Univerzita Hradec Králové
Fakulta informatiky a managementu
katedra informatiky a kvantitativních metod

BAKALÁŘSKÁ PRÁCE

Real-time komunikace ve webových aplikachích

Autor: Miroslav Pokorný

Vedoucí práce: Ing. Ondřej Klapka

Duben, 2017

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a uvedl všechny použité prameny a literaturu.

V Hradci Králové dne 12. dubna 2017 Miroslav Pokorný

Poděkování

Rád bych poděkoval vedoucímu bakalářské práce Ing. Ondřeji Klapkovi za věcné připomínky a vedení práce. Dále bych rád poděkoval svým rodičům za podporu při studiu a psaní této práce.

Anotace

Tato bakalářská práce je rozdělena na dvě hlavní části. V první části se zabývá možnostmi navázání komunikace v reálném čase v prostředí webových aplikaci. V této části se nachází charakteristika, jednoduchá ukázka implementace a porovnání jednotlivých technik komunikace v reálném čase mezi klientem (nejčastěji webovým prohlížečem) a serverem. Druhá část této práce je návrh a implementace komunikační knihovny, použitelné v prostředí webových aplikací, které vyžadují komunikaci v reálném čase. Knihovna podporuje odesílání libovolných binárních dat, včetně podpory pro snadný přenos a příjem multimediálních dat.

Annotation

This Bachelor Thesis is divided into two main parts. The first part deals with possible ways how to establish a real-time connection in web applications environment. In this section are a description, simple implementation example, and comparison of each individual methods of communication in real-time between a client (usually a web browser) and server. The second part of this work is design and Implementation of a communication library, usable in web applications environment, which requires communication in real-time. The library supports sending of raw binary data, including support for simple sending and receiving of multimedia data.

Obsah

1. Úvod

Webové aplikace jsou čím dál častější formou tvorby aplikací a pomalu nahrazují klasické desktopové aplikace, stávají se standardem tvorby aplikací od velmi jednoduchých aplikací po složité informační systémy. Webové aplikace jsou tedy z praktického hlediska velice žádané a jejich vývoj je perspektivní. Aby se zlepšil uživatelský dojem z webových aplikací, je vhodné je doplňovat o funkcionality, které uživateli zpříjemní používání dané webové aplikace. Zpříjemněním v tomto kontextu se převážně jedná o snížení odezvy mezi uživatelskou akcí a následnou odpovědí webového serveru (aplikace). Z tohoto důvodu je dobré věnovat se možnostem komunikace v reálném čase, které mohou tuto odezvu minimalizovat a mnohdy i zcela odstranit nutnost uživatele „žádat“ o nějaká data a místo toho mu je poskytnout automaticky v okamžiku, kdy jsou dostupná.

1.1. Cíl práce

Tato práce je rozdělena na dvě hlavní části. Cílem první části je seznámení čtenáře s možnostmi navázání komunikace v reálném čase v prostředí webových aplikací. Čtenář by měl získat základní přehled o tom, jak fungují webové aplikace a jak fungují jednotlivé techniky komunikace v reálném čase.

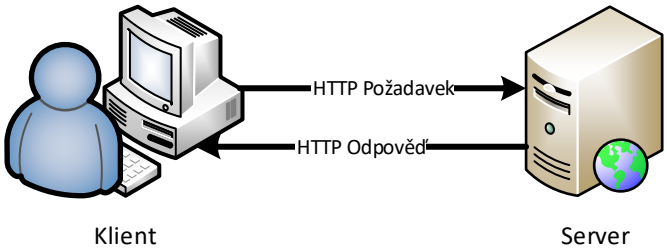
V druhé části této práce je popisován návrh a implementace knihovny umožňující komunikaci v reálném čase v prostředí webových aplikací. Tato část seznámí čtenáře s úskalím výběru platformy, formátu přenášených dat a následný návrh chování a implementaci samotné knihovny.

2. Techniky komunikace

V následujících podkapitolách budou popsány jednotlivé technologie, pomocí kterých lze komunikovat v prostředí webových aplikací a docílit komunikace v reálném čase mezi klientem a serverem.

2.1. HTTP komunikace

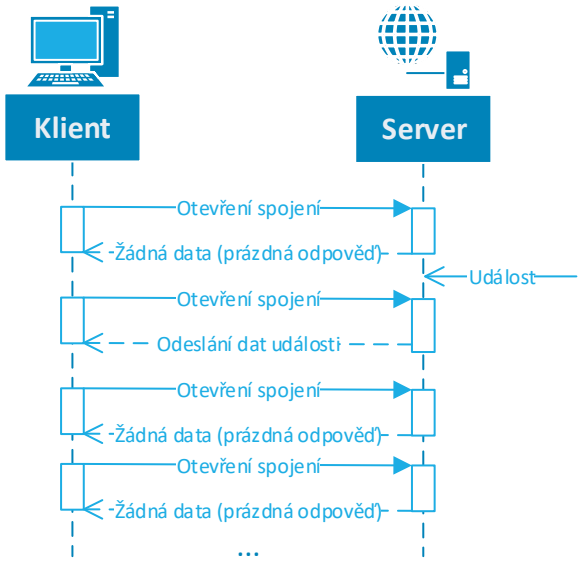
Klasická HTTP komunikace funguje na principu požadavku (request) a odpovědi (response). Z tohoto principu vyplývá, že komunikaci iniciuje uživatel odesláním požadavku na server. Na serveru je požadavek zpracován a následně je uživateli odeslána odpověď. Klasickou HTTP komunikaci pomocí HTTP mezi klientem a serverem lze vidět na obrázku 2.1. Klasická HTTP komunikace je bezstavová, což znamená, že klient ani server si neukládají žádné informace o komunikaci, ale jakmile je požadavek vyřízen, je spojení uzavřeno. Při dalším požadavku musí být spojení opět otevřeno. [1]



Obrázek 2.1.: Klasická HTTP komunikace

2.2. Polling

Polling (někdy nazývaný Short Polling) funguje na periodickém odesílání požadavku na data na server (např. každých 500 milisekund). Pokud server má nějaká data, tak je v tomto požadavku odešle klientovi, pokud žádná data připravená nemá, odešle prázdnou odpověď. Tímto mechanizmem lze docílit, že webová stránka reaguje téměř



Obrázek 2.2.: Sekvenční diagram znázorňující průběh fungování Polling komunikace

okamžitě po vyvolání události. Jedinou možností jak dostat data ke klientovi je, že klient pošle požadavek na server. Server nemá možnost jak kontaktovat klienta ihned po té, co nastane událost (doba odezvy bude záležet na frekvenci odesílání požadavků na server a také na době, nutné ke zpracování požadavku). Nevýhodou tohoto řešení je vytváření spojení každý definovaný interval, a to i za předpokladu, že server nemá žádná nová data pro odeslání. [2] Princip Pollingu lze vidět na obrázku 2.2.

2.2.1. Implementace na straně klienta

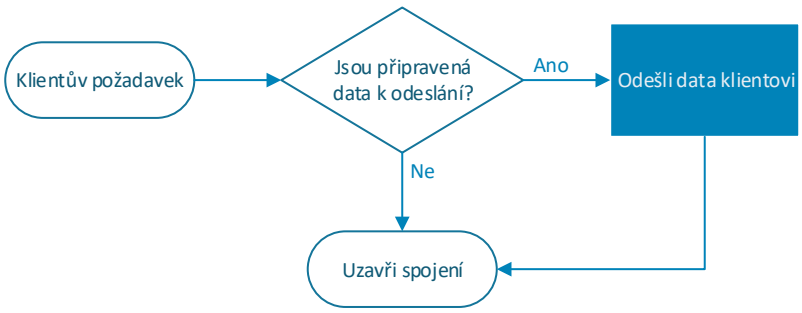
Na ukázce kódu 2.1 lze vidět příklad implementace Polling komunikace na straně klienta pomocí jazyku JavaScript. Na ukázce lze vidět jednu funkci „polling“, která je volána pomocí funkce „setInterval“ každé cca 3 sekundy. Funkce „polling“ nejprve otestuje, zda mezi klientem a serverem už neexistuje spojení, které ještě nebylo dokončeno (tj. spojení stále probíhá). Pokud spojení ještě neexistuje, nebo již bylo dokončeno, je vytvořeno nové spojení pomocí rozhraní XMLHttpRequest. Tomuto rozhraní je přidán jeden listener, který odchytává událost „load“, která je vyvolána ve chvíli, kdy klient obdrží data od serveru. [3] Pomocí metody „open“ se nastaví parametry spojení. Jako první parametr je metoda HTTP komunikace (např. GET, POST,...), druhý parametr je URL adresa, na které budou distribuovány updaty komunikace.

```
var xhr;
function polling() {
    if (xhr != null && xhr.readyState != XMLHttpRequest.DONE) return;
    xhr = new XMLHttpRequest();
    xhr.addEventListener("load", function (event) {
        alert(xhr.response);
    })
    xhr.open("GET", "http://example.com/pollingSource");
    xhr.send();
}
setInterval(polling, 3000);
```

Ukázka kódu 2.1: Implementace Polling komunikace na straně klienta pomocí jazyku JavaScript

2.2.2. Implementace na straně serveru

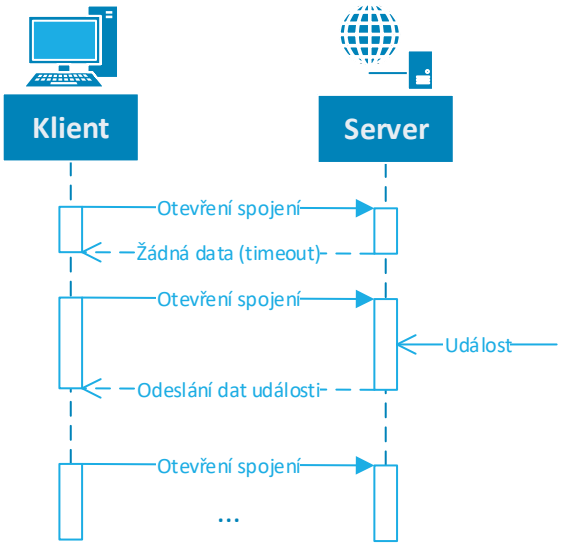
Implementace na straně serveru je velice jednoduchá. Na obrázku 2.3 je vývojový diagram znázorňující funkcionalitu serveru při využití Polling komunikace. Na serveru se po obdržení požadavku od klienta pouze zkontroluje, jestli jsou připravená nějaká data k odeslání, pokud ano, server data odešle, v opačném případě server vrátí prázdnou odpověď a spojení je ukončeno.



Obrázek 2.3.: Vývojový diagram zpracování požadavku klienta na serveru při využití Polling komunikace

2.3. Long Polling

Jelikož se spojení mezi klientem a serverem po každém požadavku uzavře, není v běžné komunikaci možnost, aby server kontaktoval klienta. Jednou z možností, jak této funkcionality docílit, je použít metodu Long Polling. Long Polling funguje podobně jako Polling s tím rozdílem, že klient otevře spojení se serverem a nechá je otevřené, server si převezme spojení a počká až do doby, než nastane nějaká událost (než bude mít



Obrázek 2.4.: Sekvenční diagram znázorňující průběh fungování Long Polling komunikace

připravená nějaká data k odeslání). Jakmile nastane nějaká událost, server odešle data klientovi a spojení je ukončeno. Jelikož bylo spojení ukončeno, je nutné, aby klient opět inicioval nové spojení se serverem, aby server mohl odeslat data v okamžiku, kdy nastane událost. Pokud odeslání dat trvá příliš dlouho, zareaguje timeout mechanizmus (je specifikován přesný časový interval, když tento interval uplyne, nastane tzv. timeout), který spojení ukončí, aniž by byla odeslána data klientovi. Tento princip se opakuje, dokud je požadováno, aby klient komunikoval se serverem. [2] Znázornění, jak Long Polling funguje, lze vidět na obrázku 2.4. Long Polling je podporován ve všech hlavních webových prohlížečích (tj. Chrome, Firefox, Opera, Internet Explorer a Edge).

2.3.1. Implementace na straně klienta

Na ukázce kódu 2.2 lze vidět jednoduchou implementaci Long Polling komunikace na straně klienta napsanou v jazyce JavaScript. V ukázce je definována jedna funkce „newConnection“, tato funkce vytvoří komunikační objekt pomocí rozhraní XMLHttpRequest, kterému je nastaven čas timeoutu (v tomto případě 30000 ms = 30 sekund). [3] Dále jsou přiřazeny listenery událostí a to konkrétně listener události „timeout“, která nastane v okamžiku, když klient neobdrží odpověď. Druhým listenerem v této ukázce je listener události „load“, tato událost je vyvolána v okamžiku, kdy klient obdrží celou

odpověď. Pomocí metody „open“ se nastaví požadovaná metoda HTTP komunikace (tj. GET, POST,...). Jako poslední příkaz funkce „newConnection“ je volána metoda „send“, tato metoda odešle požadavek na server.

Spojení mezi klientem a serverem je uzavřeno v okamžiku, kdy server odešle odpověď klientovi, nebo v okamžiku, když vyprší timeout čas. Z tohoto důvodu je zapotřebí znovu vytvořit spojení mezi klientem a serverem, proto obsluha události „load“ volá znovu funkci „newConnection“, která vytvoří nové spojení.

```
var xhr;
function newConnection() {
    xhr = new XMLHttpRequest();
    xhr.timeout = 30000; //ca. 30 seconds timeout
    xhr.addEventListener("timeout", timeoutHandler);
    xhr.addEventListener("load", function (event) {
        alert(xhr.response);
        newConnection();
    });
    xhr.open("GET", "http://example.com/longPollingSource");
    xhr.send();
}
newConnection();
```

Ukázka kódu 2.2: Implementace Long Polling komunikace na straně klienta v jazyce JavaScript

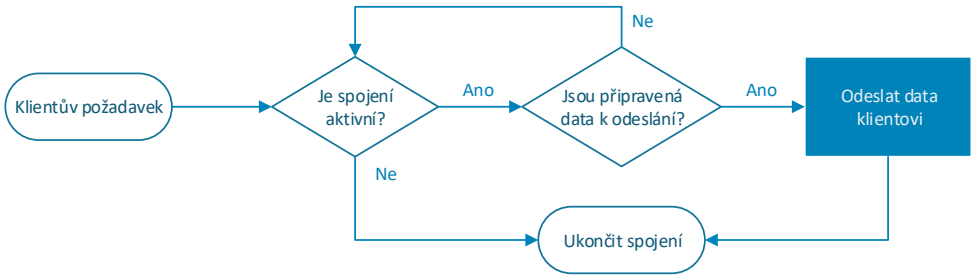
Tabulka 2.1 shrnuje události, které mohou být vyvolány rozhraním XMLHttpRequest. Jak lze v tabulce vidět, některé události mohou nastat pouze jednou.

