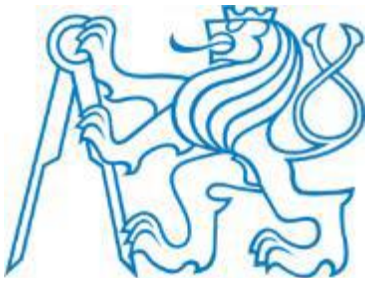


ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta elektrotechnická

Katedra mikroelektroniky

Mobilní aplikace pro řízení v reálném čase

Mobile application for real-time control

Diplomová práce

Studijní program: Komunikace, multimédia a elektronika
Studijní obor: Elektronika
Vedoucí práce: Ing. Stanislav Vítek, Ph.D.

Jan Šára

Praha 2013

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra mikroelektroniky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Bc. Š Á R A Jan**

Studijní program: Komunikace, multimédia a elektronika
Obor: Elektronika

Název tématu: **Mobilní aplikace pro řízení v reálném čase**

Pokyny pro vypracování:

Podejte přehled technologií, vhodných k vytvoření multiplatformní mobilní aplikace pro řízení v reálném čase. Na základě analýzy dostupných nástrojů a technologií navrhněte a realizuje mobilní aplikaci pro vzdálené ovládání robotického dalekohledu, řízeného systémem RTS2. Aplikace bude vhodně uživatelsky konfigurovatelná a umožní přístup k vybrané podmnožině vlastností systému na základě stupně autentifikace.

Seznam odborné literatury:

[1] Adrian Kosmaczewski, Mobile JavaScript Application Development: Bringing Web Programming to Mobile Devices, 2012, ISBN: 1449327850

[2] Pro HTML5 Programming (2nd edition), 2011, ISBN: 143023864X

[3] Petr Kubánek, RTS2—The Remote Telescope System, Advances in Astronomy, 2010, doi:10.1155/2010/902484

Vedoucí: **Ing. Stanislav Vítek, Ph.D.**

Platnost zadání: 31. 8. 2014

L.S.

Prof. Ing. Miroslav Husák, CSc.
vedoucí katedry

Prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 30. 1. 2013

Čestné prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Datum:

……..

podpis diplomanta

Poděkování

Rád bych poděkoval Ing. Stanislavu Vítkovi, Ph.d. za cenné připomínky a rady při vedení této diplomové práce.

Abstrakt

Tato diplomová práce se zabývá vývojem webové mobilní aplikace pro komunikaci s robotickým dalekohledem. Práce poskytuje přehled nástrojů pro vývoj multiplatformní mobilní aplikace. Pro realizaci jsou použity technologie Node.js a WebSocket, které jsou vhodné pro komunikaci v reálném čase. Výsledkem je plně použitelná aplikace sloužící k ovládání dalekohledu na katedře radioelektroniky.

Klíčová slova: jQuery Mobile, Socket.io, Node.js, Websocket

Abstract

This diploma thesis deals with the development of mobile web application to communicate with a robotic telescope. The work provides an overview of tools for developing cross-platform mobile application. For the implementation are used Node.js and WebSocket technology, which are suitable for real-time communication. The result is a fully usable application used to control the telescope at the Department of Radio Electronics.

Key words: jQuery Mobile, Socket.io, Node.js, Websocket

Obsah

1 Úvod

Tato práce se zaměřuje na vývoj multiplatformní aplikace fungující v reálném čase pro ovládání robotického dalekohledu řízeného systémem RTS2. K jejímu vytvoření jsou použity moderní technologie založené na JavaScriptu a HTML5, které jsou dostupné v dnešních moderních prohlížečích. Práce by měla sloužit jako návod všem vývojářům, kteří vyvíjejí aplikaci komunikující v reálném čase pomocí HMTL5. Téma jsem si vybral, protože využívá dynamicky se rozvíjejících metod pro tvorbu nejenom klientských částí aplikací, ale i těch serverových při využití pouze jednoho programovacího jazyka. Práce je inspirována rozsáhlým projektem GLORIA, která je založena na vývojovém prostředí Liferay a poskytuje přístup uživatelům k robotickým dalekohledům. Kromě aplikace bude výsledkem také rozhraní pro komunikaci v reálném čase.

V diplomové práci je nejprve obsaženo seznámení a shrnutí nejznámějších frameworků pro vývoj multiplatformních aplikací. Následuje popis API pro komunikaci se systémem RTS2. V teoretické části jsou také popsány všechny potřebné technologie pro vývoj aplikace jako je Websockets, webové úložiště HTML5 pro ukládání uživatelského nastavení a server Node.js. Práce se také zabývá zálohováním projektu pomocí systému Git, jehož znalost je nezbytná pro instalaci nezbytných částí projektu.

Následuje praktická část, kde je nejprve uvedena a popsána základní struktura aplikace a způsob komunikace mezi jednotlivými komponentami. Poté je zde uvedeno nastavení a instalace Node.js včetně všech potřebných modulů a vytvoření Webserveru. V poslední kapitole je návod pro produkční nasazení aplikace.

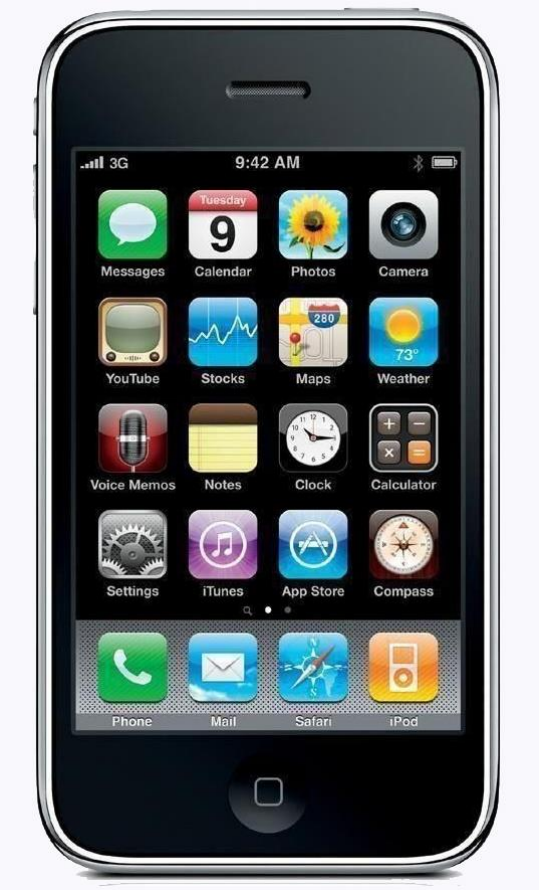
2 Teoretický rozbor

Web prošel v posledních letech obrovským rozvojem, než se dostal do podoby, ve které ho známe dnes. Zatímco první stránky byly statické s pevně daným obsahem, dnes je běžné dynamické načítání stránek a uživatel má často možnost přímo tvořit obsah. Společně s nároky uživatelů vzrůstaly i ty technologické. Už nestačilo pouze načíst a zobrazit data, ale udržovat stále aktuální informace na stránce, umožnit komunikaci lidí v reálném čase atd. Nejvýstižnějším příkladem jsou Google Docs, kde je možné editovat jeden dokument s několika uživateli v reálném čase současně. K dnešním možnostem interakce webu v reálném čase byla však velmi dlouhá cesta.

Prvním významným krokem k dynamickému webu byla technologie AJAX (Asynchonous JavaScript and XML). Díky tomu bylo možné měnit obsah stránky bez nutnosti znovunačtení. Dochází tak k mnohem větší plynulosti práce a snížení zátěže na servery, jelikož jsou obnovována pouze data, která jsou opravdu potřeba. Narůstá však počet HTTP požadavků, a tak při nevhodné implementaci může zátěž serveru naopak stoupnout. Velmi záleží také na kvalitě připojení, které má vliv na plynulost aplikace. Nicméně i přes některé problémy je AJAX velkým krokem vpřed a přibližuje webové aplikace k těm desktopovým. Společně s touto technologií však také nastal problém jakým způsobem a v jakém formátu se mají posílat data mezi klientem a serverem. Touto problematikou se podrobně zabývá kapitola 4.

Dalším velkým milníkem byla snaha pro komunikaci v reálném čase. Tento problém AJAX neřeší, jelikož je vždy nejprve nutné poslat požadavek na stranu serveru, který zašle odpověď. Což nesplňuje definici webu komunikujícího reálném čase, která říká, že je to souhrn metod, technik a postupů, které slouží k tomu, aby se informace dostala k příjemci v ten okamžik, kdy ji autor vydá, a to bez toho, aby musel příjemce kontrolovat, zda se nestalo něco nového (1). Tato definice bývá obcházena pomocí metody “heartbeat”. Jak název napovídá, je založené na periodickém dotazování serveru o aktuální data. Tato metoda však není moc efektivní a způsobuje zbytečné vytěžování serveru a klienta, což je především u mobilních zařízení značně kritické jak z pohledu datové náročnosti, tak z pohledu spotřeby energie.

Pro skutečnou aplikaci pracující v reálném čase je však několik překážek. Ta nejzásadnější z nich spočívá ve schopnosti serveru odeslat požadavek klientovi. Na serveru není problém uchovávat IP adresu klienta, je však problém v následném doručení zprávy. V cestě totiž stojí nejrůznější proxy servery, NATy a firewally. Řešením by byla veřejná IP adresa. Tato podmínka je však v dnešní době nesplnitelná a to z jednoduchého důvodu. S nástupem mobilních zařízení dramaticky vzrostl počet požadavků na přidělení IP adres, kterých je omezený počet. To vyústilo k použití technik, jako je NAT. NAT slouží k překládání adres z lokálních sítí na jedinečnou adresu, která umožňuje přístup do jiné sítě. Díky tomuto řešení, plynoucího z nedostatku IP adres, je dnes nemožné zaručit veřejnou IP dresu všech klientů.



Obrázek 1 Příchod iPhonu značně změnil způsob práce s webem a daty.

Jedním z řešení je vytvořit takzvaný Comet server. Princip tohoto přístupu je velmi jednoduchý. Klient naváže spojení se serverem, který odpoví, ale spojení ponechá otevřené. Vytvoří se tak komunikační kanál. Výhodou je, že není potřeba žádných dodatečných pluginů a knihoven, všechno funguje pouze pomocí webového prohlížeče. Je zde však jedna velmi podstatná nevýhoda, a to že server musí udržovat neustále všechna spojení a blokuje tak svoje prostředky.

