit. 2023-05-16]. Dostupné z: https://www.skillshare.com/en/blog/what-is-figma-a-complete-introduction-for-beginners/

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
CDN	Content Delivery Network
CORS	Cross-Origin Resource Sharing
CSS	Cascading Style Sheets
DOM	Document Object Model
GUID	Globally Unique Identifier
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IDE	Integrated Development Environment
IETF	Internet Engineering Task Force
JSON	JavaScript Object Notation
NPM	Node Package Manager
RTCP	Real-Time Control Protocol
RTP	Real-Time Transport Protocol
SEO	Search Engine Optimization
SPA	Single-Page Application
SRTP	Secure Real-Time Transport Protocol
SSE	Server-Sent Events
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
URI	Uniform Resource Identifier

URL	Uniform Resource Locator
UTF-8	Unicode Transformation Format - 8-bit
VoIP	Voice over Internet Protocol
VS Code	Visual Studio Code
W3C	World Wide Web Consortium
WPF	Windows Presentation Foundation
XHR	XMLHttpRequest
XML	eXtensible Markup Language
XOR	Exclusive OR

SEZNAM OBRÁZKŮ

Obrázek 1. Schéma architektury klient-server [13] 13
Obrázek 2. Microsoft Excel v prohlížeči 22
Obrázek 3. Aplikace Signal [36] 23
Obrázek 4. Aplikace Trello [37] 24
Obrázek 5. Obchodovací platforma xStation 5 24
Obrázek 6. Hra Agar.io 25
Obrázek 7. Návrh rozhraní v aplikaci Figma 47
Obrázek 8. Seznam slov v souboru WordList.json 53
Obrázek 9. Přihlašovací stránka 80
Obrázek 10. Lobby hry 82
Obrázek 11. Herní místnost 85
Obrázek 12. Frekvence zpráv eventu mousemove 93
Obrázek 13. Frekvence zpráv optimalizované funkce 95
Obrázek 14. Vytvoření prostředku v Microsoft Azure 96
Obrázek 15. Publikování aplikace v JetBrains Rider 96

SEZNAM TABULEK

Tabulka 1. Vlastnosti objektu Context 32
Tabulka 2. Metody objektu Context 33
Tabulka 3. Vlastnosti objektu Clients...33
Tabulka 4. Metody objektu Clients...33
Tabulka 5. SignalR balíčky pro klienty 35

SEZNAM PŘÍLOH

Příloha P I: CD

PŘÍLOHA P I: CD

CD obsahuje:

- Zdrojové kódy aplikace v ASP.NET Core: prilohy.zip / DrawAndGuessAPI
- Zdrojové kódy aplikace v Angularu: prilohy.zip / DrawAndGuess
- Prezentace určená k výuce o SignalR: prilohy.zip / prezentace.pptx
- Diplomová práce ve formátu PDF: fulltext.pdf