2.3.2. Implementace na straně serveru

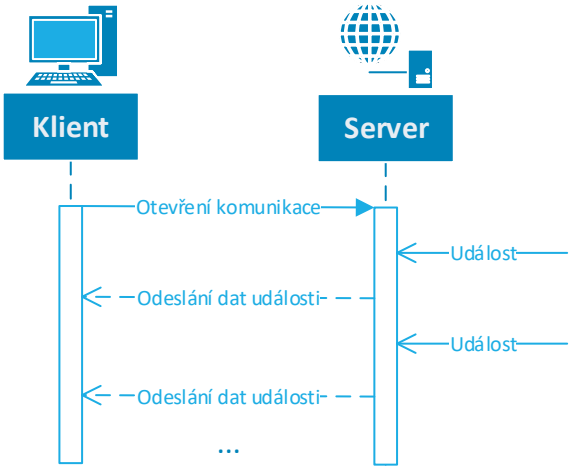
Implementace na straně serveru se nebude příliš lišit od implementace klasické HTTP odpovědi. Na obrázku 2.5 lze vidět jednoduchý vývojový diagram zachycující funkci serveru, obsluhujícího Long Polling komunikaci. Pokud bude mít server připravená data v okamžiku, kdy mu přijde požadavek, tak je odešle klientovi. Pokud data nemá, pozdrží odpověď do doby, než nějaká data bude mít, nebo dokud nevyprší timeout.

2.4. HTTP Streaming

HTTP Streaming funguje na principu otevření jednoho spojení mezi klientem a serverem. Toto spojení není nikdy uzavřeno a server postupně pomocí něj odesílá data klientovi. Komunikace pomocí HTTP Streamingu je iniciována klientem. Server obdrží požadavek a pozdrží jej, dokud nemá potřebná data, nebo dokud nenastane timeout. Jakmile má server potřebná data, odešle je klientovi. Rozdílem oproti Pollingu (nebo Long Pollingu) je, že odeslání dat nezpůsobí ukončení spojení mezi klientem a serverem. Klient tak nemusí znovu odesílat požadavek na otevření spojení. Jakmile bude mít server připravená data k odeslání, budou pomocí tohoto spojení odeslána klientovi. [2] Obrázek 2.6 znázorňuje jak HTTP Streaming funguje.



Obrázek 2.5.: Vývojový diagram, zpracování klientova požadavku na serveru, při Long Polling komunikaci



Obrázek 2.6.: Sekvenční diagram znázorňující funkci komunikace pomocí HTTP Streaming

Tabulka 2.1.: Možné události rozhraní XMLHttpRequest a jejich význam. Zdroj [3]

Druh události	Popis	Kolikrát nastane	Kdy nastane
loadstart	Volána, když proces začne	Jednou	Jako první (hned na začátku)
progress	Volána, v okamžiku, kdy je proces aktivní	Jednou nebo vícekrát	Poté co nastane „loadstart“
error	Volána, když proces selže	Ani jednou nebo jednou (tyto události se vzájemně vylučují, tj. může nastat právě jedna možnost)	Poté co nastal poslední „progress“
abort	Volána, když je proces přerušen (ukončen)		
timeout	Volána, když je proces přerušen, kvůli vypršení času		
load	Volána, když je proces úspěšně dokončen		
loadend	Volána, když proces skončí	Jednou	Poté co nastane jeden z „error“, „abort“, „timeout“ nebo „load“

2.4.1. Forever frame

Forever frame využívá HTTP Streamingu, konkrétně využívá vlastnosti HTTP/1.1 „Chunked encoding“. Jelikož server dopředu neví, jak velká bude celá komunikace, nemůže v hlavičce odpovědi uvést velikost zprávy (vlastnost „Content-length“), proto odpoví s vlastností „chunked encoding“, díky tomu může server odesílat data, aniž by věděl dopředu, jak velká bude celá komunikace. Data jsou rozdělena na tzv. „chunky“, které jsou následně odesílány klientovi. [4] V ukázce kódu 2.3 lze vidět HTTP hlavičku chunkované komunikace a následně dvě zprávy, každá zpráva je předcházena hexadecimálním číslem, které udává velikost chunku, toto číslo je ukončeno odřádkováním, na dalším řádku je pak samotná zpráva. Poslední chunk se vyznačuje velikostí rovnou nule (0).

```
HTTP/1.1 200 OK
Content-Type: text/plain
Transfer-Encoding: chunked

25
This is the data in the first chunk

1C
and this is the second one

0
```

Ukázka kódu 2.3: Ukázka odpovědi při chunkované komunikaci. Zdroj: [2]

2.4.1.1. Implementace na straně klienta

Na ukázce kódu 2.4 lze vidět ukázkovou implementaci Forever frame na straně klienta (Webového prohlížeče). Do zdrojového kódu webové aplikace se vloží nejprve element „script“, který přidává knihovnu s funkcemi. Jako druhý je přidán element „iframe“, který má v atributu „src“ vloženou URL adresu místa, kde budou distribuovány updaty. Tento prvek je pomocí stylů CSS skryt, aby nerušil vykreslování stránky. V elementu „iframe“ budou přijímány zprávy tak, že budou obaleny HTML značkou „script“ a uvnitř bude JavaScriptový kód volat funkce z knihovny (v tomto případě „ForeverFrameLibrary.js“). Přístup k těmto funkcím bude z iframe pomocí tečkové notace, kde bude jako prefix použito klíčové slovo „parent“ (tj. např. „parent.ZobrazData(„data“);“). [4]

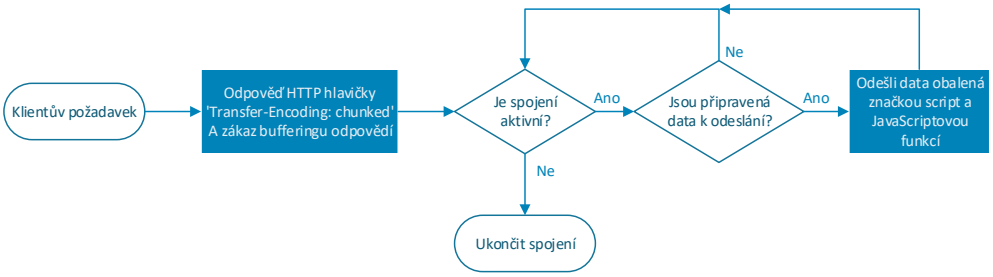
Aby nedocházelo k nadměrnému využití paměti, je dobré DOM elementy v prvku „iframe“ odstranit hned poté co budou zpracovány.

```
<script src="ForeverFrameLibrary.js" type="text/javascript"></script>
<iframe src="http://example.com/FFSource" style="display: none;"></iframe>
```

Ukázka kódu 2.4: Implementace Forever frame komunikace na straně klienta

2.4.1.2. Implementace na straně serveru

Implementace Forever frame komunikace na straně serveru se příliš neliší od implementace Long Polling komunikace, až na to, že tento druh komunikace využívá jedno spojení mezi klientem a serverm, které po odeslání zprávy neuzavře. Na obrázku 2.7 lze vidět vývojový diagram zachycující funkcionalitu serveru při využití tohoto druhu komunikace. Nejprve je zapotřebí v HTTP hlavičce odpovědi specifikovat, že se budou data mezi serverem a klientem přenášet pomocí chunků. Dále je vhodné zakázat na serveru buffering odpovědí, aby nedocházelo k pozdržení odeslání zpráv. Poté server



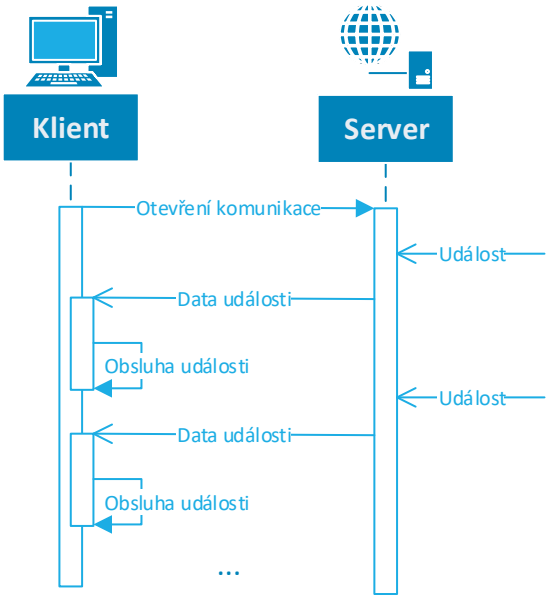
Obrázek 2.7.: Vývojový diagram zpracování klientova požadavku na straně serveru, při Forever frame komunikaci

zkontroluje, zda je spojení mezi klientem a serverem stále aktivní. Pokud ano, tak zkontroluje, jestli jsou připravená nějaká data k odeslání. Pokud ano, tak je obalí do HTML značky „script“ a JavaScriptové funkce. Některé platformy po nastavení „chunked-encodig“ začnou automaticky odesílat data jako chunky (tj. formát odpovědi bude automaticky odpovídat formátu odpovědi z ukázky kódu 2.3), jiné platformy toto automaticky nedělají a je zapotřebí zprávy ručně formátovat (tj. před odesláním dat je nutné spočítat jejich velikost a odeslat ji před samotnou zprávou), aby odpověď byla ve formátu, který lze vidět v ukázce kódu 2.3.

Různé webové prohlížeče pracují s elementem „iframe“ rozdílně. V některých webových prohlížečích může nastat zpoždění vykreslování (např. v Safari), do té doby, než přijmou nějaké předem určené množství dat, z tohoto důvodu je vhodné odeslat na začátku komunikace „prázdnou odpověď“, která bude mít velikost 1kB (nejčastěji to bude ve formátu mezer). [4] Prohlížeč Internet Explorer zase požaduje nějaký neprázdný HTML element, aby vše fungovalo správně. Proto je vhodné ještě za HTML značku „script“ přidat HTML značku „br“.

2.5. Server-Sent Events

Server-Sent Events (SSE) fungují tak, že poskytují klientovi (např. webový prohlížeč) API (v JavaScriptu se konkrétně použije rozhraní EventSource), pomocí kterého se vytvoří spojení se serverem. Pomocí tohoto spojení pak server odesílá data klientovi. Data jsou v prohlížeči zpracována jako DOM události. Jako v předchozích případech (Forever frame, HTTP Streaming, Long Polling a Polling) jsou data odesílána za pomoci protokolu HTTP. [5] Obrázek 2.8 znázorňuje, jak Server-Sent Events fungují. Výhodou tohoto typu spojení je, že data odesílá server sám automaticky, není nutné, aby se klient sám dotazoval serveru, zda jsou pro něj připravená nějaká data. V případě, že je



Obrázek 2.8.: Sekvenční diagram znázorňující fungování Server-Sent Events

spojení mezi klientem (webovým prohlížečem) a serverem ztraceno, klient se spojení sám pokusí obnovit, a je vyvolána událost, která může být odchycena a zpracována.

2.5.1. Formát zpráv

Na ukázce kódu 2.5 lze vidět formát zpráv odesílaných serverem ke klientovi. Zprávy v tomto formátu se označují pomocí MIME type „text/event-stream“. [6] V tomto případě jsou využity 2 druhy událostí:

1. Výchozí událost („data-only“) - jsou odeslány rovnou data bez pojmenování události, tato událost bude pojmenována výchozím jménem události (výchozí jméno je „message“)

2. Pojmenované události - data jsou předcházena jménem události. Na této ukázce jsou události pojmenovány jménem „Udalost1“ a „Udalost2“

První událost je výchozí událostí (tj. událost „message“). Každá Událost je odeslána (ukončena) prázdným řádkem. Je jedno, jestli bude použito unixové ukončení řádku

(LF[1]) nebo windowsové ukončení řádku (CR+LF), popřípadě ukončení řádku (CR[2]).

Druhá událost je opět výchozí událostí, ale hodnota události je rozdělena na dva řádky (tj. hodnota obsahuje speciální znak odřádkování, v tomto případě to budou 2 řádky). Třetí zpráva je pojmenovanou událostí se jménem „Udalost2“. Čtvrtá zpráva je pojmenovanou událostí se jménem „Udalost1“. [7]

Na posledním řádku ukázky lze vidět komentář. Komentář nevyvolá žádnou událost. Komentáře se poznají tak, že řádek, na kterém jsou, začíná znakem dvojtečky „:“ (kód znaku v kódování Unicode je U+003A).

```
data: data výchozí události

data: data výchozí události
data: včetně speciálního znaku odřádkování

event: Udalost2
data: data Události 2

event: Udalost1
data: data Události 1

:řádek komentáře, žádná událost
```

Ukázka kódu 2.5: Formát zpráv Server-Sent Events

2.5.2. Implementace na straně klienta

Na ukázce kódu 2.6 lze vidět ukázkovou implementaci Server-Sent Events na straně klienta. V ukázce je použito rozhraní „EventSource“, pomocí kterého se naváže Server-Sent Events spojení se serverem. Rozhraní se v konstruktoru předá parametr s URL, na kterém budou distribuovány data událostí. Poté co je vytvořeno spojení, jsou k němu přiřazeny listenery událostí (v tomto příkladu nejprve listener pro nepojmenovanou událost [výchozí událost „message“], poté listener pojmenované události „Udalost1“, a nakonec listener pojmenované události „Udalost2“).