Technologií, která řeší všechny stávající problémy je Websocket(podrobně v kapitole 5.3). Funguje na principu síťových soketů, tedy navázání spojení pomocí IP technologie. Soket je tedy popsán pomocí dvou IP adres, protokolem a portem. Soket lze následně použít k přenášení dat. Pro Websocket nepředstavuje problém ani technologie jako firewall, NAT nebo proxy. Spojovací tunel je vytvořen pomocí HTTP příkazu connect. Tím je otevřeno TCP/IP spojení se serverem na specifikovaném portu. Pomocí tohoto tunelu již mohou být posílány zprávy oběma směry. WebSocket má dnes velmi dobrou podporu ve všech významných prohlížečích. (1) (2)

Nyní se nabízí otázka, zda je vůbec něco takového potřeba. Pro odpověď si stačí uvědomit, kam se posunul způsob práce na počítači. Počítač bez připojení k internetu je dnes prakticky nepoužitelný a podle statistik je nejpoužívanějším programem právě webový prohlížeč. Obrovskou výhodou je také to, že webové technologie jsou nezávislé na platformě a zařízení. Snadno tak můžeme používat tu samou aplikace na našem domácím počítači i iPadu. Tím se dostáváme ke vzniku aplikace pro ovládání dalekohledu. Nepochybně by šla vytvořit velmi kvalitní nativní aplikace pro android, ovšem uživatelé ostatních systémů by se k ní nedostali.

A pokud bychom chtěli udržovat aktuální aplikaci na všechny existující platformy, bylo by to mnohem méně efektivní, než vytvořit jednu webovou aplikaci, která se chová prakticky identicky. Tato práce si tedy klade za úkol přinést pohodlné uživatelské rozhraní pro ovládání dalekohledu, ke kterému by se jinak bylo velmi obtížné dostat. Každý se tedy bude moct prohlížet naměřená data a stahovat obrázky, aniž by se musel učit složité příkazy a postupy.

Na závěr tohoto rozboru je potřeba dobré říci, že výše popsané technologie, které budou použity v této práci, mají před sebou obrovskou perspektivu. Firma Google již dnes nabízí celý operační systém běžící v podstatě v klasickém prohlížeči (Chrome OS), přičemž je možné aplikace spouštět i v offline režimu. V budoucnu se tedy dá očekávat, že stále více aplikací se bude přesunovat do prostředí mimo náš počítač a klienti se k nim budou pouze připojovat. Výsledkem tedy budou stále aktuální aplikace bez nutnosti jejich instalace dostupné na všech možných zařízeních.

3 RTS2

3.1 Obecné vlastnosti

RTS2(Remote Telescope System) je systém založený na otevřeném zdrojovém kódu, který slouží pro vzdálenou kontrolu hvězdáren. Úkolem RTS2 je vytvořit systém, který umožní zasílání a přijímání dat, pořizování obrázků, kontrolu a nastavování pozice dalekohledu, zasílání upozornění při změně dat a mnoho dalších funkcí. Veškerá data jsou ukládána pomocí databáze.

V současné době RTS2 běží na mnohých dalekohledech po celém světě. Systém je přizpůsoben mnoha výrobcům a spoustě odlišných zařízení.

Hlavní funkce RTS2:

- Modul pro měření a zpracování gama záření
- PostgreSQL databáze pro ukládání dat
- Možnost snadného pozorování Měsíce, planet, trpasličích planet a slunečních soustav
- Možnost připojení senzorů počasí

3.2 Historie a vývoj RTS2

Současný systém RTS2 vychází z jeho předchůdce RTS (Remote Telescope System), který vznikl na MFF UK pro ovládání teleskopu Meade LX200. Původní verze RTS byla napsána v jazyce Python a měla sloužit k ovládání dalekohledu a pořizování snímků. Následně byl systém kompletně předělán na současnou verzi RTS2. Hlavním důvodem bylo vytvořit více přenosný systém, který by mohl být použit na více hvězdárnách. Jazyk Python, který již nebyl vhodný pro takový účel, byl nahrazen jazykem C a následně C++. Dalším zásadním rozdílem bylo, že RTS2 využívá k ukládání dat a informací z hvězdáren databázi PostgreSQL a ne pouze textové soubory, jako tomu bylo u původní verze RTS.

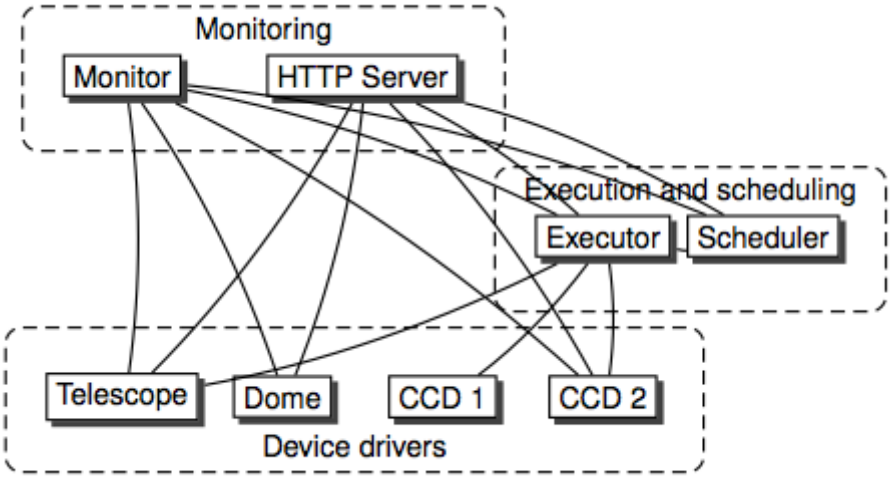
3.3 Architektura

RTS2 je založena na několika nezávislých částech. Programy mezi sebou komunikují pomocí protokolu RTS2. Jednotlivé programy mezi sebou komunikují pomocí protokolu RTS2 (kapitola 3.6). Tento protokol obsahuje příkazy pro zjištění a ověření zařízení, autentizaci a autorizaci, přenos dat a nastavení proměnných. Jednotlivé RTS2 programy mezi sebou komunikují pomocí protokolu TCP/IP. Rozpoznání zařízení a autentizace je zajištěno pomocí modulu rts2-centrald. Jedno zařízení může být připojeno k několika rts2-centrald, což umožňuje, aby například data z jedné meteorologické stanice mohla být využita na několika teleskopech.

Uživatelské programy komunikují také pomocí protokolu RTS2. Příkazy a data jsou posílána ve formátu JSON nebo XML-RPC.

3.4 Struktura RTS2

RTS2 je rozděleno do několika vrstev. Nejnižší vrstva poskytuje přístup k hardwaru skrze RTS2 protokol. Vrchní vrstva slouží k ovládání dané hvězdárny. Poslední, prostřední vrstva, slouží k plánování a vykonávání příkazů od uživatele. Blokové schéma vrstev je popsáno na následujícím obrázku.



Obrázek 2 Schéma vrstev RTS2 (3)

Monitor a HTTP server slouží k uživatelskému vstupu. Telescope, Dome, CCD1 a CCD2 slouží jako rozhraní mezi hardwarem a RTS2. Executor a Selector zajišťují provádění a plánování operací.

Důležitou RTS2 vlastností je, že ke své funkci nepotřebuje žádné standartní rozhraní (ASCOM). Tato vlastnost vyplývá z toho, že k ovládání a zobrazování stavu zařízení jsou potřeba pouze 3 příkazy - čtení hodnoty, zapisování hodnoty a vykonání příkazu. Výhody tohoto řešení spočívají především ve snadné rozšiřitelnosti a pohodlným a rychlým přidáváním různých rozšíření.

3.5 Monitorování

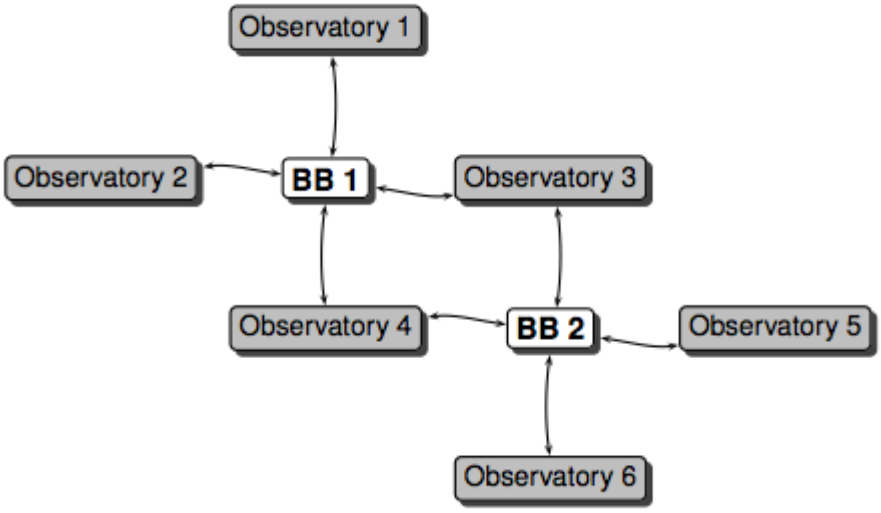
K přímému sledování a nastavování proměnných slouží funkce *rts2-mon*. Tato funkce je většinou přímo připojena k RTS2 přes SSH a má přístup ke všem modulům jednotlivých hvězdáren.

3.6 RTS2 protokol

V současné verzi RTS2 se pro komunikaci mezi serverem a klientem objevuje vlastní protokol. Je inspirován protokolem SMTP a je založený na ASCII kódování. RTS2 protokol uměl v první verzi pouze jednosměrnou komunikaci od klienta směrem k serveru. V současné době jsou si obě strany (klient a server) rovny. Tím pádem není rozdíl mezi klientem (který začíná spojení) a serverem (který zpracovává dotaz).

3.7 RTS2 - síťová komunikace

Veškerá komunikace probíhá přes protokol HTTP pomocí centrálního serveru nazvaného BB. Server komunikuje s hvězdárnami pomocí rts2-xmlrpcd. Komunikace probíhá pomocí rozhraní JSON API. Následující obrázek zobrazuje blokové schéma komunikace mezi BB serverem a několika hvězdárnami.



Obrázek 3 RTS2 - blokové schéma síťové komunikace (3)

3.8 Simulace zařízení

Pro vývoj je velice důležité mít možnost testování. Systém RTS2 nabízí možnost simulace jednotlivých zařízení bez přístupy k hardwaru. Tento systém nabízí možnost použití fotoaparátu, teleskopu, nejrůznějších filtrů a senzorů včetně všech jejich parametrů. Proto po úspěšném otestování je možné aplikaci nasadit na reálné zařízení.