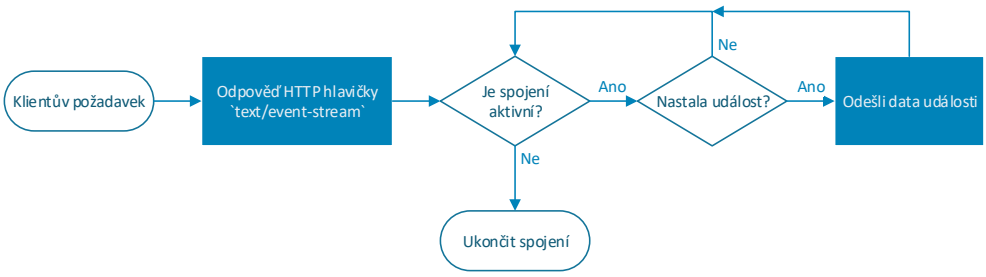
```
var source = new EventSource("http://example.com/updates");
source.addEventListener("message", function (event) {
    alert("Received data = " + event.data);
});
source.addEventListener("Udalost1", udalost1Handler);
source.addEventListener("Udalost2", udalost2Handler);
```

Ukázka kódu 2.6: Implementace Server-Sent Events na straně klienta (webového prohlížeče) v jazyce JavaScript

Pozn. ukázka kódu 2.6 je napsána tak, aby byla schopná zpracovávat zprávy z ukázky kódu 2.5.

[1] Line feed, kód znaku v kódování Unicode je U+000A (hexadecimálně 0x0A)

[2] Carriage return, kód znaku pomocí kódování Unicode je U+000D (hexadecimálně 0x0D)



Obrázek 2.9.: Vývojový diagram zpracování požadavku Server-Sent Events komunikace na straně serveru

Rozhraní EventSource má ve výchozím stavu tyto události:

- „open“, tato událost je volána v okamžiku, kdy je vytvořeno spojení mezi klientem a serverem
- „message“, událost je volána v momentě, kdy jsou přijata data, která nemají specifikovanou událost (výchozí událost [„data-only“])
- „error“, tato událost je volána, pokud nastane nějaká chyba (např. přerušení spojení)

2.5.3. Implementace na straně serveru

Implementace na straně serveru se provede podobně jako odpověď na jakýkoliv HTTP požadavek. Na obrázku 2.9 je znázorněn vývojový diagram, který znázorňuje jednotlivé kroky, které by měl server udělat.

Nejprve je nutné v hlavičce HTTP odpovědi uvést druh obsahu odpovědi na event-stream (tj. „Content-Type: text/event-stream“). [6] Jakmile je nastavena HTTP hlavička, může nastat samotné odesílání událostí. Formát zpráv odesílaných serverem bude odpovídat formátu zpráv z ukázky kódu 2.5. Server by měl spojení držet otevřené tak dlouho, dokud to bude potřeba (spojení někdy může vypadnout, například v důsledku výpadku spojení mezi klientem a serverem, nebo v případě, že klient zavře webový prohlížeč), tohoto chování může být docíleno pomocí cyklu, který bude kontrolovat, zda nenastaly události, o kterých je zapotřebí informovat klienta.

Některé starší proxy servery ukončují HTTP spojení po chvilce, kdy není aktivní (timeout). Z tohoto důvodu je dobré na straně serveru implementovat mechanizmus odesílání komentáře. Komentář bude odeslán klientovi každých například 10 sekund, díky tomu spojení bude aktivní a nebude proxy serverem ukončeno. [6]

2.5.4. Podpora ve webových prohlížečích

Jelikož jsou Server-Sent Events poměrně novou technologií, může se stát, že ve starších webových prohlížečích nebude podporována a komunikace pomocí ní nebude možné navázat. Tabulka 2.2 zachycuje, od jaké verze jsou Server-Sent Events podporované v desktopových webových prohlížečích, tabulka 2.3 zachycuje, od jaké verze jsou podporované v mobilních webových prohlížečích.

Tabulka 2.2.: Podpora Server-Sent Events v desktopových webových prohlížečích. Zdroj: [8]

Chrome	Firefox (Gecko)	Internet Explorer (Edge)	Opera	Safari
6	6.0	Není známo[3]	9	5

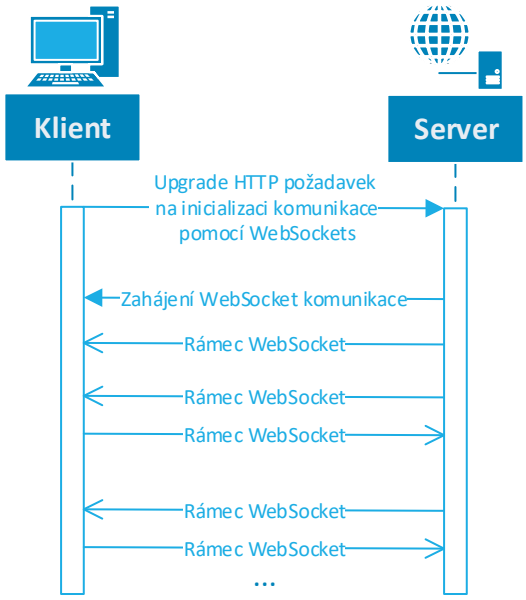
Tabulka 2.3.: Podpora Server-Sent Events v mobilních webových prohlížečích. Zdroj: [8]

Android	Firefox Mobile (Gecko)	IE Mobile	Opera Mobile	Safari Mobile
4.4	45	Není známo[3]	12	4.1

2.6. WebSockets

Nejplnohodnotnějším řešením implementace komunikace v reálném čase ve webových aplikacích je protokol WebSockets. Protokol WebSockets pomocí „upgrade HTTP požadavku“ naváže plně duplexní (obousměrné) spojení mezi klientem a serverem (všechny předchozí techniky komunikace fungovaly pouze na principu jednosměrné komunikace). Velkou výhodou tohoto spojení je, že může odesílat data ve formátu obyčejného textu (plain text) nebo ve formátu binárních dat. Protokol WebSocktes funguje na protokolu TCP, což znamená, že je zaručeno, pokud nebude přerušeno spojení mezi klientem a serverem, že všechna data, která buď klient nebo server odešlou, dorazí do svého cíle v neporušeném stavu a stejném pořadí. WebSocktes byly navrženy tak, aby fungovaly ve stávajícím internetovém prostředí (využily existující strukturu HTTP serverů, webových prohlížečů,…) z tohoto důvodu se komunikace iniciuje pomocí HTTP požadavku. Samotná komunikace pomocí WebSocktes využívá výchozího portu 80 (stejně jako HTTP) nebo 443 pro šifrovanou komunikaci (stejně jako HTTPS). [9] Obrázek 2.10 naznačuje, jak k probíhá komunikace pomocí WebSockets.

[3] V době psaní tohoto dokumentu (tj. 1. 11. 2016) žádná verze Internet Explorer (verze 11) ani Microsoft Edge (EdgeHTML 14) Server-Sent Events nepodporovaly.



Obrázek 2.10.: Diagram znázornění fungování WebSockets komunikace mezi klientem a serverem

2.6.1. Implementace na straně klienta

Na ukázce kódu 2.7 lze vidět implementaci WebSockets na straně klienta v jazyce JavaScript. Komunikace je vytvořena pomocí rozhraní „WebSocket“, jehož konstruktor očekává jako parametr URL adresu místa, ze kterého bude probíhat WebSockets komunikace. V dalším kroku je instanci rozhraní WebSocket přidán listener události „message“, tato událost nastane vždy v okamžiku, když budou přijata nějaká data, které server odešle. Poslední příkaz v ukázce volá metodu „send“, která slouží k odeslání dat na server. [6]

```
var ws = new WebSocket("ws://example.com/webSocketsSource");
ws.addEventListener("message", function (event) {
    alert(event.data);
})
ws.send("This data will be sent to server!");
```

Ukázka kódu 2.7: Implementace WebSockets komunikace na straně klienta pomocí jazyku JavaScript

Rozhraní WebSocket definuje tyto druhy událostí:

- „open“, tato událost je volána v okamžiku, kdy je vytvořeno spojení mezi klientem a serverem
- „message“, tato událost je volána v okamžiku, když klient přijme zprávu ze serveru
- „error“, tato událost je volána, když nastane nějaká chyba
- „close“, tato událost je volána, když je spojení mezi klientem a serverem přerušeno (ukončeno)

2.6.2. Implementace na straně serveru

Implementace komunikace WebSockets na straně serveru je asi nejnáročnější ze všech výše uvedených technik komunikace. Implementace se velice liší na různých platformách. Na ukázce kódu 2.8 lze vidět implementaci WebSockets serveru na platformě Node.js s využitím npm modulu „nodejs-websocket“. [10] Server v ukázce má jednoduchou funkcionalitu, když přijme zprávu od klienta, tak všechny její znaky převede na velká písmena (upper case) a poté ji upravenou odešle zpátky klientovi. Server naslouchá požadavky na portu 8001.

```
var ws = require("nodejs-websocket")

// Scream server example: "hi" -> "HI!!!"
var server = ws.createServer(function (conn) {
    console.log("New connection")
    conn.on("text", function (str) {
        console.log("Received "+str)
        conn.sendText(str.toUpperCase()+"!!!")
    })
    conn.on("close", function (code, reason) {
        console.log("Connection closed")
    })
}).listen(8001)
```

Ukázka kódu 2.8: Implementace komunikace pomocí WebSockets na straně serveru, pomocí platformy Node.js. Převzato z [10]

2.6.3. Podpora ve webových prohlížečích

WebSocktes je relativně nová technologie, která byla ale rychle implementována a všechny hlavní webové prohlížeče ji dnes již plně podporují. V tabulkách 2.4 a 2.5 jsou první verze prohlížečů, které WebSockets podporovaly (jedná se o podporu protokolu standardizovaného v RFC 6455).

Tabulka 2.4.: Podpora WebSockets v desktopových webových prohlížečích. Zdroj: [11]

Chrome	Firefox (Gecko)	Internet Explorer (Edge)	Opera	Safari
16	11.0	10	12.10	6.0

Tabulka 2.5.: Podpora WebSockets v mobilních webových prohlížečích. Zdroj: [11]

Android	Firefox Mobile (Gecko)	IE Mobile	Opera Mobile	Safari Mobile
4.4	11.0	Není známa	12.10	6.0

2.7. Srovnání jednotlivých přístupů

Pomocí všech přístupů se docílí stejného výsledku, čímž je webová aplikace komunikující se serverem v reálném čase. Tabulka 2.6 shrnuje základní údaje o jednotlivých technikách komunikace v reálném čase.

Tabulka 2.6.: Srovnání jednotlivých technik komunikace v reálném čase ve webových aplikacích

	Polling	Long Polling	Forever frame	Server-Sent Events	WebSockets
Periodické požadavky na server	ano	ne	ne	ne	ne
Uzavření spojení při odpovědi	ano	ano	ne	ne	ne
Podpora ve starších prohlížečích	ano	ano	ano	ne	ne
Transport dat pomocí HTTP	ano	ano	ano	ano	ne
Obousměrné spojení	ne	ne	ne	ne	ano

Z tabulky 2.6 vyplývá, že nejlepší metodou pro komunikaci v reálném čase jsou buď WebSockets (pokud potřebujeme, komunikovat v obou směrech), nebo Server-Sent Events (pokud stačí komunikace ze serveru na klienta). Tyto dvě techniky komunikace jsou standardizované a měly by se napříč prohlížeči chovat totožně. Tyto dvě techniky jsou nativně podporované, což znamená, že si vezmou méně zdrojů než techniky ostatní (u ostatních technik závisí hlavně na implementaci).

U techniky Forever frame může nastat zpoždění způsobené tím, že některé prohlížeče potřebují přijmout nějaké množství dat, než začnou zpracovávat požadavek v elementu „iframe“. Další nevýhodou může být to, že se příkazy posílají jako „script“ tagy, čímž se posílají kromě dat ještě kousky JavaScriptového zdrojového kódu.

Technika Long Polling má velkou nevýhodu v tom, že je nutné vytvořit spojení pro každou zprávu, kterou bude server odesílat. Jelikož se požadavek a odpověď neskládá pouze z dat, ale i z HTTP hlaviček, dochází k plýtvání přenosového pásma řídícími informacemi. Velkou výhodou na druhou stranu je to, že je tato technika komunikace na rozdíl od Server-Sent Events a WebSockets, podporována i ve starších webových prohlížečích, proto je vhodné ji použít jako záložní metodu, pokud Server-Sent Events nebo WebSockets nejsou dostupné.

Nejhorší variantou je pak Polling (Short Polling), protože vytváří spojení na rozdíl od Long Polling periodicky. Pokud nejsou dostupná data, server odpoví s prázdnou odpovědí. Kvůli tomu dochází ke zbytečnému vytěžování přenosového pásma, protože i když je odpověď prázdná, dochází k odesílání HTTP hlaviček.

3. Komunikační frameworky

Existuje velké množství různých frameworků nebo knihoven pro různé programovací jazyky, usnadňující vývoj webových aplikací komunikujících v reálném čase. Tyto frameworky se vyznačují tím, že zachovávají podporu i pro straší klienty (starší webové prohlížeče), které mají omezenou podporu moderních technik komunikace v reálném čase. Pokud klient nepodporuje nějakou z novějších technik komunikace, framework automaticky použije jinou techniku komunikace, a to vše při zachování jednoho zdrojového kódu (provede se tzv. „fallback“). Framework obvykle sám automaticky vybere komunikaci, která bude nejvhodnější v daném prostředí a pomocí této techniky naváže spojení mezi klientem a serverem. Příkladem těchto frameworků nebo knihoven může být SignalR fungující na platformě ASP.NET, nebo Socket.IO pro platformu Node.js, popřípadě Spring Framework nebo Atmosphere Framework oba na platformě Java.

3.1. SignalR

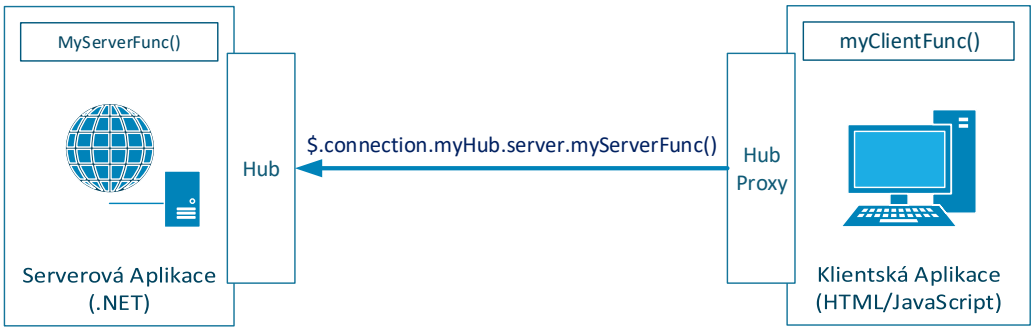
SignalR je knihovna fungující na platformě ASP.NET. Knihovna se primárně snaží navázat komunikaci pomocí moderního HTML5 protokolu WebSockets. Pokud není možné navázat komunikaci pomocí WebSockets, provede se fallback k jiné možnosti komunikace (Server-Sent Events, Forever frame nebo Long Polling). [12]

Knihovna implementuje dva rozdílné modely komunikace mezi klientem a serverem. Každý model je vhodný pro jiné typy aplikací. První model navazuje a řídí spojení mezi klientem a serverem (Persistent Connection API) a data, která se budou posílat, už ovlivňuje přímo vývojář. Druhou možností komunikace jsou tzv. Huby. Huby jsou z programátorského hlediska na vyšší úrovni než Persistennt Connection API. Pomocí Hubů je možné zavolat z klienta metodu na serveru nebo naopak zavolat ze serveru metodu u klienta (RPC[1]).

Na obrázku 3.1 je naznačeno fungování Hubů, volání metody na klientovi ze serveru a na obrázku 3.2 je naznačeno fungování v opačném směru, tj. volání metody na serveru z klienta.

[1] Remote procedure call (vzdálené volání procedury)

Obrázek 3.1.: SignalR Huby - volání metody klienta ze serveru. Zdroj: [12]



Obrázek 3.2.: SignalR Huby - volání metody na serveru z klienta. Zdroj: [12]

3.2. Socket.IO

Socket.IO je open source real-time framework určený pro platformu Node.JS (na straně serveru) a webový prohlížeč (na straně klienta) šířený pod licencí MIT. Podporuje obousměrnou komunikaci mezi klientem a serverem, tato komunikace je řízená událostmi (event-based). Socket.IO má od verze 1.0 má plnou podporu pro přenos binárních dat (je možno přenášet jakýkoliv BLOB), není díky tomu žádný problém přednášet data jako obrázky, zvuky, nebo videa. Jako jiné frameworky se Socket.IO snaží vytvořit spojení mezi serverem a klientem pomocí protokolu WebSockets, na rozdíl od ostatních ale nepoužívá fallback, ale nejprve vytvoří Polling spojení a teprve poté se pokusí provést upgrade na WebSockets. [13]

3.3. Spring Framework WebSockets

Spring Framework od verze 4 obsahuje modul „spring-websocket“, který přidává podporu pro protokol WebSockets. Funguje jako vrstva aplikace, která přidává podporu pro komunikaci v reálném čase, pokud nelze z nějakého důvodu navázat komunikaci pomocí WebSockets framework, provede fallback k jiným technikám, pomocí kterých se WebSockets simulují. Ve Spring Frameworku je fallback implementován na základě protokolu SockJS. Komunikace mezi klientem a serverem bude na straně klienta probíhat pomocí „sockjs-client“, který emuluje W3C WebSocket API. [14]

Jelikož je protokol WebSockets nízkoúrovňový, tj. pracuje velmi blízko protokolu TCP, nespecifikuje žádný přesně daný styl (formát, nebo protokol) zpráv (styl zpráv může být specifikovaný při navazování WebSockets komunikace v hlavičce zprávy), proto Spring Framework přidává podporu pro protokol STOMP[2], která definuje, jak zprávy budou vypadat.

3.4. Atmosphere Framework

Atmosphere je framework fungující na JVM (Java Virtual Machine). Framework slouží k tvorbě asynchronních webových aplikací. Tento framework je možné použít s jinými běžně užívanými frameworky. Komunikace pomocí tohoto frameworku probíhá s využitím technologií WebSockets, Server-Sent Event, Long-Polling a nebo Forever frame. [15]

3.5. Shrnutí

Existuje velké množství možností, jak udělat webovou aplikaci, ve které mezi sebou bude server a klient komunikovat v reálném čase. Jakou techniku (metodu) komunikace použít záleží na konkrétní aplikaci. Jiné nároky na komunikaci bude mít například emailový klient (který přijímá email „jen několikrát za den“) a webová hra pro více hráčů (kde se mohou odesílat desítky zpráv za vteřinu v obou směrech, tj. ze serveru na klienta a z klienta na server).

Pokud bude jednou z priorit komunikace v reálném čase o vysoké frekvenci a nízké odezvě (například přenos zvuku, videa, hraní online her…), bude vhodné využít metodu WebSockets, která je pro tyto účely navržena. Všechny prohlížeče bohužel nepodporují WebSockets (někteří uživatelé používají staré, neaktualizované prohlížeče), je nutné implementovat fallback a pokusit se nasimulovat techniku, která není podporována pomocí jiné podporované techniky (například nasimulovat WebSockets pomocí

[2]The Simple (or Streaming) Text Oriented Messaging Protocol [Zdroj: https://stomp.github.io/]

Long Polling), díky fallbacku bude zajištěná funkcionalita aplikaci i pro uživatele se starším softwarovým vybavením (v některých případech bude nutné omezit frekvenci zasílání zpráv, aby nedošlo k pádu prohlížeče).

Za předpokladu, že se bude vyvíjet aplikace, která není závislá na komunikaci s vysokou frekvencí, nebo na co nejmenším zpoždění (například novinkové servery, textový chat, emailový klient,...), je možné použít rovnou nějakou z „starších“ metod, a nemusí se řešit problém s fallbackem v případě, že klient starší metodu komunikace nepodporuje.

Nejlepší volbou při vývoji webové aplikace je ale použití komunikačního frameworku. Framework přidává vrstvu abstrakce, kdy programátor nemusí řešit, jak je spojení navázáno, ale pouze ví, že spojení bylo navázáno a je automaticky udržováno frameworkem, v případě, že není nějaká metoda komunikace dostupná, tak framework automaticky provede fallback, díky tomu se programátor nemusí zabývat implementací navázání komunikace, ale může více času věnovat tvorbě samotné aplikace.

4. Návrh komunikační knihovny

Knihovna by měla splňovat následující požadavky:

- Snadná použitelnost
- Přenos a identifikace libovolných textových a binárních dat
- Podpora přenášení multimediálních dat (zvuk a video)

4.1. Výběr serverové platformy

Existuje velké množství programovacích jazyků, pomocí kterých lze vyvíjet serverovou část webové aplikace (např. C#, Java, JavaScript, Python, PHP,...) a ještě větší množství různých frameworků, napsaných pomocí těchto jazyků.

4.1.1. ASP.NET

Platforma ASP.NET původně obsahovala pouze technologii Web Forms, která sloužila pro tvorbu webových stránek, přičemž vývoj pomocí této technologie byl podobný vývoji klasických desktopových aplikací. [16] Později Microsoft rozšířil platformu o MVC framework (možnost vytvářet webové aplikace pomocí návrhového vzoru MVC), Web API (tvorba webových služeb a webových API, které poskytují obsah pomocí XML nebo JSON) a SignalR (webové aplikace schopné komunikace v reálném čase).

Velkou výhodou ASP.NET je možnost vyvíjet aplikace v jakémkoliv programovacím jazyku, který je podporovaný platformou .NET (C#, Visual Basic,...). Další výhodou je možnost využívání veškerých možností frameworku .NET. Framework ASP.NET je šířen jako open source pod licencí Apache 2.0. [17]

V současné době je vyvíjena nová verze ASP.NET známá jako ASP.NET Core (aktuálně je vydaná verze 1.1[1]). [18] ASP.NET Core zásadně předělává framework ASP.NET, je open source mezi platformní (cross-platform). ASP.NET Core funguje na všech hlavních platformách jako jsou Windows, Linux a Mac.

[1] Verze 1.1 byla vydána v listopadu 2016

4.1.2. Java EE

Java EE (Enterprise Edition) podporuje stejné API jako Java SE (Standart Edition) a navíc k němu přidává další API jako jsou například [19]:

- Java Persistence API (JPA), rozhraní pro persistentní ukládání objektů
- Java Transaction API (JTA), rozhraní pro transakce
- Java Message Service (JMS), sada tříd a rozhraní, umožňující aplikacím komunikovat mezi sebou
- A další…

Java EE poskytuje technologie a rozhraní, které jsou určeny pro vytváření komplexních, robustních Enterprise webových aplikací.

4.1.3. Shrnutí

Jak Java EE tak i ASP.NET byly navrženy pro tvorbu velkých firemních aplikací, které jsou schopné obsloužit velké množství požadavků. Obě platformy jsou open source a multiplatformní.

Při výběru vhodné platformy budou tedy rozhodující:

- Znalost dané platformy vývojářem
- Platforma, na které jsou vyvíjené předchozí systémy
- Cena pořízení softwarového a hardwarového vybavení
- Osobní preference

4.2. Analýza protokolů (formát zpráv)

Aby mezi sebou mohli klient a server komunikovat, je nutné, aby obě strany znaly a používaly stejný formát zpráv (formát dat), které budou odesílat, nebo přijímat (tj. aby používaly stejný protokol). Existuje velké množství formátu serializace dat, které by bylo možné využít. V následujících podkapitolách budou uvedeny jen některé z nich.

4.2.1. XML

XML (Extensible Markup Language) je univerzální formát dokumentů a dat používaných v prostředí WWW (World Wide Web). [20] Formát XML byl odvozen z jazyku SGML , jedná se tedy od značkovací jazyk, podobně jako HTML. Na rozdíl od jiných

značkovacích jazyků nemá předem definovanou množinu použitelných značek (tagů). XML je textový formát, který je snadno čitelný člověkem a upravitelný v kterémkoliv textovém editoru. Nevýhodou tohoto formátu je, že pokud je zapotřebí přenášet pomocí něj binární data, je nutné je nejprve zakódovat do textového formátu např. Base64 (kódování Base64 transformuje jakékoliv binární data na sekvenci tisknutelných znaků tj. velkých a malých písmen, číslic a speciálních znaků „+“, „/“ a „=“, každé 3 bajty jsou nahrazeny 4 ASCII znaky). [21] Další nevýhodou tohoto formátu je, že obsahuje velké množství dat, která nenesou žádnou užitnou informaci (tj. data jsou obsazená až v atributu značky [tagu], nebo mezi značkami), což může zbytečně zatěžovat přenosové pásmo.

4.2.2. EXI

EXI (Efficient XML Interchange) je formát, který slouží pro kódování XML dokumentů do binárního tvaru, který je více efektivní a má menší velikost, takže snižuje využití šířky pásma, aniž by to negativně ovlivnilo ostatní zdroje počítače, jako jsou například výdrž baterie, využití paměti, využití výpočetního výkonu, atd. [22] Dle výsledků hodnocení organizace W3C je kódování a dekódování dat ve formátu EXI téměř ve všech testovaných případech rychlejší než ve formátu XML (pro nekomprimovaná data, je kódování v průměru 6x rychlejší, dekódování dat je pak v průměru 9,2x rychlejší), test byl prováděn na 94 dokumentech od 21 testovacích skupin. [23]

4.2.3. JSON

JSON (JavaScript Object Notation) je textový formát primárně určený pro výměnu (posílání) dat mezi aplikacemi napsanými v různých programovacích jazycích (tj. je nezávislý na použitém programovacím jazyce). [24] JSON je založený na standardu ECMA-262 (jednou z implementací ECMA-262 je JavaScript). Objekty se pomocí JSON zapíší pomocí složených závorek (tj. znaky „ … “), jednotlivé hodnoty jsou pak zapisovány ve stylu „jméno:hodnota“. Hodnotou může být další objekt, což umožňuje vytvářet složitější struktury. Nevýhodou tohoto formátu je, že nepodporuje binární data. Pokud je zapotřebí zapsat nějaká binární data, je nutné je zakódovat například pomocí Base64. Výhodou JSONu je, že je velice snadné s ním pracovat pomocí programovacího jazyku JavaScript.

4.2.4. BSON

BSON (Binary JSON) je binární serializace dokumentů JSON. [25] Na rozdíl od JSONu má BSON podporu pro datové typy, které v JSONu chybí, příkladem těchto typů je

datum (Date) a binární data (BinData). Oproti JSONu mají data menší velikost, čímž se snižuje využití šířky pásma.

4.2.5. MessagePack

MessagePack je podobný jako JSON s tím rozdílem, že je binární. [26] Data jsou zapsána ve formátu, kdy první bajt má význam datového typu, podle datového typu mají další bajty význam buď délky (počtu dalších bajtů patřícího tomuto datovému typu - použito například u textových řetězců string), nebo jsou to bajty nesoucí hodnotu.

4.2.6. Protocol Buffers

Protocol Buffers je mechanizmus pro serializování strukturovaných dat. [27] Navrhla ho společnost Google. Formát dat se specifikuje pomocí definičního souboru (proto), ve kterém se specifikuje formát zpráv. Jakmile je soubor připraven, Protocol Buffers vygeneruje zdrojový kód, který lze využít v aplikaci. Pomocí tohoto zdrojového kódu bude probíhat kódování a dekódování dat. Protocol Buffers neomezuje použití pouze pro přenos dat po síti, ale může také sloužit k ukládání dat do souborů, popřípadě na jiná média. Protocol Buffers využívá binárních dat a oproti XML má výhodu, že data jsou 3x-10x menší a je 20x-100x rychlejší. Protocol Buffer je implementován v různých programovacích jazycích (například Java, C#, Python,...).