3.9 JSON API

Systém RTS2 nabízí JSON API ke kontrole a ovládání hvězdáren a dalších systémů založených na RTS2. Příkazy jsou odesílány pomocí metod HTTP POST/GET. Dotazy jsou vyřizovány komponentou rts2-xmlrpcd a data jsou vracena také přes HTTP ve formátu JSON. V následující kapitole budou popsány nejdůležitější dotazy využité při práci na webové aplikaci.

3.9.1 API

3.9.1.1 /api/devices

Příklad http://localhost:Example 8889/api/devices

Vrací seznam dostupných zařízení jako pole.

3.9.1.2 /api/lastimage

Příklad http://localhost:8889/api/lastimage?ccd=C0
Parametr ccd -jméno zařízení

Vrací binární data naposledy pořízeného obrázku. V případě chyby vrací JSON výjimku.

3.9.1.3 /api/currentimage

Příklad http://localhost:8889/api/currentimage?ccd=C0&chan=2s
Parametr ccd - jméno zařízení
chan - číslo datového kanálu

Vrací binární data aktuálního obrázku. V případě chyby vrací JSON výjimku.

3.9.1.4 /api/get

Příklad	http://localhost:8889/api/get?d=C0&e=1&from=10000000
Parametr	d -jméno zařízení e - rozšíření. Pokud je nastaveno na 1, vrací navíc speciální metadata. Defaultně nastaveno na 0. from - Vrací data pořízená od požadovaného času. Nepovinný parametr.

Vrací JSON objekt.

"d": *{ "variable name":value,... },*	proměnné zařízení a jejich hodnoty.
"minmax": *{ "variable name":[min, max],... },*	seznam proměnných s maximální a minimální povolenou hodnotou
"idle":0 or 1,	stav dostupnosti. 1 pokud je zařízení dostupné
"stat":	stav zařízení
"f":time	čas odpovědi. Může být použito v dalším požadavku jako parametr

Pokud je nastaven parametr e = 1, v parametru d je pro každou hodnotu vráceno následující pole:

[	
flags,	popis typu hodnoty
value,	aktuální hodnota
isError,1	pokud nastala chyba
isWarning,	1 pokud je varování
description	krátký popis hodnoty
]	

3.9.1.5 /api/set

Příklad	http://localhost:8889/api/set?d=C0&n=exposure&v=200
Parametry	d - jméno zařízení n - jméno proměnné v - hodnota proměnné

Vrací stejná data jako /api/get. Pokud je hodnota proměnné return 0, proměnná byla úspěšně nastavena. V případě chyby má parametr Return hodnotu -2. (3)

4 Komunikační rozhraní a datové struktury

Pro komunikaci s libovolným zařízením je nutné zvolit správný protokol, díky kterému by bylo možné volat vzdálené procedury. K tomuto účelu bylo stvořeno několik komunikačních rozhraní. V následující kapitole budou shrnuty ty nejpoužívanější.

4.1 CORBA

Corba je velmi populární protokol pro vytváření objektově orientovaných aplikací, často používaný pro rozsáhlé podnikové aplikace. Je například využíván v grafickém prostředí Gnome. Nachází velmi dobrou podporu v programovacích jazycích C++ a Java, ale je samozřejmě dostupný i v mnoha dalších. Nabízí jazyk IDL, což je jazyk nezávislý na platformě a použité technologii, sloužící k popisu programového rozhraní. Díky tomu spolu mohou komunikovat komponenty nezávisle na použité technologii. Jeho nevýhodou je příliš velká komplexnost a složitost. Je vhodný spíše pro firemní a desktopové aplikace než pro ty webové.

(4)

4.2 XML-RPC /JSON-RPC komunikace

Aby bylo možno komunikovat se zařízením, je nutné vybrat správný protokol. Zařízení zvládá komunikaci na základě protokolů XML-RPC nebo JSON-RPC.

Jsou to protokoly, díky kterým je možné volat vzdálené procedury. Nejsou to nové technologie, ale pouze soubor pravidel a standardů, které je nutno dodržovat. Data jsou posílána ve formátu XML nebo JSON pomocí protokolu HTTP.

4.2.1 Základní specifikace

4.2.1.1 Požadavek

Každý požadavek se vždy skládá z několika parametrů. Hlavička obsahuje základní informace o přenosu, jako jsou:

- *Druh dotazu* – 3 části
  - Metoda, vždy POST
  - Informace o umístění (URI), může být prázdné
  - Verze a druh protokolu
- *User-agent* – specifikuje druh a verzi implementace
- *HOST* – adresa, na které běží server
- *Content-Type* – druh odesílaných dat
- *Content-Length* – délka dokumentu

V těle požadavku, které je uvozeno do párové značky <methodCall>, jsou uloženy veškeré potřebné informace, jako je volaná metoda a parametry. Metoda je zapouzdřena v párové značce <methodName> a parametry ve značce <param>.

4.2.1.2 Odpověď

Odpověď se opět skládá z hlavičky a těla. Hlavička obsahuje parametry:

- *Verze protokolu, stavový kód*
- *Connection*
- *Content-Length*
- *Date*
- *Server*

V těle je potom uvozena odpověď ve značce <methodResponse>. Informace jsou uvozeny ve značce <params>, která musí obsahovat alespoň jeden parametr <param>, jehož součástí je hodnota a datový typ.

Chybové odpovědi mají podobnou strukturu jako normální odpověď, obsahují ale parametr <fault>, který nese informaci o chybě.

Protokol JSON-RPC je více přímočarý. Dotaz obsahuje pouze povinné parametry. To jsou:

- *Version* – verze protokolu JSON-RPC
- *Method* – volaná metoda
- *Id* – identifikátor, díky kterému je možno přiřadit správnou odpověď k dotazu
- *Params* – parametry dotazu

Odpověď obsahuje obdobné parametry.

- *Version* – verze protokolu JSON-RPC
- *Result* – odpověď
- *Error* – informace o chybách
- *Id* – identifikátor odpovědi

Oba protokoly jsou si velice podobné. Ovšem pro menší datovou náročnost a jednoduchost byl pro aplikaci zvolen protokol JSON-RPC.

(5) (6)

4.3 SOAP

Protokol (Simple Object Access Protocol) je založený na výměně zpráv pomocí HTTP. Zprávy jsou posílány pomocí XML, přičemž všechny údaje jsou uloženy v elementu Envelope. Dále tam jsou elementy Header(nepovinné, používá se pro pomocné informace - např. identifikace uživatele) a element Body, který obsahuje informace identifikující volanou službu a parametry. Dotazy jsou nejčastěji posílány pomocí protokolu HTTP (metoda POST) díky jeho široké podpoře. Nevýhodou může být určitá složitost a zdlouhavá syntaxe XML.

4.3.1.1 Ukázka odpovědi

```
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>

<SOAP-ENV:Body>
```

```
 <m:GetLastTradePriceResponse
  xmlns:m="urn:x-example:services:StockQuote">
  <Price>14.5</Price>
 </m:GetLastTradePriceResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

V první ukázce je vidět požadavek, který obsahuje pouze tělo. Obsahuje vzdálené volání metody *GetLastTradePrice* s parametrem symbol.

(7)

4.4 SOAPjr

SOAPjr je protokol odvozený od SOAP a JSON-RPC. Je určen k tvorbě API pro komunikaci pomocí Ajaxu. Vznikl z toho důvodu, že protokol SOAP, ze kterého vychází, byl trochu těžkopádný, pomalý a díky tomu, že je založený na XML, s sebou nese spoustu nepotřebných dat. Používá podobný princip jako SOAP (hlavní elementy Envelope/Header/Body). Jako formát pro výměnu dat se používá JSON. Dotazy jsou posílány většinou pomocí metody HTTP GET.

4.4.1.1 Ukázka SOAPjr dotazu

```
{
   "HEAD" : {
      "action" : "login",
       "sid" : "80e5b8a8b9cbf3a79fe8d624628a0fe5"
   },
   "BODY" : {
       "username" : "user"
       "password" : "password"
   }
}
{
   "HEAD" : {
       "result" : "1"
   },
   "BODY" : {
     "email" : "email@email.cz"
}
```

4.5 Formáty pro výměnu dat

Z narůstající potřebou interakce mezi jednotlivými aplikacemi bylo potřeba nalézt způsob pro ukládání dat, který by byl snadno čitelný a nezávislý na platformě. Základním kamenem pro výměnu dat bylo po dlouhou dobu XML. V tomto datovém formátu mohou být ukládána základní data jako text, ale také dokumenty, obrázky a video. Nicméně pro spoustu aplikací

může být použití XML nevýhodné. Proto vznikl formát JSON. V následující části budou tyto dva formáty popsány a porovnány.

4.5.1 JSON

JSON (JavaScript Object Notation) je datový formát určený k výměně dat. Je nezávislý na počítačové platformě a data jsou ukládána v polích nebo objektech. Složitost a hloubka struktury není nijak omezena.

```
{
 "array": [
   1,
   2,
   3
 ],
 "boolean": true,
 "null": null,
 "number": 123,
 "object": {
   "a": "b",
   "c": "d",
   "e": "f"
 },
 "string": "Text"
}
```

Ukázka struktury JSON

4.5.2 XML

XML(Extensible markup language) je standardizovaný značkovací jazyk, který slouží k ukládání dat a jejich výměně mezi aplikacemi. Data jsou popsány z hlediska jejich věcného obsahu, což znamená, že XML dokumenty mají vysoký informační obsah.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
 <to>SomeOne</to>
 <from>Me</from>
 <heading>Message</heading>
 <body>How are you?</body>
</note>
```

Ukázka XML

4.5.3 JSON vs XML

V této kapitole budou oba přístupy pro práci s daty porovnány v několika bodech, přičemž budou shrnuty jejich klady a zápory.

4.5.3.1 Čitelnost a srozumitelnost

XML i JSON jsou velmi dobře čitelné, přesto lze říci, že JSON má v tomto směru lehce navrch, jelikož je o něco více striktní.

4.5.3.2 Rozšiřitelnost

Pomocí JSONu lze ukládat pouze prostý text, zatímco XML je v tomto směru více flexibilní a umožňuje přenášet více typů dat.

4.5.3.3 Sdílení dat a dokumentů

V případě sdílení dat má jednoznačně navrch JSON. Je to z toho důvodu, že data v JSONu jsou ukládány pomocí polí a objektů, zatímco v XML jsou ukládána do stromové struktury. Obojí má své výhody, ale první možnost je velmi dobře srozumitelná v objektově orientovaných jazycích. Formát JSON sebou navíc nese daleko méně popisných značek a atributů, což znamená větší datovou náročnost.

Pokud naopak je potřeba přenášet celé dokumenty, tak je jednoznačně lepší použít XML, které může kromě textu obsahovat i data nejrůznějších formátů jako video, audio záznamy nebo obrázky.