4.2.7. Shrnutí

Existuje velké množství protokolů s různými vlastnostmi. Jednou z hlavních charakteristik je, zda se jedná o textový nebo binární protokol. Textové formáty se vyznačují dobrou čitelností člověkem, na druhou stranu ale mají větší nároky na šířku pásma, protože jsou všechna data kódována jako text. Další nevýhodou textových formátů je, že než bude možné s daty pracovat, je nutné je převést z textu do nějaké počítačem lépe zpracovatelné formy (např. konvertovat textové zápisy čísel na odpovídající datový typ [např. integer, float,...], atd.), je zapotřebí provést tzv. parsování.

Na druhou stranu u binárních formátů dat je nutné znát přesnou strukturu dat (tj. jak jsou data reprezentována), bez znalosti struktury dat se z dat stává bezvýznamná sekvence jedniček a nul. Pokud je ale struktura dat známá, není velký problém data zpracovat a využít (znalost struktury dat je zapotřebí i u některých textových formátů, některé textové formáty strukturu definují dobře samy o sobě [např. XML]).

Každý z formátů má své výhody a nevýhody a velice záleží na konkrétním případu, který z formátů je pro danou aplikaci nejvhodnější. V případě, že se bude jednat o aplikaci, která bude poskytovat veřejné rozhraní (tj. klienta, nebo aplikaci, která bude zpracovávat data, může napsat kdokoliv), je vhodné využít nějaký z textových formátů

(XML nebo JSON), které jsou platformě nezávislé a je pro ně velká podpora napříč různými platformami (Java, C#, JavaScript,...) a jsou v prostředí webových aplikací hojně rozšířené a známé. Pokud se ale bude jednat o aplikaci, která nebude poskytovat veřejné rozhraní, je možné zvážit použití binárních formátů (BSON, MessagePack nebo Protocol Buffers), čímž může dojít ke snížení velikosti dat.

4.3. Komprimace dat

Existuje velké množství kompresních algoritmů, jejichž podpora se liší platforma od platformy. Příkladem některých bezztrátových algoritmů můžou být:

- LZ4, je bezztrátový kompresní algoritmus dosahující rychlosti přibližně 400 MB/s na jedno jádro [28]. Rychlost dekomprese dat může dosahovat několik GB/s na jádro. Knihovna LZ4 je distribuována jako open source pod licencí BSD.
- Snappy, je komprimační/dekomprimační knihovna vytvořena společností Google [29]. Knihovna míří hlavně na rychlost komprese s rozumným komprimačním poměrem. Knihovna Snappy je distribuována jako open source pod licencí BSD.
- ZStandard, je real-time kompresní algoritmus, který dosahuje vysokých kompresních poměrů [30]. Knihovna ZStandard je distribuována jako open source pod licencí BSD.

4.3.1. Srovnání

V tabulce 4.1 lze vidět porovnání jednotlivých kompresních knihoven, jejich kompresní poměr, rychlost komprese a dekomprese.

Tabulka 4.1.: Srovnání výkonu jednotlivých komprimačních knihoven. Převzato z [30]

Kompresní knihovna	Kompresní poměr	Rychlost komprese	Rychlost dekomprese
zstd 1.1.3-1	2.877	430 MB/s	1110 MB/s
lz4	2.101	720 MB/s	3600 MB/s
snappy 1.1.3	2.091	500 MB/s	1650 MB/s

Největším omezením pro použití těchto algoritmů je jejich podpora ve webových prohlížečích. V současné době[2] existuje JavaScriptová implementace spustitelná ve webovém prohlížeči pouze pro algoritmus LZ4.

[2] V době psaní tohoto odstavce, tj. 15. 3. 2017

4.3.2. Komprese multimediálních dat

Komprese multimediálních dat může být dvojího typu, ztrátová nebo bezztrátová. Příkladem některých kompresí může být:

- JPEG, kompresní metoda obrázků, může být ztrátová, nebo bezztrátová [31]. Nejlépe funguje na obrázky, ve kterých mají sousední pixely podobnou barvu.
- PNG, formát souboru pro uchování bezztrátově komprimovaného obrázku [32].
- VP8/VP9, kodeky, podporující ztrátovou kompresy videa [33].
- OPUS kodek, podporující realtimovou ztrátovou kompresy audia [34].

Velkou výhodou využití komprese multimediálních dat je její nativní podpora v nových webových prohlížečích.

4.4. Návrh a implementace

Komunikační knihovna, pojmenována „Hermes“, je navržena a implementována s použitím následujících technologií:

- Serverová strana je napsána v programovacím jazyce C# s využitím rozhraní OWIN[3].
- Pro komunikaci mezi serverem a klienty je použita komunikační knihovna SignalR.
- Klientská strana je napsána v programovacím jazyce TypeScript a poté kompilována do JavaScriptu spustitelného ve webovém prohlížeči.
- Data přenášená mezi klientem a serverem jsou serializována do binární formy pomocí Protocol Buffers.

4.4.1. Formát zpráv (Protocol Buffers)

Knihovna využívá celkem 3 druhy objektů, použitých knihovnou Protocol Buffers. Tyto objekty jsou definovány v souboru „message.proto“ (viz příloha zdrojový kód, projekt Hermes). Jedním objektem je výčtový datový typ (enum) „DataType“, který identifikuje, o jaký typ dat se jedná (textová data, binární data, audio data, video data nebo řídící data použitá pro správný běh knihovny).

[3]Open Web Interface for .NET (OWIN) je otevřená specifikace, která definuje standartní rozhraní pro komunikaci serveru s webovou aplikací, které pomocí abstrakce umožňuje tyto dvě komponenty, které jsou historicky velmi spjaté, dekomponovat [35]

Druhým objektem je zpráva „Message“, která je použitá pro zapouzdření dat, která boudou přenášena mezi klientem a serverem. Zpráva obsahuje následující pole:

- „id“, celočíselné nezáporné identifikační číslo (unsigned integer), identifikující proud dat.
- „origin“, textový řetězec, který obsahuje ID klienta, který inicioval tuto zprávu.
- „mimeType“, textový řetězec, který obsahuje identifikaci typu dat, která jsou obsažená ve zprávě.
- „dataType“, hodnota výčtového datového typu „DataType“, obsahuje identifikaci typu dat obsažených ve zprávě.
- „data“, pole bajtů, které nesou hodnotu zprávy.

Třetím objektem je zpráva „MessageCollection“, která je použitá pro zapouzdření více zpráv do jedné zprávy tak, aby bylo možné přenášet více zpráv během jednoho požadavku. Tato zpráva je serializována a odesílaná mezi klientem a serverem.

4.4.2. Typy dat

Knihovna rozeznává 5 základních typů dat:

- Text, datový typ obsahující textové řetězce.
- Binary, datový typ obsahující binární data (pole bajtů).
- Audio, datový typ obsahující binární data (pole bajtů), které lze dekódovat jako audio, více informací o použitém kontejneru a kodeku, musí být vyčteno z MIME type zprávy nebo z hlavičky dat (tj. z prvních přijatých/odeslaných dat)
- Video, datový typ obsahující binární data (pole bajtů), které lze dekódovat jako video, stejně jako u audia je zapotřebí dodatečné informace vyčíst z MIME type zprávy nebo hlavičky dat
- Control, poslední datový typ obsahující textová data, v těchto textových datech je datový objekt ve formátu JSON, který slouží pro předávání různých informací mezi klientem a serverem (například informaci o tom, že se někdo připojil a změnil se status komunikace,...).

4.4.3. Popis serveru

Část knihovny Hermes pro serverovou stranu obsahuje dvě hlavní komponenty. První komponentou je rozhraní „ICommunication“, které slouží k uchovávání informací o relaci komunikace. Třídy implementující toto rozhraní mají za úkol poskytnout potřebné informace o klientech, kteří se účastní aktuální komunikace, aktuální status komunikace, omezení, jaká data mohou klienti odesílat atd.

Druhou komponentou je abstraktní třída „BaseConnection“, která je potomkem SignalR třídy „PersistentConnection“. Tato třída poskytuje metody pro odesílání dat klientům a zároveň implementuje výchozí chování při přijetí dat. Výchozí chování může být předefinováno, pokud je to potřebné, kvůli dosažení požadovaného chování.

4.4.3.1. Komunikace (Communication)

Knihovna Hermes implementuje základní chování rozhraní „ICommunication“ pomocí abstraktní třídy „BaseCommunication“, která implementuje výchozí chování tříd komunikace, které může být v odvozených třídách předefinováno. Knihovna dále poskytuje tři základní potomky třídy „BaseCommunication“, každý určen pro trochu jiný use case.

První implementací je třída „OneToOneCommunication“, která slouží pro komunikaci mezi pouze dvěma klienty. Jeden klient odesílá data a druhý klient je jediný, kdo je přijímá. Pokud dojde k odpojení jednoho ze dvou klientů, komunikace končí a je odpojen i druhý klient. Příklad použití této komunikace může být při implementaci nějaké webové aplikace fungující jako instant messaging.

Druhou implementací je třída „OneToManyCommunication“, která jak název napovídá, slouží pro přenos dat od jednoho klienta k mnoha dalším klientům. První připojený klient je označen jako master a ostatní připojení klienti jsou označeni jako slave. Master klient je hlavním klientem, který odesílá data ostatním, pokud dojde k odpojení hlavního klienta, komunikace končí a ostatní klienti jsou odpojeni také. Výchozí nastavení omezení této třídy povoluje, aby data odesílal pouze master klient, ostatní slave klienti mají veškeré odesílání dat zakázáno. Příklad použití této komunikace může být „internetové rádio“, kdy master klient slouží jako vysílač a ostatní klienti slouží jako posluchači.

Třetí implementací je třída „ManyToManyCommunication“, která slouží pro komunikaci mnoha klientů s mnoha klienty. Pokud dojde k odpojení nějakého klienta, nedochází k odpojení ostatních klientů. Komunikace zaniká až ve chvíli, kdy se odpojí poslední klient. Pokud v komunikaci je v aktuální okamžik pouze jeden klient, je status komunikace roven hodnotě „čekající“ (waiting), pokud je v jeden okamžik v komunikaci více než jeden klient, je status komunikace roven hodnotě „připravený“ (ready).

Příklad použití této komunikace může být při implementaci nějaké online hry pro více hráčů.

Další komunikace mohou být vytvořeny implementací rozhraní „ICommunication“, nebo implementací abstraktní třídy „BaseCommunication“, popřípadě děděním od výše uvedených základních tříd komunikace.

4.4.3.2. Spojení (Connection)

Základním prvkem knihovny Hermes je generická třída „DefaultConnection“, která je potomkem abstraktní třídy „BaseConnection“. Tato třída přijímá jako generický parametr implementaci rozhraní „ICommunication“, který říká, jaký typ komunikace bude spojení reprezentovat. „DefaultConnection“ se ve výchozím stavu chová tak, že při přijetí dat zkontroluje, zda je povolené, aby daný klient odeslal daný typ dat, a pokud je vše v pořádku, tak data přepošle všem ostatním klientům, kteří se účastní dané komunikace.

Nové spojení může být vytvořeno pomocí implementace abstraktní třídy „BaseConnection“ nebo děděním od třídy „DefaultConnection“.

Pokud je zapotřebí upravit chování při přijetí dat, je zapotřebí redefinovat požadované metody začínající slovem „Received“ (ReceivedText, ReceivedBinary,…) tyto metody jsou volány na serveru při přijetí dat, na základě proměnné „DataType“ obsažené ve zprávě.

Pro odesílání dat ze serveru slouží metody zažínající slovem „Send“ (SendText, SendBinary,…) nebo „Broadcast“ (BroadcastText, BroadcastBinary,…). Metody „send“ slouží pro odeslání dat přesně specifikovaným klientům. Metody „broadcast“ slouží pro odeslání dat všem klientům, kteří se účastní dané komunikace. Více informací o těchto metodách lze najít v dokumentaci zdrojového kódu, nebo přímo ve zdrojovém kódu.

4.4.3.3. Ukázka použití knihovny

Pro použití knihovny na serveru stačí přidat knihovnu do projektu. Jakmile je přidána k projektu, je zapotřebí registrovat adresy URL, pomocí kterých se bude přistupovat na server během komunikace v spouštěcí (startup) OWIN třídě. Ukázku použití lze vidět na ukázce kódu 4.1. Kód použitý v ukázce kódu 4.1 bude očekávat připojení na adrese „/hermes“. Pokud je výchozí chování knihovny vyhovující, není již potřeba na serveru provádět další změny a komunikace bude funkční.

```
using Hermes.Communication;
using Hermes.Configuration;
using Hermes.Connection;
using Owin;

namespace HermesSample
{
  public class Startup
  {
    public void Configuration(IAppBuilder app)
    {
      var config = new Configuration();
      config.MapConnection<DefaultConnection<OneToOneCommunication>>("/hermes");
      config.Config(app);
    }
  }
}
```

Ukázka kódu 4.1: Spouštěcí konfigurace aplikace, která používá komunikační knihovnu Hermes.

4.4.4. Popis klienta

Klientská část knihovny HermesJS obsahuje jednu hlavní třídu „HermesConnection“, která slouží pro navázání komunikace se serverem a následné odesílání a přijímání dat. Pro odesílání dat slouží metody začínající slovem „send“ (sendText, sendBinary,...), více informací o metodách lze najít v dokumentaci zdrojového kódu nebo ve zdrojovém kódu. Pro správnou funkcionalitu knihovny je nutné připojit do HTML dokumentu před samotnou knihovnou také SignalR klienta, jehož funkcionalitu knihovna přímo využívá a knihovnu jQuery, kterou vyžaduje SignalR klient.

4.4.4.1. Kompilace klientské knihovny

Knihovna je napsaná v programovacím jazyce TypeScript (nadstavba JavaScriptu) a je pro větší přehlednost kódu rozdělena do více souborů. Samostatné soubory jsou kompilovány do JavaScriptu automaticky pomocí IDE (záleží na nastavení IDE, Visual Studio a WebStorm jsou schopny automaticky kompilovat při uložení souboru). Výstupem jsou jednotlivé JavaScriptové soubory, které pořád nejsou připravené pro použití ve webovém prohlížeči. V případě, že by byly soubory do HTML dokumentu vloženy ve správném pořadí, tak by vše fungovalo, ale došlo by k velkému množství požadavků na webový server, což by způsobilo zbytečné vytížení serveru a zpoždění načítání celého dokumentu. Z tohoto důvodu knihovna využívá rozeznávání závislostí podle Google Closure Library.