Ze srovnání vyplývá, že oba přístupy k datům mají své opodstatnění a záleží na typu aplikace, jaká struktura je vhodnější. Pokud budeme například posílat data pomocí HTTP požadavků do mobilní aplikace, je jednoznačně lepší použít JSON pro jeho datovou nenáročnost a jednoduchost. Pokud naopak budeme muset ukládat nejrůznější typy dat, je nutné použít XML.

(8)

5 Přehled dostupných technologií

5.1 JavaScriptové frameworky

V poslední době dochází k velkému uplatnění cloudových a webových aplikací. Webový prohlížeč je nejdůležitější aplikací a proto dochází k velkému uplatnění technologií souvisejících s HTML5 a JavaScriptem. Programy a aplikace založené na této technologii mají výhodu v tom, že mohou bez problémů běžet na více platformách bez jakékoliv změny. V následující podkapitole se nachází shrnutí dvou nejpoužívanějších JavaScriptových knihoven pro tvorbu webových aplikací a vyhodnocení jejich největších kladů a záporů.

5.1.1 Sencha Touch 2

Sencha touch je JavaScriptová knihovna určená k vývoji mobilních webových aplikací. Je založena na technologiích Javascript, HTML5, CSS3 a umožňuje vytvořit uživatelské rozhraní připomínající nativní aplikaci. Součástí této technologie je i Sencha SDK Tools, která umožňuje exportování webové aplikace do nativní, kterou je následně možné následně nahrát na Android Market nebo Apple Appstore.

Hlavní výhody:

- MVC model
- Relativní rychlost na iOS (oproti JQuery mobile)
- Kvalitní dokumentace s množstvím příkladů

Nevýhody:

- Těžko odhalitelné chyby
- Podpora pouze Android a iOS
- Podpora prohlížečů s jádrem WebKit

5.1.2 JQuery Mobile

Tento Framework je založený na populární knihovně JQuery. Na rozdíl od Senchy, která je kompatibilní s Androidem, iOS a v České republice nepříliš rozšířeným Blackberry, podporuje všechny známé mobilní operační systémy. JQuery Mobile také podporuje technologii PhoneGap, která nám umožní převést aplikaci psanou v JavaScriptu a HTML na mobilní aplikaci, která se chová jako nativní.

Hlavní výhody:

- Jednoduché, nenáročné
- Podporuje všechny existující platformy
- Jednodušší než Sencha Touch
- Snadné ladění kódu
- Možnost snadného vytváření vlastních šablon

Nevýhody:

- Pomalejší než Sencha touch
- Horší struktura kódu (problém u rozsáhlejších projektů)

(9) (10)

5.1.3 Kompatibilita jQuery Mobile a Sencha Touch

Přesto, že je udávána podpora všech typů operačních systémů, u reálných aplikací se často vyskytuje problém se správným zobrazením. Pokud je brána v potaz například roztříštěnost Androidu, je prakticky nemožné odladit aplikaci, která má fungovat na všech rozlišeních, verzích a navíc ještě několika operačních systémech naprosto identicky. V případě Senchy je také velký rozdíl v plynulosti a rychlosti celé aplikace mezi Androidem a iOS v neprospěch Androidu. Obecně však lze říci, že kompatibilita u nejrozšířenějších operačních systémů je dobrá. Velká výhoda také spočívá v přizpůsobení ovládacích prvků na daný operační systém. Programátor se tedy nemusí starat o vzhled jednotlivých tlačítek, listboxů, textových polí atd. O všechny prvky se postará sama knihovna, přičemž prvky jsou navrženy pro pohodlné dotykové ovládání.

5.1.4 Shrnutí a výběr frameworku

Nespornou výhodou HTML5 je jeho přenositelnost. Je možné snadno napsat aplikaci pro iOS, Android, Windows phone a Symbian. Další výhodou je poměrně strmá křivka učení. Programátor, který již má zkušenosti s JavaScriptem, CSS a HTML může prakticky ihned vytvářet aplikace.

Jsou zde ovšem i nevýhody a to poměrně závažné. Mezi ty největší patří například nemožnost push notifikací, nemožnost sledovat aktuální lokaci v případě aplikace běžící na pozadí, nemožnost využívat některých lokálních API. Dále je třeba zmínit, že Sencha Touch je poměrně komplexní a rozsáhlé prostředí, které ovšem postrádá jednoduchost a přímočarost JQuery. JQuery na druhé straně zaostává za Senchou ve výkonu. Obě tyto technologie se tedy hodí spíše na jednodušší a tenčí aplikace. Složitější a náročnější aplikace je výhodnější dělat v nativním prostředí.

Pro případ komunikace s RTS2 je ideální technologie JQuery Mobile. Výsledná aplikace bude snadno přenositelná a kompatibilní s většinou mobilních a desktopových operačních systémů. Tyto výhody jsou pro návrh komunikace s RTS2 rozhodující.

5.2 Node.js

Pomocí JavaScriptu můžeme psát nejen klasické webové aplikace běžící v prohlížeči, ale například i aplikace pro Windows 8 nebo aplikace pro mobilní telefony. A Node.js přidává další možnost využití JavaScriptu, a to využití na serveru. Díky velké komunitě, která se vytvořila okolo tohoto projektu, je dnes k dispozici obrovské množství modulů, které usnadňují nejrůznější úkony. V současné době je podporován mnoha velkými firmami. Je součástí například Yahoo, LinkedIn, eBay nebo Windows Azure. Node.js také nachází velmi dobré uplatnění v programování v reálném čase.

5.2.1 Uplatnění Node.js

- Aplikace s RESTful/JSON API
- Single-Page aplikace ve spolupráci s AngularJS
- Real-Time aplikace

Není vhodné naopak u projektů, které jsou náročné na CPU (například renderování videa).

(11)

5.2.2 Instalace a dostupnost Node.js

Server Node.js je možné nainstalovat na všechny desktopové platformy (Windows, Linux a Mac OS X) a počet hostingů podporujících node.js se stále rozrůstá. Společně s node.js se nainstaluje npm (node package manager), což je vlastně balíčkovací systém podobný Ruby Gems, jenž umožňuje instalaci dodatečných modulů a knihoven. V současné době je k dispozici přes 16 tisíc modulů.

5.2.3 Funkce Node.js

Jak je řečeno v předchozí kapitole, Node.js umožňuje běh JavaScriptu na serveru. O překlad a běh tohoto jazyka se stará Google V8 VM, tedy to samé běhové prostředí, které používá v současné době prohlížeč Google Chrome. Node.js můžeme ve stručnosti popsat jako běhové prostředí a knihovnu. Je dobré si uvědomit, že aplikace v Node.js se liší od klasické aplikace v PHP běžící nejčastěji na Apache HTTP serveru. Pomocí node.js neimplementujeme pouze aplikaci, ale zároveň i HTTP server. Lze teoreticky říci, že webová aplikace a server je jedno a to samé.

5.2.4 Vytvoření základní aplikace pomocí Node.js

V následující ukázce zdrojového kódu je ukázka vytvoření jednoduchého HTTP serveru pomocí Node.js. Na prvním řádku si vložíme modul http, který bude dostupný pomocí proměnné http. Dále zavoláme funkci, kterou modul http nabízí, a to createServer. Tato funkce vrátí objekt, který má metodu listen. Tato metoda přijímá jako parametr číslo portu, kterému HTTP server následně naslouchá. Vytvoření webserveru je podrobně popsáno v kapitole 11.

```
var http = require("http");
var server = http.createServer();
server.listen(8888);
```

Takto vytvořený server nemá žádnou funkci. V ukázce je však vidět základní princip práce v Node.js. Každá aplikace se skládá z modulů, které slouží k lepší čitelnosti výsledného zdrojového kódu. Můžeme tedy v rozsáhlejších projektech snadno udržovat přehledný hlavní soubor, který je spouštěn pomocí node.js a několik modulů, které se starají o funkčnost celého projektu. Hotové moduly je možné instalovat buď pomocí balíčkovacího systému npm, nebo snadno vytvářet vlastní. Při vytváření vlastních modulů je důležité uvědomit si pár zásad.

Modul můžeme chápat jako obyčejný JavaScriptový soubor, pokud bychom ovšem chtěli nadefinovat v našem modulu proměnnou nebo nějakou funkci, např:

```
var test = 5;
function count(a,b){
     return a+b;
}
```

A následně vložili cestu k souboru s naším modulem pomocí *require*:

```
var mujmodul = require(./jmeno_souboru.js);
```

nebyli bychom schopni ji přečíst. Je to z důvodu kolize jmen, která je v JavaScriptu tak kritická. Pokud by v jiném modulu byla definována také proměnná test, došlo by k chybě. Node proto funguje tak, že pokaždé, když načítá modul, vytvoří nový scope (oblast viditelnosti proměnné). To znamená, že nemůže docházet ke kolizím proměnných z jednotlivých modulů. Pokud tedy máme v našem modulu definovanou funkci nebo proměnnou, stačí před ně uvést klíčové slovo *exports*. Náš ukázkový soubor tedy stačí upravit:

```
exports.test = 5;
exports.add = function add(a,b){
     return a+b;
}
```

Pokud následně chceme použít modul, stačí ho přidat do souboru.

```
var nas_modul = require('./jmeno_souboru.js);
var soucet = nas_modul.add(5,3);
```

(12)

5.2.5 Princip a architektura Node.js

Node.js funguje na jiném principu než například PHP, Python, Ruby nebo Java.

Pro názornost je uveden následující příklad:

```
var result = database.query("SELECT * FROM table");
console.log("next line");
```

V tomto případě nejprve překladač node.js načte výsledek z databáze a následně vypíše text do konsole. Tento princip je velice podobný synchronnímu zpracování v PHP. Základem je tedy to, že dokud není dokončen předchozí příkaz, nemůže se začít zpracovávat příkaz následující

a blokuje tedy ostatní. Zatímco však v případě PHP web server spustí s každým HTTP požadavkem nový proces, node.js používá pro všechny requesty pouze jedno vlákno. V PHP nás proto neovlivní například pomalá databáze, jelikož každý požadavek má vlastní proces. V node.js by to však znamenalo poměrně zásadní problém.

Abychom se tomu vyhnuli, upravíme předešlou ukázku do následujícího tvaru:

```
database.query("SELECT * FROM table", function(rows) {
  var result = rows;
});
console.log("next line");
```

V této ukázce je kód zpracováván asynchronně. Funkce database.query() je součástí asynchronní knihovny. Nejprve je zaslán požadavek na databázi, ale nezáleží na tom, jak rychle server vyřídí dotaz a je ihned proveden další řádek, který vypíše řetězec na konzoli. Teprve až server vyřídí požadavek na databázi a pošle výsledek dotazu, tak je provedená funkce předaná jako parametr do database.query(). Tento přístup můžeme popsat jako asynchronní, událostmi řízené volání.

Díky tomu, že node.js má pouze jedno vlákno, dojde k vytvoření a inicializaci všeho potřebného při zpracování prvního požadavku a následně jsou pouze vyřizovány konkrétní dotazy. Naopak v případě kombinace PHP + Apache je pro každý požadavek alokováno určité množství paměti, a proto i v případě zpracování jednoduchého skriptu musí být načtena veškerá konfigurace projektu, spojení s databází atd. Dále je třeba říci, že v jednom vlákně běží pouze výkonný kód programu, který se v podstatě chová jako manažer a rozhoduje o úkolech, které se mají vykonat, případně ukončit. Jednotlivé úkoly se však vykonávají mimo hlavní vlákno.

Nelze tvrdit, že tento přístup je bezchybný a dokonalý. Jako každá technologie má své klady a zápory. Lze však obecně říci, že tento postup je dobře dosažitelný a efektivní.

(13)

5.3 WebSockets

WebSockets je technologie, která umožňuje navázat spojení mezi prohlížečem klienta a serverem. Tato komunikační technologie vychází z protokolu TCP. Od TCP se však liší tím, že umožňuje přenos zpráv, zatímco protokol TCP je založen na přenosu bytů. Vychází ze starších technologií jako je AJAX, který umožňuje zaslat jednorázový asynchronní požadavek klienta na server. Pomocí tohoto API je možné posílat zprávy serveru a přijímat odpovědi v reálném čase. Lze tedy implementovat takzvanou full-duplex komunikaci. Celý přenos je přitom řízen událostmi, lze se tak vyhnout technice heartbeat, která je založená na neustálém dotazování serveru v pravidelných intervalech. Další velkou výhodou WebSockets je řešení problémů s Firewally a proxy servery. WebSockets zjistí, zda je v cestě nějaký proxy server a nastaví tunel pomocí HTTP příkazu connect. Tímto požadavkem se otevře spojení se serverem na určitém portu. Pokud je vytvořen tímto způsobem tunel, je možné libovolně přenášet zprávy oběma směry. (2)

5.3.1 Použití WebSockets

Prakticky ve všech moderních prohlížečích je dnes implementována podpora WebSockets. Pro vytvoření spojení potřebujeme tedy pouze navázat spojení:

```
var connection= new WebSocket("ws://www.nejakyserver.cz/");
```

Pro Web Socket URL je použit prefix ws:// a pokud je použito SSL spojení, lze použít wss://wss

Vytvořený objekt nabízí tři události:

- **onopen** - voláno při otevření spojení
- **onclose** - oznamuje uzavření spojení
- **onmessage** - oznamuje přijetí zprávy ze serveru

Pokud je úspěšně navázáno spojení se serverem (je zavolána událost *onopen*), je možné posílat zprávy, k čemuž slouží metoda send(). V prvních verzích WebSockets bylo možné posílat pouze textové řetězce, ale v poslední specifikaci je možné posílat a přijímat i binární data. Na straně serveru je zapotřebí, aby podporoval technologii WebSockets. Toho lze snadno dosáhnout například pomocí Node.js. Technologie WebSockets klade také nové požadavky na server. Například v případě použití LAMP serveru, který je přizpůsoben na HTTP požadavek a odpověď, může být problém s přizpůsobením na velké množství WebSocket spojení otevřených současně. (14)

Technologie WebSocket je v současné době podporována ve všech moderních prohlížečích a je to klíčový prvek pro vytvoření real-time mobilní aplikace. Pokud by aplikace byla vytvořena pomocí technologie heartbeat, docházelo by k velké datové náročnosti. Neustálá kontrola dat by také měla za následek rychlé vybíjení baterie. Pomocí WebSockets dnes například funguje populární online kancelář Google Docs a spousta real-time her.

5.3.2 Princip Real-Time aplikace pomocí Node.js a Websockets

Pokud chceme vytvořit aplikace komunikující v reálném čase pomocí WebSockets, potřebujeme k tomu dvě části

- server Node.js s instalovaným modulem pro komunikaci pomocí WebSockets
- webový prohlížeč podporující technologii Websockets

5.3.3 Požadavky na server

Server se stará o komunikaci s klientem. Můžeme tedy například přistupovat do databáze a odpovídat na požadavky klienta. Důležitou součástí je WebSocket, který musí být nainstalována v node.js. V současné době máme tři možnosti instalace:

- node-websocket-server
- WebSocket-Node
- knihovnu Socket.io

První možnost je velice snadná na použití, ovšem v současné době nepodporuje verzi WebSockets draft-10. To je poměrně zásadní problém, jelikož tato verze je podporována v prohlížeči Chrome od verze 14. Druhá možnost je také velice snadná na použití a podporuje mnohem více verzí WebSockets. Největší podporu a kompatibilitu i se staršími prohlížeči však poskytuje knihovna Socket.io.

5.3.4 Požadavky na klienta

Jediný požadavek na klienta je podpora technologie WebSocket a zapnutý JavaScript. Klient má za úkol posílat požadavky na server ve formě jednoduchého textu a přijímat zprávy od serveru. Websocket je dnes podporován všemi moderními prohlížeči včetně těch mobilních.

(15)

5.4 HTML5 úložiště

V následující kapitole se nachází porovnání několika možností, jak pomocí HTML5 ukládat data na straně klienta. Je to ideální způsob pro ukládání nastavení webové aplikace, aniž by se musel uživatel přihlašovat.

5.4.1 Web Storage

Web storage je velmi jednoduché lokální úložiště, které uchovává data ve formátu klíč-hodnota. Data je možné ukládat dvěma způsoby:

- sessionStorage - vhodné pro ukládání krátkodobých dat, které mají platnost po dobu relace prohlížeče.
- localstorage - ukládá data pro více relací, doba platnosti dat není ovlivněna chováním prohlížeče.

Operace s daty je prováděna pomocí těchto funkcí:

setItem(key / value) : přidá pár (klíč / hodnota) do objektu,
getItem(key) : načte hodnotu pro daný klíč,
clear() : odstraní všechny páry (klíč / hodnota),
removeItem(key) : odstraní pár (klíč / hodnota),
key(k) : načte hodnotu klíče (k).

Výhody:

- jednoduché API, podpora ve všech moderních prohlížečích

Nevýhody:

- žádný dotazovací jazyk, nehodí se pro ukládání velkého množství dat

(16)

5.4.2 Indexed Database

IndexedDB je technologie umožňující ukládat strukturovaná data a následně v nich rychle vyhledávat pomocí indexů. Hodí se pro ukládání většího objemu dat. V současné době poskytuje oddělené API pro synchronní a asynchronní přístup.

Výhody:

- jednoduché API, komplexnější než Web Storage

Nevýhody:

- zatím žádný dotazovací jazyk, slabá podpora v prohlížečích

(17)

5.4.3 Web SQL Database

Tuto technologii je možné chápat jako sqlite databázi vloženou do prohlížeče. Je podporováno prohlížeči s jádrem Webkit (Safari, Chrome, Opera) a velikost databáze není nijak omezena.

Web SQL API ovšem není v současnosti součástí HTML5 specifikace, je tedy otázkou, jak to s touto technologií bude do budoucna.

5.4.4 Metody pro ovládání Web SQL

1. **openDatabase:** Metoda, která vytvoří databázový objekt s využitím existující databáze. V případě, že databáze neexistuje, vytvoří novou.
2. **transaction:** Metoda, sloužící ke kontrole přenosu dat, která provádí buď potvrzení, nebo vrácení databáze do předchozího stavu.
3. **executeSql:** Metoda pro provádění SQL dotazů.

(18)

Výhody:

- jednoduchá a rychlá implementace SQL

Nevýhody

- Prozatím není standardem W3C
- Není podporováno všemi prohlížeči

5.4.5 Výběr úložiště

Všechny metody mají svůj význam a každá se hodí pro jiný typ aplikace. Situace ovšem v současné době není ideální, jelikož technologie Web SQL není standardizována a je otázkou, jak to bude s podporou v budoucnosti. Implementace IndexedDB také nelze považovat za bezproblémovou. Pro případ aplikace komunikující se RTS je tedy vhodné použít první metodu - Web Storage. Je vhodná především pro jednoduchost, jelikož je nutné ukládat pouze uživatelem nastavované proměnné. Výhodou je také její dobrá podpora ve všech dostupných prohlížečích včetně všech majoritních mobilních operačních systémů.

(19)

5.5 Canvas vs SVG

V dnešní době existují dvě rozšířené technologie pro vložení pokročilé grafiky do stránky (nutné například pro tvorbu grafů, ovládacích widgetů atd.). V následující kapitole budou popsány obě technologie včetně jejich výhod a nevýhod.

5.5.1 Scalable vector graphics

Scalable vector graphics (SVG) je vektorový grafický formát, který je součástí DOM. Proto je velice flexibilní a umožňuje vytvářet nejrůznější animované a interaktivní animace, efekty, filtry. Obsah SVG, který je dán párovým tagem <svg> je možné upravovat pomocí CSS.

```
<svg height="1000px" width="1000px">
  <rect id="myRect" height="100px" width="100px" fill="blue"/>
</svg>
```

Ukázka použití SVG - vytvoření modrého čtverce.

5.6 Canvas

Další možností jak vytvářet složitější animace je použití technologie canvas. Canvas je objekt definovaný tagem <canvas> v HTML stránce, do kterého je možné kreslit. Samotné kreslení potom probíhá pomocí takzvaných kontextů, které mohou být 2D nebo 3D(WebGL). Základním rozdílem oproti SVG je ten, že tu není žádný DOM, vše jsou pouze pixely. Díky tomu se v paměti nebude vytvářet stále složitější objektový model, vše je závislé pouze na velikosti canvasu.

```
<canvas id="myCanvas" width="1200px" height="1200px"></canvas>
```

Ukázka vytvoření “plátna” pro vykreslování.

V kontextu 2D nemá Canvas žádné možnosti pro animování. Pokud tedy chceme udělat jednoduchou animaci, musíme stále dokola volat funkci pro překreslení, na rozdíl od SVG, které animace obsahuje.