4.4.4.1.1. Google Closure Library

V Google Closure Libraray se používají funkce „goog.provide“ a „goog.require“ pro definování závislostí na dalších komponentách (souborech, modulech) [36]. Provide se používá pro definování, že daný soubor obsahuje nějakou definici. Require se používá pro definování toho, že soubor vyžaduje definice, které jsou definovány v jiném souboru. Pomocí provide a require lze pak sestavit strom závislostí, podle kterého lze zjistit správné pořadí, ve kterém mají být soubory přidávány do stránky. K vytvoření stromu je v Google Closure Library připraven pythonovský skript, který tento strom vytvoří a vrátí seřazený seznam souborů v pořadí, které odpovídá pořadí, v jakém mají být přidány do stránky.

Tento seznam lze použít s dalším nástrojem Closure Compiler, který je pak schopen tyto soubory zkompilovat do jednoho jediného JavaScriptového souboru, který obsahuje veškerý kód.

Aby byla kompilace co možná nejplynulejší, je ke knihovně přidán pythonovský script „build.py“, který má za úkol zkompilovat jednotlivé soubory do jednoho velkého s využitím předchozího postupu.

4.4.4.1.2. Závislosti

Aby mohla kompilace úspěšně proběhnout, musí být přítomny všechny požadované soubory. Pro automatické stažení závislostí je ke knihovně přidán pythonovský skript „download_dependency.py“, který slouží pro automatické stažení požadovaných knihoven.

První požadovanou knihovnou je Protocol Buffers, která je nutná, aby knihovna mohla správně serializovat odesílaná data a deserializovat přijímaná data. Tato knihovna pro svou funkcionalitu vyžaduje Google Closure Libarary. Druhou požadovanou knihovnou tedy je Google Closure Library. Tato knihovna je využitá kvůli knihovně Protocol Buffers a také kvůli rozeznávání závislostí.

Více informací o závislostech, a zprovoznění kompilování kódu lze najit v souboru „Hermes/HermesJS/SOURCE_README.md“, který je součástí zdrojového kódu knihovny.

4.4.4.2. Události knihovny

Knihovna HermesJS definuje několik druhů událostí, které mohou nastat a být zpracovány klientským kódem. Přidání obsluhy události se provádí pomocí funkce „addEventListener“, kde první parametr je název události a druhý parametr je funkce, která bude zavolána v okamžik, když nastane událost (callback). V tabulce 4.2 lze vidět shrnutí všech podporovaných událostí.

Tabulka 4.2.: Názvy a popis událostí, podporovaných objektem typu HermesConnection

Název události	Popis	Typ objektu v callbackové funkci
connect	Událost, která je vyvolána v okamžik, když je spojení mezi klientem a serverem vytvořeno	HermesEvent
disconnect	Událost, která je volána v okamžik, když je spojení mezi klientem a serverem odpojeno	HermesEvent
full	Tato událost je zavolána po události „connect“ za předpokladu, že daná komunikace je již plná a klient se nemůže zúčastnit komunikace	HermesEvent
ready	Tato událost je zavolána po události „connect“ za předpokladů, že daná komunikace je připravená odesílat a přijímat data	HermesEvent
received	Událost, která je vyvolána, když jsou přijatá data ze serveru	ReceivedEvent
waiting	Tato událost je zavolána po události „connect“ za předpokladů, že daná komunikace ještě čeká na další klienty, než bude moci odesílat a přijímat data	HermesEvent

Pro přesnější přijímání dat lze požít metodu „addReceivedEventListener“, kde první parametr je číslo datového typu (lze využít objekt HermesJS.DataType, pro specifikování požadovaného datového typu v lépe čitelné podobě), druhý parametr je callbacková funkce a třetí nepovinný parametr je číslo ID, které identifikuje zprávu, když je tento parametr vynechán bude callback volán pro všechny ID, pokud není vynechán, bude volán, pouze pokud id zprávy bude rovno hodnotě tohoto parametru.

4.4.4.3. Ukázka použití knihovny

Na ukázce kódu 4.2 lze vidět ukázku použití knihovny. V ukázce se připojují požadované skripty, poté je vytvořen objekt „HermesConnection“. Pomocí metody start se vytvoří spojení mezi klientem a serverem, poslední dva příkazy demonstrují přidávání obsluhy událostí.

```
<script src="~/Scripts/jquery-1.10.2.min.js"></script>
<script src="~/Scripts/jquery.signalR-2.2.1.min.js"></script>
<script src="~/Scripts/HermesJS.js"></script>
<script type="text/javascript">
  var connection = new HermesJS.Connection.HermesConnection("/hermes");
  //TODO Set required properties of connection (sessionId, logging,...)
  connection.start();
  connection.addEventListener("ready", function () {console.log("READY")});
  connection.addReceivedEventListener(HermesJS.DataType.TEXT, function(event) {
    //TODO listener code, for text with id 132
  }, 132);
</script>
```

Ukázka kódu 4.2: Ukázka použití knihovny HermesJS v HTML stránce

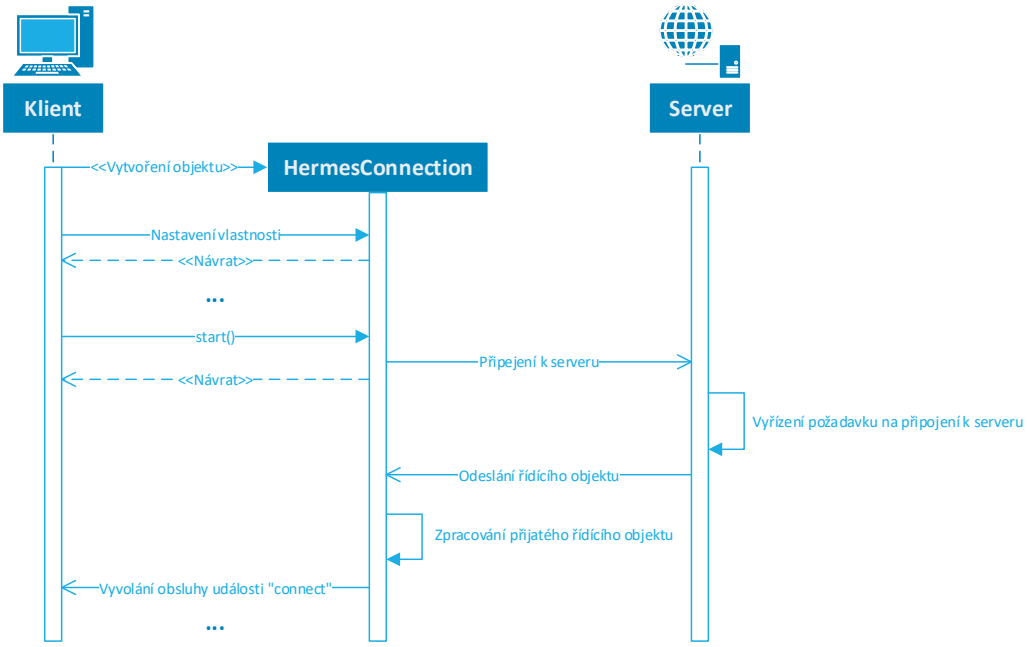
4.4.5. Postup připojení na server

Aby mohla nastat komunikace mezi klientem a serverem (a následně mezi klienty), je nutné, aby se klient připojil na server. Postup připojení k serveru je následující. Na klientské straně (ve webovém prohlížeči) se vytvoří objekt typu „HermesConnection“, tato třída ve svém konstruktoru přijímá parametr URL, které slouží jako koncový bod, na který se spojení naváže (tato URL musí být v konfiguraci serveru na mapována na nějakou implementaci abstraktní třídy „BaseConnection“, nebo může být použita generická třída „DefaultConnection“, která obsahuje základní implementaci abstraktních metod třídy „BaseConnection“.

Jakmile je na klientovi vytvořen objekt „HermesConnection“, je možné nastavit jeho vlastnosti tak, aby co nejlépe odpovídaly požadovanému chování (například zapnout logování do konsole,...). Jakmile je vše nastaveno, je zapotřebí vytvořit spojení mezi klientem a serverem. Toho se docílí zavoláním metody start, která zajistí vytvoření spojení. Jakmile je spojení vytvořeno, server odešle klientovi řídící objekt „ControlObject“, který obsahuje informace o připojení (status připojení, ID komunikace), přijetím tohoto objektu dojde u klienta k vyvolání události „connect“ a na základě vlastnosti status tohoto objektu nastane jedna z dalších událostí „ready“, „waiting“, „full“, nebo „disconnect“. Postup připojení na server lze vidět na obrázku 4.1.

4.4.6. Odesílání dat na server

Klient si při navázání komunikace nastaví frekvenci. Tato frekvence říká, jak často mají být data odeslána na server. Vždy když klient chce odeslat data na server, je na základě dat vytvořena Protocol Buffers „Message“ (viz kapitola 4.4.1), která je uložena lokálně v bufferu do chvíle, než časovač nastavený na požadovanou frekvenci vyvolá událost. V rámci obsluhy této události je vytvořena zpráva „MessageCollection“, která je postupně naplněna všemi zprávami, které byly doposud uloženy do bufferu (pozn. pokud je velikost dat zprávy rovna 0 bajtů, tak je zpráva zahozena a nebude odeslána



Obrázek 4.1.: Sekvenční diagram znázorňující postup připojení klienta na server pomocí knihovny Hermes

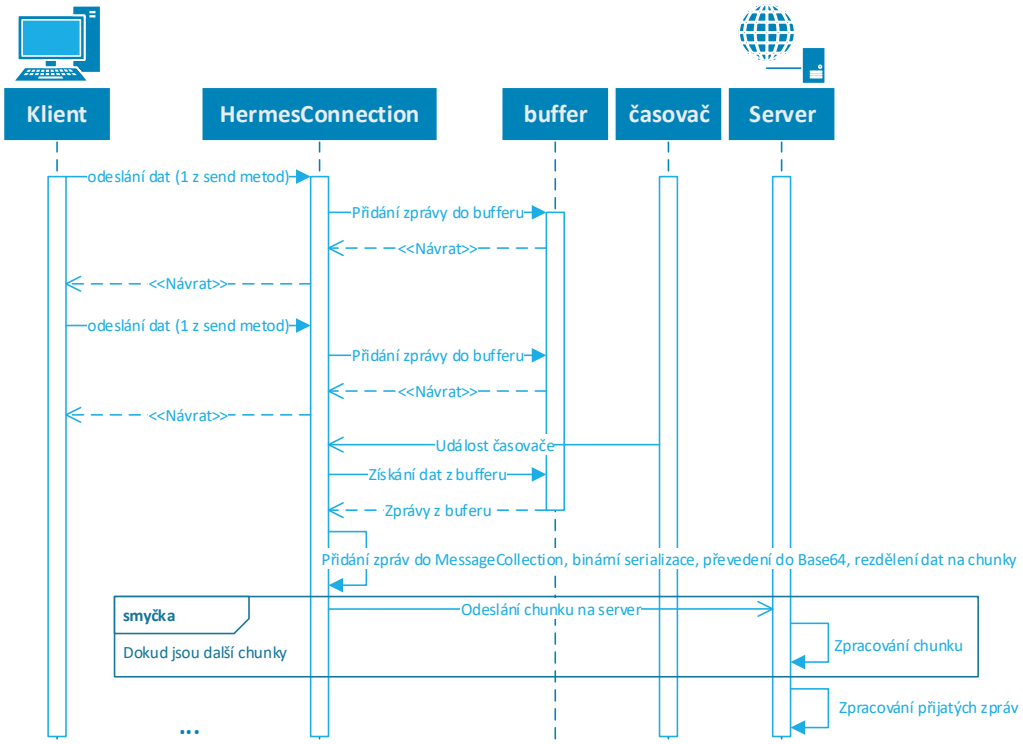
na server). Jakmile jsou všechny zprávy přidané do zprávy „MessageCollection“, provede se serializace této kolekce do binárního tvaru.

Jelikož SignalR podporuje pouze textové zprávy [37], je zapotřebí binární data zakódovat do Base64 formátu. Dalším omezením SignalR je maximální velikost zprávy, kterou klient může odeslat na server. Tato velikost je ve výchozím nastavení omezena na 64kB [38], z tohoto důvod je nutné zprávy, které mají větší velikost, rozdělit na menší části (chunky) a postupně je odeslat na server. Aby bylo zřejmé, že se jedná o chunk a ne celou zprávu je nutné, tuto zprávu nějak označit. Toho je docíleno tak, že na začátek všech chunků je přidán znak vykřičníku „!“, který není použit jako vyhrazený znak Base 64 kódování [21]. Aby bylo možné rozeznat konec zprávy, je na konec posledního chunku přidán také znak vykřičníku „!“.

Odesílání dat je znázorněno na obrázku 4.2, na kterém lze vidět sekvenční diagram, který popisuje kroky odesílání dat na server.

4.4.7. Podpora multimediálního obsahu

Knihovna Hermes má v sobě zabudovanou podporu pro přenos multimediálních dat (audio a video), které získává z uživatelova mikrofonu a webkamery. Pro jednoduché



Obrázek 4.2.: Sekvenční diagram popisující odesílání dat z klienta na sever

odesílání a příjem dat je v knihovně přítomna třída „MediaHandler“, která má za úkol sdílet uživatelova multimediální data a přijímat multimediální data sdílená jinými uživateli.

Třída „MediaHandler“ může vyvolávat události, které shrnuje tabulka 4.3, obsluha událostí může být přidána pomocí metody „addEventListener“, kde první parametr je název události a druhý parametr je callbacková funkce.

Na ukázce kódu 4.3 lze vidět jednoduchý příklad práce s MediaHandlerem. Ukázkový kód vytvoří objekt typu MediaHandler, kterému následně přidá obsluhu události „newstream“, která vezme nově přijatý media element, kterému nastaví vlastnost „autoplay“ na true, čímž se zajistí, že přehrávání dat začne automaticky samo, druhou věc, kterou obsluha události udělá, je, že přidá media element do webové stránky do elementu s ID „media“. Poslední věcí, kterou ukázkový kód dělá je, že volá metodu „shareUserMedia“, která slouží ke sdílení uživatelových medií, v tomto případě bude sdíleno pouze video (tj. vstup z webkamery).

Tabulka 4.3.: Názvy a popis událostí, podporovaných objektem typu MediaHandler

Název události	Popis	Typ objektu v callbackové funkci
endstream	Událost, která je vyvolána ve chvíli, když stream se specifikovaným ID skončí (přestane vysílat)	MediaEvent
newstream	Událost, která je vyvolána ve chvíli, když klient přijme data, která obsahují ID, která ještě nebylo zpracováno.	NewStreamEvent
mediashared	Událost, vyvolána ve chvíli, kdy klient začne sdílet data.	MediaSharedEvent

```
<div id="media"></div>
<script type="text/javascript">
    var media = new HermesJS.Media.MediaHandler(connection);
    media.addEventListener("newstream", function(event) {
        event.mediaElement.autoplay = true;
        $("#media").append(event.mediaElement);
    });
    media.shareUserMedia({audio: false, video: true});
</script>
```

Ukázka kódu 4.3: Ukázka použití třídy MediaHandler z knihovny HermesJS

Podpora přenosu multimediálních dat je postavena pouze na API poskytovaném novými webovými prohlížeči (tj. naprogramováno jen za pomoci JavaScriptu, není zapotřebí používat žádný plugin jako je například flash). Knihovna používá tyto moderní API:

- MediaDevices
- MediaRecorder
- MediaSource

4.4.7.1. MediaDevices

MediaDevices API slouží pro přístup k vstupním zařízením klienta, jako jsou mikrofony a webkamery [39]. Pomocí metody „getUserMedia“ se získá objekt typu MediaStream, který reprezentuje požadovaná data z vstupních zařízení klienta [40].

4.4.7.2. MediaRecorder

MediaRecoreder API slouží ke snadnému nahrávání MediaStreamů [41]. Při vytváření objektu typu MediaRecoreder je v konstruktoru předán MediaStream, druhý nepo-

vinný parametr je objekt, který obsahuje vlastnosti, které ovlivňují proces nahrávání (např. formát, ve kterém bude nahráváno). Nahrávání je iniciováno pomocí metody „start“. Data jsou pak dostupná pomocí obsluhy události „dataavailable“.

MediaRecorder API je dostupné pouze[4] v prohlížečích Firefox (testováno ve verzi 51), Chrome (testováno ve verzi 56) a Opera (testováno ve verzi 43). Aktuální implementace podporují pouze WebM kontejner (z tohoto důvodu knihovna HermesJS pracuje pouze s tímto formátem). Všechny tři implementace podporují video kodek VP8. Video kodek VP9 podporují pouze Chrome a Opera.

4.4.7.3. MediaSource

Objekt MediaSoruce reprezentuje zdroj dat pro HTML media element (audio nebo video) [42]. Objekt obsahuje objekty SourceBuffer, která slouží k přidávání dat, které mají být přehrána. Pomocí metody „appendBuffer“ se mohou do SourceBufferu přidávat další data, která poté boudou za běhu dekódována a přehrána.

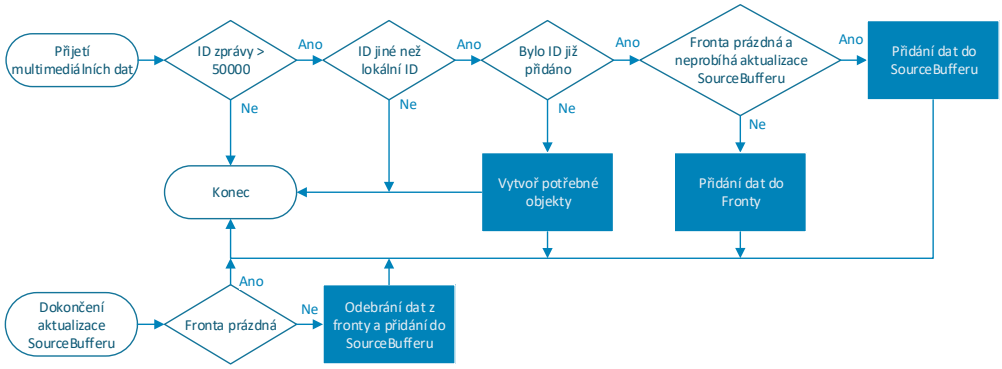
4.4.7.4. Popis multimediální funkcionality knihovny HermesJS

V prvním kroku odesílání multimediálních dat si knihovna HermesJS zažádá server o unikátní ID, které bude použito pro identifikaci multimediálních dat, která budou odesílána. Unikátní ID je generováno jako číslo větší než 50000. K žádosti o ID se využije metoda requestStreamId objektu HermesConnection.

V dalším kroku je zapotřebí získat přístup k hardwarovému zařízení, toho se docílí pomocí MediaDevices API a metody getUserMedia, která vrátí objekt MediaStream. Tento objekt se následně použije jako zdroj dat pro objekt MediaRecorder. Jakmile je nahrávání spuštěno pomocí metody „start“, začnou se generovat události „dataavailable“. V obsluze této události lze přistoupit k datům, která jsou reprezentována jako Blob. Aby bylo možné získat data jako pole bajtů, je zapotřebí vytvořit instanci třídy FileReader, na které se následně zavolá metoda „readAsArrayBuffer“, pomocí které se Blob přečte do objektu ArrayBuffer. Z ArrayBufferu se následně vytvoří Uint8Array (pole 8 bitových unsigned integerů, tj. pole bajtů). Jakmile je k dispozici Uint8Array, je možné využít metodu sendVideo nebo sendAudio objektu HermesConnection, čímž dojde k odeslání dat. O všechny tyto kroky se knihovna postará sama zavoláním metody „shareUserMedia“ třídy „MediaHandler“.

Když klient přijme audio nebo video data, jsou data zachycena v objektu „MediaHandler“. Tento objekt zkontroluje, zda ID je větší než 50000 (tj. jedná se o generované ID) a zda lokální ID streamu dat není stejné s přijatým (nechceme, aby data, která byla tímto klientem odeslána v něm i zpracována), pokud není, je zkontrolováno, jestli již

[4] Dostupnost MediaRecoreder API v době psaní odstavce tohoto dokumentu tj. březen 2017



Obrázek 4.3.: Vývojový diagram popisující postup přidávávání dat do SourceBufferu při přijetí dat

nebyla data s daným ID přijata v minulosti. Pokud data s daným ID již byla přijata, budou přidána do SourceBufferu. Než je možné data přidat do SourceBufferu, je zapotřebí zkontrolovat, zda již předchozí přidání dat bylo dokončeno. Kontrolu, zda aktualizace SourceBufferu je stále prováděna, lze provést pomocí vlastnosti „updating“, která je součástí SourceBuffer objektu. Pokud aktualizace stále provádí, jsou data přidána do fronty (FIFO) a přidána do SourceBufferu až ve chvíli, kdy je předchozí aktualizace dokončena. Na obrázku 4.3 je znázorněný vývojový diagram, popisující výše popsaný proces přijímání dat a jejich následné přidávání do SourceBufferu.

Pokud byla přijata data s ID, které ještě nebylo přijato, jsou vytvořeny všechny potřebné objekty (MediaSource, SourceBuffer, video nebo audio element). Jakmile jsou tyto objekty vytvořeny a připraveny, je vyvolána událost „newstream“.

4.4.7.5. Připojení klienta k běžícímu streamu

Pokud se nějaký klient připojí ve chvíli, kdy jiný již vysílá multimediální data, nastává problém, protože při pokusu připojit přijatá data do SourceBufferu vyvolá chybu, protože první připojená data musí být takzvaný inicializační segment [43], který obsahuje hlavičku WebM souboru, ve které jsou informace o následujících datech (informace o jednotlivých stopách, kodeky, rozlišení,...). Z tohoto důvodu knihovna u přijatých dat detekuje, zda se jedná o začátek dat nebo ne.

Formát kontejneru WebM vychází z formátu EBML používaného kontejnerem matroska [44], ze kterého vychází (je stejný jako matroska, až na to, že některé prvky, které matroska obsahuje, jsou ze specifikace WebM vynechány).

Detekce začátku WebM je tedy snadná dle specifikace EBML, každý soubor musí

začínat stejnou sekvencí 4 bajtů, jejichž hexadecimální hodnota je 1A 45 DF A3 [45]. Pro detekci, zda se jedná o hlavičku, tedy stačí zkontrolovat první 4 přijaté bajty, zda odpovídají dané hodnotě.

V případě, že neodpovídají dané hodnotě, byl do knihovny implementován mechanizmus požadavku na hlavičku. Pokud klient neobdrží data s hlavičkou, vyšle požadavek na server, který mu obratem vrátí hlavičku. Server na začátku komunikace odchytí dostatečné množství dat, ze kterého poté extrahuje tuto hlavičku (na serverové straně je pro tento účel vytvořená třída WebMHelper).

Jakmile je hlavička přijata, jsou vytvořeny všechny požadované objekty a hlavička je přidána do SourceBufferu. Za daty hlavičky musí vždy následovat EMBL element Cluster. Z tohoto důvodu knihovna čeká, dokud nepřijme data, která začínají elementem cluster, a teprve pak začne přidávat data do SourceBufferu.

4.4.8. Podporované softwarové vybavení

Klientská strana knihovny nebude fungovat ve starších prohlížečích, které nemají implementované nové API, které tato knihovna využívá. Knihovna je otestována a funkční v následujících webových prohlížečích:

- Firefox verze 51
- Google Chrome verze 56
- Opera verze 43
- Microsoft Edge verze EdgeHTML 14 (Prohlížeč Edge má pouze částečnou podporu, aby fungoval, tak potřebuje polyfill pro Encoder API [TextEncoder, TextDecoder], aby fungovalo přijímání videa, musí se v prohlížeči v „about:flags“ povolit video s kodekem VP9 [kodek VP8 není v prohlížeči podporován]. Jelikož Edge nepodporuje MediaRecorder, je odesílání videa nahrazeno odesíláním obrázků)

Serverová strana knihovny je otestována a funkční ve Windows 10 64 bit (1607 Anniversary Update) a webovém serveru IIS Express 10.

4.4.9. Struktura projektu

Celý projekt Hermes je rozdělen na tři menší projekty:

- Hermes, implementace serverové strany komunikační knihovny
- HermesJS, implementace klientské strany komunikační knihovny pro webové prohlížeče

- HermesSample, ASP.NET MVC aplikace, která využívá předchozích projektů, aby demonstrovala funkcionalitu komunikační knihovny.

4.4.9.1. Licence

Projekty Hermes a HermesJS jsou licencovány pod licencí Apache Licence 2.0, která umožňuje projekt využít v dalších projektech, a to i komerčních, aniž by vyžadovala, aby tyto projekty byly šířeny pod touto licencí. [46]

Projekt HermesSample je licencován pod licencí Unlicense, která umožňuje komukoliv s projektem dělat, co ho napadne bez jakýchkoliv omezení. [47]

4.4.10. Shrnutí

Výsledkem návrhu a implementace je komunikační knihovna, použitelná v prostředí webových aplikací. Knihovna podporuje přenášení libovolných textových nebo binárních dat. V knihovně je rovněž implementována podpora pro přenášení multimediálních dat.

Podpora pro přenos multimediálních dat je napsána v čistém JavaScriptu, není tedy zapotřebí využívat žádný modul plugin. Rozhraní, která byla použita pro dosažení požadovaného chování, jsou velice nová, některá dokonce označena jako experimentální, nebo ve fázi návrhu, což znamená, že podpora je zajištěna pouze v nejnovějších webových prohlížečích a jejich definice může být v budoucnu upravena. Z tohoto důvodu nasazení v produkčním prostředí nemůže být prozatím doporučeno. Technologie jsou to na druhou stranu velice zajímavé a jistě budou mít veliké využití v budoucnu, až se více rozšíří jejich podpora.

Knihovna je licencována pod licencí Apache Licence 2.0, což znamená, že se kdokoliv může podílet na vývoji, popřípadě vytvořit vlastní odnož vývoje této knihovny a upravit ji k obrazu svému.

5. Závěr

Existuje více možností, jak udělat z obyčejné webové aplikace webovou aplikaci komunikující v reálném čase. Některé techniky jsou méně vhodné, jiné více. Pokud bude někdo vytvářet aplikaci komunikující v reálném čase s nárokem na velké objemy přenesených dat mezi klientem a serverem s co možná nejmenší odezvou, například přenos multimediálních dat, hraní online her pro více hráčů atd., bude nejvhodnější použít WebSockets, protože jsou navrženy přesně pro tyto účely, podporují odesílání libovolně velkých dat a umožňují odesílání dat v jakémkoliv formátu (textovém nebo binárním), význam dat je pak řešen až na aplikační úrovni. Jelikož ale ne všechny webové prohlížeče podporují WebSockets, je zapotřebí v produkčním prostředí připravit podporu i pro uživatele se starším softwarovým vybavením, v tomto případě se WebSockets pokusí nasimulovat pomocí jiné ze zmíněných technik komunikace, například Long polling.

Za předpokladů, že se bude vyvíjet aplikace, která není závislá na komunikaci s co možná nejmenší odezvou, nebo nebude přenášet velké objemy dat, například novinkové servery, textový chat, je možné použít rovnou nějakou ze „starších“ technik, a nemusí se řešit problém s fallbackem v případě, že novější technika není k dispozici.

Další možnosti, jak implementovat webovou aplikaci komunikující v reálném čase, je použití nějakého komunikačního frameworku nebo knihovny. Vývojář si může vybrat framework nebo knihovnu, která bude nejvíce odpovídat jeho potřebám a velice si tak usnadnit práci, jelikož knihovny vytvoří spojení mezi klientem a serverem samy podle dostupných zdrojů. Komunikace mezi klientem a serverem je pak abstrahována pouze na metody „odeslat“ a „přijmout“. Jaká technika komunikace byla použita, je už na knihovně samotné (nebo na její konfiguraci).