5.6.1 Shrnutí obou technologií

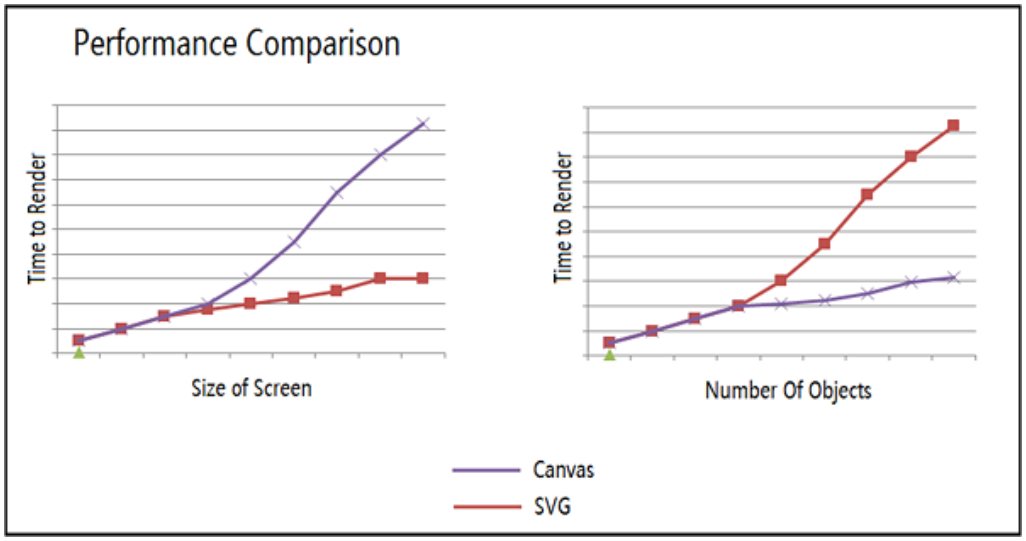
5.6.2 Canvas

- založeno pouze na manipulaci s pixely
- pouze jeden HTML element
- překreslení pouze pomocí API daného kontextu
- ideální pro menší plátno a více objektů

5.6.3 SVG

- založeno na vykreslování tvarů
- tvořeno více HTML elementy
- překreslení pomocí CSS nebo API
- ideální s menším počtem objektů na větším plátně

Na následujícím obrázku je srovnání obou technologií ilustrující závislost na rozlišení a počtu objektů.



Obrázek 4 Porovnání Canvasu a SVG

Ze srovnání na obrázku vyplývají výhody a nevýhody obou technologií. Výhodou canvasu je, že si ho lze představit jako výkonné 2D plátno, které má stálý výkon, měnící se pouze s velikostí plátna. Lze tedy snadno editovat obrázky, aplikovat nejrůznější filtry a provádět operace, které vyžadují přístup k pixelům. Nevýhodou je naopak absence DOM a chybějící API pro animace, vše musí být překresleno pomocí časovačů - canvas tedy není vhodný pro tvorbu uživatelského rozhraní.

SVG má naopak výhodu v nezávislosti na rozlišení a ve velmi dobré podpoře pro animace. Máme přístup ke všem elementům přes DOM, tím pádem je velmi snadné reagovat na události od uživatelů, což umožňuje vytvořit multiplatformní uživatelské rozhraní. Nevýhodou SVG je vzrůstající náročnost, závislá na počtu elementů.

Canvas se hodí především pro úpravu obrázků, manipulaci s videem, vizualizaci dat nebo vytváření grafiky pro hry. SVG je naopak vhodné použít pro tvorbu interaktivních uživatelských rozhraní, grafů. Obě technologie mají tedy svůj prostor pro využití. Vždy je důležité zvážit, zda problematiku není možné řešit pouze pomocí HTML+ CSS. Často může být vhodné použít obě technologie naráz - vykreslení rastrové grafiky pomocí canvasu a animace pomocí SVG. (20) (21)

5.7 Nerelační databáze a NoSQL přístup k datům

Základní vlastností všech NoSQL databází je to, že nevyužívají dotazovací jazyk SQL (jeho použití však není vyloučeno). Důvod vzniku nerelačních databází je především ten, že v poslední době došlo k velkému nárůstu objemu dat, na který nebyly relační databáze optimalizovány. Využívaly pevně stanovená datová schémata a běžely na jednom serveru. Proto jsou dnes stále populárnější datové struktury založené na nerelačním přístupu.

5.7.1 NoSQL – nerelační přístup k datům

NoSQL databáze můžeme chápat jako jednoduché úložiště klíč/hodnota. Hodnota v tomto případě může obsahovat téměř cokoliv, například datový záznam ve formátu JSON. Protože je databáze nerelační a nepodporuje tedy žádný dotazovací jazyk nebo spojení tabulek, je potřeba všechny dotazy dělat "ručně". Díky jednoduchosti oproti SQL-databázím umožňuje vyšší škálovatelnost a rychlost. NoSQL databáze jsou často využívány pro velká cloudová úložiště (Amazon S3, Google Storage).

5.7.2 Výhody a nevýhody NoSQL přístupu

Výhody:

- obecně rychlejší (jednodušší datový model, absence ACID transakcí)
- jednoduché rozšiřování dokumentů o nové atributy
- jednoduchá možnost dotazování
- nástroje pro obnovu, monitorování, zálohy (vše pouze pomocí JSONu a JavaScriptu)
- sharding - možnost sdílení dat na několika zařízeních
- cena

Nevýhody:

- Větší datová náročnost
- Menší funkcionalita (absence JOIN)
- NoSQL databáze neposkytují takovou spolehlivost (může dojít k případ porušení konzistence dat)
- absence univerzálního dotazovacího jazyka

(22)

Ze srovnání vyplývají základní výhody a nevýhody nerelačních databází. Největší silou NoSQL přístupu je flexibilita a škálovatelnost, což ho činí ideální pro práci s velkými daty. Na druhou stranu nabízí menší funkcionalitu než relační databáze. Pro spoustu aplikací je však stále vhodnější a také efektivnější použít klasický relační přístup. Ani o jednom způsobu přístupu k datům (SQL vs. NoSQL) nelze říci, že je lepší než ten druhý. Použití tedy záleží na typu a architektuře aplikace.

5.7.3 MongoDB

MongoDB je dokumentově orientované nerelační databáze vytvořená společností 10gen, přičemž v roce 2010 byla vydána první stabilní verze. Je to multiplatformní opensource databáze napsaná v C++. Je to takzvaně nerelační („NoSQL" databáze). Podporuje jazyky Java, Python, Ruby, PHP a JavaScript.

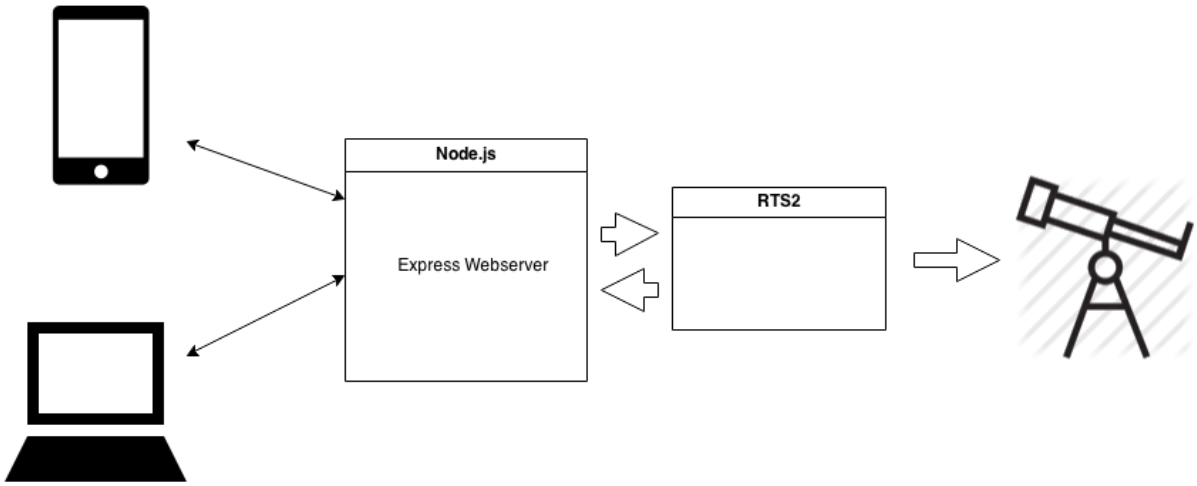
S databází MongoDB se komunikuje pomocí jednoho z podporovaných jazyků, přičemž data jsou ukládána ve formátu BSON(Binary JSON). V klasické databázi typu MySQL máme databáze, které obsahují několik tabulek s jasně danou datovou strukturou. V MongoDB jsou také databáze, které ovšem neobsahují tabulky, ale takzvané kolekce. Každá kolekce může obsahovat libovolný počet dokumentů, které obsahují data uložená ve formátu JSON. Jelikož v MongoDB nemusíme specifikovat, jak budou data v tabulce vypadat, nemusíme předem vytvářet žádnou strukturu ani kolekci. Vše bude vytvořeno automaticky při prvním vložení záznamu. Pokud se následně zeptáme na data z kolekce, která ještě neexistuje, vrátí se prázdný záznam. Z logiky databáze vyplývá, že zde neexistují dotazy typu JOIN jako v SQL (lze však implementovat pomocí technologie map-reduce). To často vede k tomu, že data jsou v databázi duplikována. Na jedné straně je tedy jednoduchý způsob získávání dat, na straně druhé je složitější aktualizace záznamů.

(23) (24)

6 Realizace aplikace

6.1 Základní struktura

Aplikace se skládá ze dvou hlavních částí - klientské a serverové. Klientská část se stará o interakci s uživatelem, posílá a přijímá data od Node.js. Obě části jsou spolu propojené pomocí technologie WebSocket, což umožňuje okamžitě reagovat na události. Kromě WebSocketů probíhá komunikace také pomocí metody HTTP POST (správa a ověřování uživatel). Node.js server přijímá a vyřizuje žádosti od jednotlivých klientů. Na obrázku č.5 je zobrazeno blokové schéma aplikace.



Obrázek 5 Základní schéma aplikace

6.1.1 Funkce klienta

Klient má za úkol zprostředkovat uživateli přístup ke všem dostupným nastavením. Je vytvořen pomocí technologie jQuery Mobile. Ovládací prvky by tedy měly být přizpůsobené mobilním zařízením. Přístup k seznamu zařízení a hodnotám je umožněn pouze registrovaným uživatelům. Parametry je možné nastavovat z jednotlivých widgetů, které si lze libovolně navolit. Ty základní dovolují sledování základních proměnných, ty složitější potom slouží k zobrazování dat v grafu, načítání galerie nebo pořizování snímků.