Vývojář také může použít knihovnu Hermes, implementovanou v této bakalářské práci. Tato knihovna je postavena na komunikační knihovně SignalR. Výhodou využití této knihovny je přidání podpory pro odesílání libovolných binárních dat mezi klientem a serverem. Další výhodou může být podpora pro jednoduché odesílání a příjem multimediálních dat. Nevýhodou knihovny je to, že využívá moderních API webových prohlížečů, takže některé funkce knihovny nemusí fungovat správně nebo vůbec. Použití této knihovny, je tedy vhodnější spíše v experimentálních projektech, které budou využívat jen uživatele s aktuálními webovými prohlížeči. V produkčním mainstrea-

movém prostředí je prozatím asi vhodnější použít konvenčnější metody pro přenos multimediálních dat (flash, webová služba,…), popřípadě implementovat kombinaci obou metod.

Literatura

[1] FIELDING, R. et al. *RFC 2616 - Hypertext Transfer Protocol – HTTP/1.1* [online]. IETF, Červen 1999. [cit. 1. Červen 2016]. Dostupné z: <https://tools.ietf.org/html/rfc2616>.

[2] LORETO, S. et al. *RFC 6202 - Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP* [online]. IETF, Duben 2011. [cit. 1. Červen 2016]. Dostupné z: <https://tools.ietf.org/html/rfc6202>.

[3] KESTEREN, Anne van. *XMLHttpRequest Living Standard* [online]. WHATWG, Listopad 2016. [cit. 9. Listopad 2016]. Dostupné z: <https://xhr.spec.whatwg.org/>.

[4] SCHIEMANN, Dylan. *The forever-frame technique* [online]. Comet Daily, Listopad 2007. [cit. 30. Květen 2016]. Dostupné z: <http://cometdaily.com/2007/11/05/the-forever-frame-technique/>.

[5] HICKSON, Ian. *Server-Sent Events* [online]. W3C, Únor 2015. [cit. 2. Červen 2016]. Dostupné z: <http://www.w3.org/TR/2015/REC-eventsource-20150203/>.

[6] HICKSON, Ian et al. *HTML Living Standard* [online]. WHATWG, Listopad 2016. [cit. 8. Listopad 2016]. Dostupné z: <https://html.spec.whatwg.org/>.

[7] Mozilla Contributors. *Using server-sent events* [online]. Mozilla Developer Network, Září 2015. [cit. 8. Listopad 2016]. Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events>.

[8] Mozilla Contributors. *Server-sent events* [online]. Mozilla Developer Network, Duben 2016. [cit. 15. Červen 2016]. Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events>.

[9] FETTE, Ian a Alexey MELNIKOV. *RFC 6455 - The WebSocket protocol* [online]. IETF, Prosinec 2011. [cit. 1. Červen 2016]. Dostupné z: <http://tools.ietf.org/html/rfc6455>.

[10] SOUZA, Guilherme. *Nodejs Websocket* [online]. Září 2016. [cit. 11. Listopad 2016]. Dostupné z: <https://github.com/sitegui/nodejs-websocket/blob/master/README.md>.

[11] Mozilla Contributors. *WebSockets* [online]. Mozilla Developer Network, Květen 2016. [cit. 15. Červen 2016]. Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API>.

[12] FLETCHER, Patrick. *Introduction to SignalR* [online]. The ASP.NET Site, Červen 2014. [cit. 10. Červen 2016]. Dostupné z: <http://www.asp.net/signalr/overview/getting-started/introduction-to-signalr>.

[13] RAUCH, Guillermo. *Introducing Socket.IO 1.0* [online]. Socket.IO, Květen 2014. [cit. 8. Listopad 2016]. Dostupné z: <http://socket.io/blog/introducing-socket-io-1-0/>.

[14] JOHNSON, Rod et al. *Spring Framework Reference Documentation* [online]. Spring, 2013. [cit. 10. Červen 2016]. Dostupné z: <http://docs.spring.io/spring/docs/4.0.0.RELEASE/spring-framework-reference/htmlsingle/#websocket>.

[15] *Welcome to Atmosphere: The Asynchronous WebSocket/Comet Framework* [online]. Atmosphere Framework, 2016. [cit. 12. Červen 2016]. Dostupné z: <https://github.com/Atmosphere/atmosphere/blob/master/README.md>.

[16] FREEMAN, Adam. *Pro ASP.NET MVC 5 Platform*. Apress, 2014. ISBN 978-1-4302-6541-2.

[17] *Open Source* [online]. The ASP.NET Site, Leden 2015. [cit. 25. Listopad 2016]. Dostupné z: <https://www.asp.net/open-source>.

[18] ROTH, Daniel, Rick ANDERSON a Shaun LUTTIN. *Introduction to ASP.NET Core* [online]. Microsoft Docs, Říjen 2016. [cit. 25. Listopad 2016]. Dostupné z: <https://docs.microsoft.com/en-us/aspnet/core/>.

[19] GONCALVES, Antonio. *Beginning Java EE 7*. Apress, 2013. ISBN 978-1-4302-4626-8.

[20] SIKOS, Leslie F. *Web Standards-Mastering HTML5, CSS3, and XML*. Apress, 2014. ISBN 978-1-4842-0884-7.

[21] JOSEFSSON, S. *RFC 4648 - The Base16, Base32, and Base64 Data Encodings* [online]. IETF, Říjen 2016. [cit. 23. Listopad 2016]. Dostupné z: <https://tools.ietf.org/html/rfc4648>.

[22] SCHNEIDER, John et al. *Efficient XML Interchange (EXI) Format 1.0 (Second Edition)* [online]. W3C, Únor 2014. [cit. 23. Listopad 2016]. Dostupné z: <http://www.w3.org/TR/2014/REC-exi-20140211/>.

[23] BOURNEZ, Carine. *Efficient XML Interchange Evaluation* [online]. W3C, Duben 2009. [cit. 23. Listopad 2016]. Dostupné z: <https://www.w3.org/TR/2009/WD-exi-evaluation-20090407/>.

[24] JACKSON, Wallace. *JSON Quick Syntax Reference*. Apress, 2016. ISBN 978-1-4842-1862-4.

[25] *BSON - Binary JSON* [online]. [cit. 23. Listopad 2016]. Dostupné z: <http://bsonspec.org/>.

[26] FURUHASHI, Sadayuki. *MessagePack specification* [online]. Duben 2013. [cit. 23. Listopad 2016]. Dostupné z: <https://github.com/msgpack/msgpack/blob/master/spec.md>.

[27] *Developer Guide | Protocol Buffers* [online]. Google Developers, Srpen 2016. [cit. 23. Listopad 2016]. Dostupné z: <https://developers.google.com/protocol-buffers/docs/overview>.

[28] *LZ4 - Extremely fast compression* [online]. [cit. 14. Březen 2017]. Dostupné z: <http://lz4.github.io/lz4/>.

[29] *Snappy by google* [online]. [cit. 14. Březen 2017]. Dostupné z: <http://google.github.io/snappy/>.

[30] *Zstandard - Real-time data compression algorithm* [online]. [cit. 14. Březen 2017]. Dostupné z: <http://facebook.github.io/zstd/>.

[31] SALOMON, David. *Data Compression The Complete Reference Second Edition*. Springer, 2000. ISBN 978-3-540-78086-1.

[32] BOUTELL, T. et al. *RFC 2083 - PNG (Portable Network Graphics) Specification Version 1.0* [online]. IETF, Březen 1997. [cit. 15. Březen 2017]. Dostupné z: <https://tools.ietf.org/html/rfc2083>.

[33] BANKOSKI, J. et al. *RFC 6386 - VP8 Data Format and Decoding Guide* [online]. IETF, Listopad 2011. [cit. 15. Březen 2017]. Dostupné z: <https://tools.ietf.org/html/rfc6386>.

[34] VALIN, JM., K. VOS a T. TERRIBERRY. *RFC 6716 - Definition of the Opus Audio Codec* [online]. IETF, Září 2012. [cit. 15. Březen 2017]. Dostupné z: <https://tools.ietf.org/html/rfc6716>.

[35] AGUILAR, José M. *SignalR Programming in Microsoft ASP.NET*. Microsoft Press, 2014. ISBN 978-0-7356-8388-4.

[36] ZHOU, Michael. *Managing Dependencies* [online]. Únor 2016. [cit. 8. Březen 2017]. Dostupné z: <https://github.com/google/closure-compiler/wiki/Managing-Dependencies>.

[37] *PersistentConnection.OnReceived Method* [online]. Microsoft Developer Network. [cit. 10. Březen 2017]. Dostupné z: <https://msdn.microsoft.com/en-us/library/microsoft.aspnet.signalr.persistentconnection.onreceived(v=vs.118).aspx>.

[38] *SignalR/WebSocketHandler.cs* [online]. Duben 2013. [cit. 10. Březen 2017]. Dostupné z: <https://github.com/SignalR/SignalR/blob/d0df7a8004169a70dd7f8c42c06c0bdf68ff6591/src/Microsoft.AspNet.SignalR.Owin45/WebSockets/WebSocketHandler.cs>.

[39] Mozilla Contributors. *MediaDevices* [online]. Mozilla Developer Network, Březen 2017. [cit. 9. Březen 2017]. Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices>.

[40] Mozilla Contributors. *MediaStream* [online]. Mozilla Developer Network, Leden 2017. [cit. 9. Březen 2017]. Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/API/MediaStream>.

[41] CASAS-SANCHEZ, Miguel, Jim BARNETT a Travis LEITHEAD. *MediaStream Recording* [online]. W3C, Prosinec 2016. [cit. 9. Březen 2017]. Dostupné z: <https://www.w3.org/TR/2016/WD-mediastream-recording-20161222/>.

[42] WOLENETZ, Matthew et al. *Media Source Extensions™* [online]. W3C, Listopad 2016. [cit. 9. Březen 2017]. Dostupné z: <https://www.w3.org/TR/2016/REC-media-source-20161117/>.

[43] WOLENETZ, Matthew, Jerry SMITH a Aaron COLWELL. *WebM Byte Stream Format* [online]. W3C, Říjen 2016. [cit. 9. Březen 2017]. Dostupné z: <https://www.w3.org/TR/2016/NOTE-mse-byte-stream-format-webm-20161004/>.

[44] *WebM Container Guidelines* [online]. The WebM Project, Duben 2016. [cit. 9. Březen 2017]. Dostupné z: <https://www.webmproject.org/docs/container/>.

[45] *Specifications* [online]. Matroska. [cit. 9. Březen 2017]. Dostupné z: <https://www.matroska.org/technical/specs/index.html>.

[46] *Apache License, Version 2.0* [online]. The Apache Software Foundation, Leden 2004. [cit. 10. Březen 2017]. Dostupné z: <https://www.apache.org/licenses/LICENSE-2.0>.

[47] *Unlicense Yourself: Set Your Code Free* [online]. Unlicense.org. [cit. 10. Březen 2017]. Dostupné z: <http://unlicense.org/>.

Přílohy

Příloha A

Seznam obrázků

Příloha B

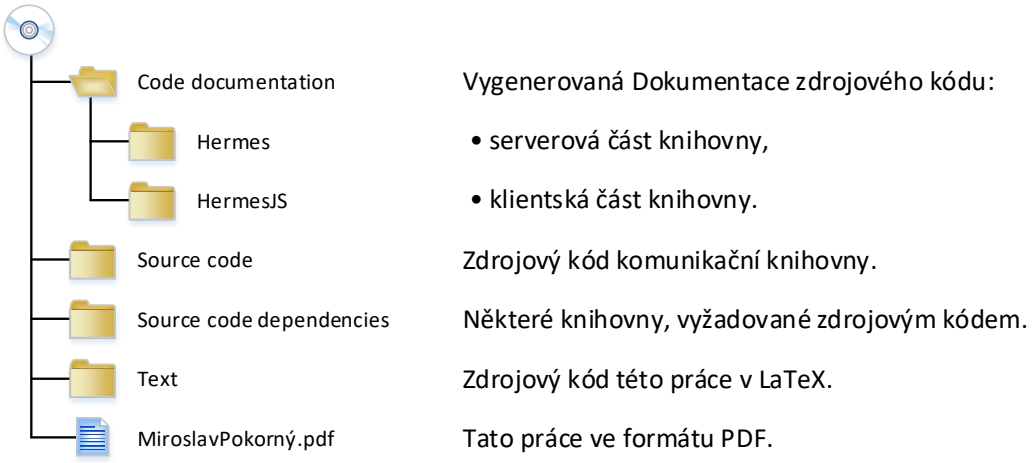
Seznam tabulek

Příloha C

Seznam ukázek kódu

Příloha D

Obsah přiloženého DVD



Obrázek D.1.: Obsah přiloženého DVD

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2016/2017

Studijní program: Aplikovaná informatika
Forma: Prezenční
Obor/komb.: Aplikovaná informatika (ai3-p)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Pokorný Miroslav	Prachovská 255, Jičín - Holínské Předměstí	I14128

TÉMA ČESKY:

Real-time komunikace ve webových aplikacích

TÉMA ANGLICKY:

Real-time communication in web applications

VEDOUCÍ PRÁCE:

Ing. Ondřej Klapka - KIKM

ZÁSADY PRO VYPRACOVÁNÍ:

Základní teorie real-time komunikace na webu
- Popis různých technologií
- Srovnání jednotlivých technologií
- Prozkoumat možnosti komprese zpráv
- Abstraktní vrstva komunikace na webu (komunikační frameworky)

Návrh a implementace komunikační knihovny
- Analýza protokolů, popřípadě návrh a dokumentace nového protokolu, využitého do této knihovny
- Implementace na serverové straně
- Implementace klienta v JavaScript
- Přidat do knihovny podporu pro přenos multimediálních dat (audio, popřípadě video)

SEZNAM DOPORUČENÉ LITERATURY:

SignalR Programming in Microsoft ASP.NET
- Microsoft Press, ISBN: 978-0-7356-8388-4

Podpis studenta: ..... **Datum:** .....

Podpis vedoucího práce: ..... **Datum:** .....

(c) IS/STAG , Portál - Podklad kvalifikační práce , I14128 , 25.10.2016 17:40