6.1.2 Funkce Node.js serveru

Server komunikuje s klientem především pomocí WebSocketů, aby veškerá data mohla být okamžitě aktualizována. Výjimku tvoří pouze přihlašování, registrace a odhlašování uživatelů, které probíhá pomocí Ajaxu. Server ovšem nezajišťuje komunikaci pouze s klientem, ale také s RTS2. V následující kapitole bude popsán veškerý způsob komunikace.

6.1.3 Uživatelské rozhraní

Klientská část aplikace se skládá ze dvou hlavních stránek. Výchozí stránka obsahuje automaticky generovaný seznam všech dostupných zařízení. Druhá stránka obsahuje seznam

konkrétních proměnných pro dané zařízení. Uživatel může navolit proměnné, které mají být sledovány. Kromě toho může také navolit, které proměnné se mají zobrazovat ve výběru a ty ostatní odfiltrovat. Nastavení je možné kdykoliv změnit.

(25)

6.1.4 jQuery Mobile - Ajax a načítání stránek

jQuery Mobile používá navigační systém, který využívá načtených stránek pomocí technologie Ajax. Nejprve jsou všechny stránky v dokumentu načteny a každý požadavek na stránku je potom vyřízen jako Ajax požadavek. Jedna stránka je uvozena pomocí atributu *data-role=”page”*. Element s tímto atributem může obsahovat jakýkoliv obsah, ale je zvykem používat uvnitř elementy, které specifikují header, content a footer. Pokud dokument neobsahuje žádnou stránku, jQuery jednu vytvoří automaticky.

Pokaždé, když je kliknuto na libovolnou stránku, je ihned zobrazena, aniž by muselo dojít k obnovení celé stránky.

Technicky probíhá načtení jedné stránky tak, že jQuery Mobile vyhledá požadovaný element s data-role=”page” a vloží do DOM. Všechny další části jako například přiložené styly a skripty jsou zahozeny.

Pro vytvoření struktury stránek existují dvě možnosti:

- model s jednou stránkou,
- model s více stránkami najednou.

V případě první možnosti je každá stránka uložena ve vlastním HTML dokumentu. V případě druhé možnosti každý HTML dokument obsahuje několik stránek najednou, přičemž jsou uvozeny pomocí unikátního id.

Je důležité podotknout, že pokud dochází k přechodu modelu s jednou stránkou na model s více stránkami, je nutné vypnout Ajax. To lze provést pomocí atributů *rel=”external”* nebo *data-ajax=”false”*. Oba výrazy mají stejnou funkci, ale *rel=”external”* je doporučeno použít v případě přechodu na jinou doménu, zatímco *data-ajax=”false”* je používáno pro navigaci mezi stránkami uvnitř jedné domény. Oba parametry způsobí kompletní načtení nové stránky.

6.1.5 Přidávání nových elementů do stránky

Další důležitou vlastností je trochu odlišné přidávání elementů do stránky. Pokud použijeme standartní funkci jQuery pro vložení elementu (např. metoda append), nedojde k požadovanému efektu. Je to z důvodu dynamického načítání stránek pomocí Ajaxu. Proto je potřeba po vložení nového elementu do stránky zavolat speciální funkci. Jsou zde opět dvě možnosti. První možností je zavolat funkci trigger("create") na požadovaný element. Tato funkce se používá například při vložení nového prvku (neuspořádaný seznam). Druhou možností je funkce refresh (např. *listview('refresh')* pro obnovení seznamu). Tato funkce je na rozdíl od create použita v případě, že element již existuje, ale jeho obsah byl nějakým způsobem změněn.

6.2 Komunikační rozhraní RTS - Node.js - Klient

6.2.1 Komunikace klienta směrem k serveru

Třída Connection v klientské části aplikace se stará o veškerou komunikaci pomocí WebSocketů. Služby, které poskytuje, jsou popsány v následujících podkapitolách.

setData

Funkce odesílá nové hodnoty k nastavení. Hodnoty jsou uloženy v poli.

Parametry:

Název události: *set*
Parametr dat: *update* (obsahuje JSON s proměnnými)

Příklad: socket.emit('set', { update: JSON.stringify(updateArray) });

chooseVariable

Funkce pro vybrání proměnné pro sledování.

Parametry:

Název události: *choosevariables*
Parametr dat: *variable* - název sledované proměnné
set - proměnná typu bool(informace pro server, zda má posílat aktuální hodnoty proměnné)
Příklad: socket.emit('chooseVariables', { variable: focstep, set: true });

getData

Funkce pro získání dat pro konkrétní zařízení.

Parametry:

Název události: *get*
Parametr dat: *device* - název zařízení

Příklad: socket.emit('get', { device: F0 });

getDevices

Funkce pro získání seznamu všech zařízení

Parametry:

Název události: getDevices

Příklad: socket.emit('getDevices', { device: "null" });

graph

Funkce pro přijímání dat do grafu.

Název události: *graph*
Parametry dat: *variable* - název sledované proměnné
set - proměnná typu bool (informace pro server, zda má posílat aktuální hodnoty proměnné)
interval – interval pro obnovení

Příklad: socket.emit('graph', { variable: variable, set: set, interval: insterval });

6.2.2 Komunikace serveru směrem ke klientovi

Server odpovídá na požadavky klienta. Odpovědi jsou zpracovávány asynchronně. O zpracování odpovědi na straně klienta se stará třída Connection.

devices

Odpověď serveru na požadavek k získání seznamu zařízení.

Parametry:

Název události: *devices*
Parametr dat: *message* - seznam zařízení

Příklad: *socket.emit('devices', { message: devices })*

actualdata

Odpověď s aktuálními hodnotami z RTS2. Server pravidelně v určitém intervalu kontroluje, zda na serveru nedošlo ke změně. Pokud došlo ke změně a týká se některé z proměnných vyžádaných klientem, je odeslána klientovi. Nedochází tedy k zatěžování klienta nepotřebnými informacemi, ale jsou odesílána pouze relevantní data.

Parametry:

Název události: *actualdata*
Parametr dat: *message* - seznam aktuálních hodnot

Příklad: *socket.emit('actualdata', { message: data })*

variables

Odpověď serveru na požadavek o získání seznamu proměnných pro konkrétní zařízení.

Parametry:

Název události: devices
Parametr dat: message - seznam proměnných

Příklad: socket.emit('variables', { message: variables })

checkresponse

Odpověď serveru po nastavení proměnných. Po pokusu o nastavení proměnné vrací RTS2 seznam všech hodnot pro dané zařízení (podobně jako po volání metody *get*), navíc je zde pouze parametr *return*, který oznamuje, zda byla proměnná úspěšně nastavena.

Parametry:

Název události: *checkresponse*
Parametr dat: *message* - seznam proměnných + return

Příklad: *socket.emit('checkresponse', { message: variables })*

graphData

Odpověď serveru s daty do grafu. Data jsou periodicky odesílána v požadovaném intervalu.
Parametr dat: *value* – nová hodnota

Název události: *graphdata*
socket.emit("graphData", { value: anotherValue });

6.2.3 Přijímání obrázků

Obrázky jsou přijímány jako binární data. Je možné zobrazit aktuální obrázek- metoda *currentimage*, případně poslední pořízený obrázek- metoda *lastimage* (viz. kapitola 3.9 - RTS2 API). Obrázky jsou ve formátu jpg a v případě dotazu je obrázek zobrazen přímo v prohlížeči bez využití WebSocketu.

6.2.4 Komunikace Node.js a RTS2

Komunikace probíhá pomocí metody HTTP GET. Data jsou posílána ve formátu JSON. RTS2 server vyžaduje pro každý dotaz uživatelské oprávnění pomocí HTTP Basic Auth. Modul needle (základní popis v kapitole 6.3) nabízí podporu pro tento způsob ověření. Dotaz tedy poslán:

```
needle.get('https://api.server.com', { username: 'name',
password: pass },
  function(err, resp, body){
    // used HTTP auth
});
```

Ukázka HTTP GET dotazu s HTTP Basic Auth

Z ukázky je zřejmé, že stačí do parametrů uvést uživatelské jméno a heslo, o vše ostatní se modul postará automaticky.

6.3 Nastavení a instalace Node.js

V této kapitole bude popsán postup nastavení a zprovoznění serverové části aplikace. Node.js slouží jako spojovací uzel mezi klientem a RTS2. Jeho úkolem je tedy posílat aktuální data na stranu klienta a zároveň odesílat informace o změnách na RTS2.

6.3.1 Instalace a přehled potřebných modulů

Pro správnou funkci Node.js je zapotřebí instalovat několik modulů. V následujícím seznamu je uveden jejich seznam, základní popis a návod na použití.

6.3.2 Websocket

První modul, který je třeba nainstalovat pro real-timeovou aplikaci je websocket. Pro instalaci použijeme výchozí balíčkovací modul npm. Instalaci tedy provedeme příkazem:

```
npm install websocket
```

6.3.3 Needle

Pro komunikaci s RTS2 potřebujeme zasílat HTTP požadavky z Node.js. K tomuto účelu slouží modul needle. Jeho instalaci lze provést příkazem:

```
npm install needle
```

Modul needle má mnoho funkcí včetně zasílání HTTP požadavků. Základní implementace vypadá následovně:

```
needle.get('http://www.google.com', function(error, response,
body){
 console.log("Got status code: " + response.statusCode);
});
```

V předchozí ukázce je ukázka použití. Dojde k zaslání HTTP požadavku pomocí metody GET.

6.3.4 Nodemon

Následující modul není pro funkci serveru nutný, je však velice platný pro ladění aplikace. Modul můžeme nainstalovat pomocí příkazu:

```
npm install -g nodemon
```

Použití neznamená téměř žádnou změnu. Jediný rozdíl je, že server není spouštěn pomocí příkazu node server-file.js, ale pomocí příkazu nodemon server-file.js. Výhoda tohoto modulu spočívá v tom, že kdykoliv dojde ke změně nějakého souboru v kořenovém adresáři serveru, je server automaticky restartován a není to nutné dělat manuálně. Největší využití tedy tento server nachází při ladění aplikace.

6.3.5 MongoJS

MongoJS je modul, sloužící k přístupu k databázi MongoDB (podrobnější popis nerelačních databází se nachází v kapitole č.5.7. Ke komunikaci používá API, které je velice podobné výchozí JavaScriptové konzoli, dostupné po instalaci MongoDB. Modul je možné instalovat příkazem:

```
npm install mongojs
```

Spojení s lokální databází lze provést následovně:

```
var mongojs = require('mongojs');
var db = mongojs(connectionString, [collections]);
```

Základní API pro komunikaci s databází

Pro vložení záznamu do databáze slouží metoda *save*

```
db.users.save({email: "email@gmail.com", password: "pass},
function(err, ok) {
  if( err || !ok ) console.log("Error");
  else console.log("OK");
});
```

Pro aktualizaci záznamu slouží metoda analogicky *update*

```
db.users.update({email: "email@gmail.com"}, {$set: {password:
"new_pass"}}, function(err, updated) {
  if( err || !updated ) console.log("Error");
  else console.log("OK");
});
```

A pro vyhledání záznamu slouží metoda *find* se syntaxí

```
db.users.find({name: "jmeno"}, function(err, users) {
  if( err || !users) console.log("Uživatel nenalezen");
  else users.forEach( function(users) {
    console.log(users.name);
```

Pro práci s databází MongoDB je možné použít ještě modul mongoose. Základním rozdílem mezi těmito dvěma moduly je ten, že mongoose vyžaduje vytvoření schématu pro databázi, což je v podstatě předpis, jak bude vypadat struktura databáze. Výhodou tohoto přístupu je nejenom následná čitelnost struktury databáze, ale i mnohem větší komplexnost, kterou tento modul nabízí. Pro složité projekty, kde je potřeba udržovat rozsáhlý databázový model a pracovat s mnoha daty, je tak lepší použít modul mongoose. Pro jednodušší projekty je ovšem mongojs plně dostačující.

(26)

6.3.6 Express

Express je framework pro vytoření webserveru, pomocí kterého snadno vytvoříme serverovou část projektu. Podporuje mnoho šablonovacích modulů (Jade, EJS). Podporuje architekturu MVC a umožnuje snadnou implementaci routování, zpracovávání požadavků a směrování. Instalaci lze provést příkazem:

```
npm install express
```

6.3.7 EJS

EJS je šablonovací systém pro Node.js. Instalaci lze provést příkazem `npm install ejs` nebo případně použít pro vytvoření projektu příkaz `express --ejs`, který se postará o automatickou instalaci všech potřebných knihoven. Použití je velice jednoduché a na rozdíl od

podobně populárního modulu Jade vyžaduje pouze znalosti HTML. Pokud tedy k vyrenderování stránky použijeme např. kód, vypsání proměnné provedeme:

```
<div class="title">
    <%= variable %>
</div>
```

Modul EJS ovšem poskytuje spoustu užitečných vlastností usnadňujících práci. Mezi nejdůležitější patří například psaní "filtrů", které výrazně zjednodušují manipulaci s daty.

(27)

6.3.8 Debugování - node-inspector

Modul pro debugování, stejně jako modul nodemon, nemá na funkci serveru vliv. Umožnuje pohodlné debugování pomocí webového prohlížeče. Instalace je možná pomocí příkazu:

npm install node-inspector

Pokud chceme spustit serverovou aplikaci, kterou chceme debugovat, přidáme parametr –debug:

node --debug serverFile.js

Pro spuštění serveru je však lépe použít příkaz nodemon, který nám vždy automaticky restartuje server. Poslední krok ladění souboru je spuštění modulu node-inspector a to provedeme příkazem:

node-inspector &

Tento příkaz vypíše URL, na které je možné debugovat soubor běžící na serveru pomocí libovolného webového prohlížeče.

6.4 Socket.io

Socket.io je JavaScriptová knihovna, která nachází uplatnění v realtime webových aplikacích. Skládá se ze dvou částí - klientské a serverové. Obě části mají prakticky totožné API. Stejně jako node.js, je Socket.io řízeno událostmi. Primárně využívá pro komunikaci Websockety, je ale možné použít mnoho dalších protokolů (Adobe Flash sockets, JSONP atd.). Výhodou je velká podpora všech prohlížečů.

Pomocí npm lze knihovnu nainstalovat do node.js:

npm install socket.io

Použití je velice podobné jako základní API pro Websocket. V následující části bude popsáno základní použití. Je třeba si uvědomit, že cestu k této knihovně je potřeba vložit i na straně klienta. Ke správnému načtení je třeba uvést celou cestu ke knihovně, například takto:

```
<script
src="http://example.com/socket.io/socket.io.js"></script>
```

V ukázce č.6.5 je uvedena ukázka základního použití na straně serveru. Po aktivování socket.io pomocí *io.sockets.on* je možné ihned komunikovat s klientem pomocí dvou základních funkcí, které poskytuje Socket.io API. První z nich je *socket.emit*, která slouží k odeslání zprávy na stranu klienta. Funkce má dva argumenty, první z nich obsahuje název události a druhý data. Druhá metoda - *socket.on* má naopak za úkol zachytávat jednotlivé události a reagovat na ně.

V ukázce č.6.6 je vidět příklad použití na straně klienta. Funkce socket.on a socket.emit pracují zcela analogicky jako na straně serveru. Podstatný rozdíl oproti základnímu API je ten, že už není potřeba tří metod (onopen, onclose, onmessage) na práci s Websockety. Na vyřizování všech událostí včetně těch rezervovaných, jako je error, disconnect, reconnect atd. je používána metoda socket.on.

```
io.sockets.on('connection', function (socket) {
  socket.emit('news', { hello: 'world' });
  socket.on('my other event', function (data) {
    console.log(data);
  });
});
```

Ukázka č.6.5 Socket.io na straně serveru.

```
var socket = io.connect('http://localhost');
  socket.on('news', function (data) {
    console.log(data);
    socket.emit('my other event', { my: 'data' });
  });
```

Ukázka 6.6 Socket.io na straně klienta.

(28)

6.5 Vytvoření Webserveru

V následující kapitole je popsán postup pro vytvoření webserveru. Je zde postupováno od základního řešení bez přídavných modulů, až po finální návrh použitelný v reálné aplikaci.

Node v základní instalaci obsahuje HTTP module, pomocí kterého je velice jednoduché vytvořit základní webserver.

```
// Require what we need
var http = require("http");

// Build the server
var app = http.createServer(function (req, res) {
  fs.readFile(__dirname + req.url, function (err,data) {
    if (err) {
      res.writeHead(404);
      res.end(JSON.stringify(err));
      return;
      }
      res.writeHead(200);
      res.end(data);
  });
}).listen(1337);

app.listen(1337, "localhost");
console.log("Server running at http://localhost:1337/");
```

Ukázka vytvoření jednoduchého webserveru (29)

V předchozí ukázce nejprve vytvoříme server pomocí *http.createServer*. Metoda *http.createServer* přijímá jako parametr funkci, která vyřizuje požadavky a vložíme ho do proměnné *app*. Nakonec nastavíme port serveru na 1337.

V tomto případě si tedy server vezme cestu k souboru, která je obsažena v requestu a zobrazí ho. Pokud soubor neexistuje, zobrazí chybové hlášení. Tento příklad funguje správně a jeho výhodou je, že není potřeba instalovat žádný přídavný modul - vše je dostupné ve výchozí instalaci Node, bohužel má i několik nedostatků. Mezi ty nejzávažnější patří absence cachování stránek na straně klienta. Server statických souborů by měl správně odeslat odpověď "Not Modified" v případě, že nedošlo k žádným změnám. (30) (29)

Předešlá ukázka je velice jednoduchá a vzdálená reálné aplikaci, je však vhodná pro pochopení základní struktury webserveru.

6.6 Přidání mezivrstvy

Další možnost pro vytvoření webserveru je použití modulu connect, který nabízí více abstrakce. V následující ukázce se nachází jednoduchý příklad.

```
// Require the stuff we need
var connect = require("connect");
var http = require("http");

// Build the app
var app = connect();

// Add some middleware
app.use(function(request, response) {
  response.writeHead(200, { "Content-Type": "text/plain" });
  response.end("Hello world!\n");
});

// Start it up!
http.createServer(app).listen(1337);
```

Ukázka - Webserver pomocí modulu connect (29)

V předchozí ukázce je nejprve vytvořena za pomoci modulu *connect* proměnná *app*. Dále je pomocí metody *app.use* nastavena základní funkci webserveru (v této základní ukázce je funkce webserveru naprosto identická s ukázkou v kapitole 6.5) Následně už jenom předáme proměnnou app metodě *http.createServer*. Přidaná mezivrstva („middleware") je tedy v podstatě jakýsi správce žádostí. Použití modulu connect však stále není ideální. Pro představu uveďme ukázku webserveru se dvěma stránkami:

```
app.use(function(request, response, next) {
  if (request.url == "/") {
    response.writeHead(200, { "Content-Type": "text/plain" });
    response.end("Welcome to the homepage!\n");
    // The middleware stops here.
  } else {
    next();
  }
});

// About page
app.use(function(request, response, next) {
  if (request.url == "/about") {
    response.writeHead(200, { "Content-Type": "text/plain" });
    response.end("Welcome to the about page!\n");
    // The middleware stops here.
  } else {
```

```
    next();
  }
});
 // 404'd!
app.use(function(request, response) {
  response.writeHead(404, { "Content-Type": "text/plain" });
  response.end("404 error!\n");
});
```

Ukázka správce požadavků modulu connect (29)

Modul connect postupuje od první definice pro zpracování požadavku až po tu poslední, dokud nenajde definici, která by mu vyhovovala. Tedy konkrétně v předchozí ukázce se nejprve podívá zda url požadavku vyhovuje "/", pokud ne, podívá se dále, kde je podmínka "/about". Pokud url nesplňuje ani jeden požadavek, webserver vrátí chybu 404. Celý proces lze však zjednodušit ještě více a to pomocí modulu Express.

6.7 Express

Konečně se dostáváme k finálnímu modulu Express, který nabízí nejvíce možností. Celý modul vychází z modulu Connect a i původní inicializace je velice podobná.

```
var express = require("express");
var http = require("http");
var app = express();


http.createServer(app).listen(1337);
```

Ukázka inicializace Webserveru pomocí modulu Express

Jak je vidět v této ukázce, vytvoření webserveru pomocí Express je velice podobné jako u modulu Connect. Nabízí nám ovšem daleko více možností směrování, vyřizování požadavků a snadné použití šablonovacích systémů.

6.7.1 Směrování

Nejlepší způsob jak ukázat jednoduchost směrování modulu Express je následující ukázka.

```
app.all("*", function(request, response, next) {
  response.writeHead(200, { "Content-Type": "text/plain" });
  next();
});


app.get("/", function(request, response) {
  response.end("Welcome to the homepage!");
});

app.get("/about", function(request, response) {
```

```
  response.end("Welcome to the about page!");
});

app.get("*", function(request, response) {
  response.end("404!");
});
```

Ukázka směrování pomocí Express (29)

V předchozí ukázce vidíme ukázku směrování, která je funkčně naprosto identická s ukázkou v kapitole 6.6, řešení je ovšem na první pohled kratší a přehlednější.

6.7.2 Vyřizování requestů

Express nabízí i další jednoduché možnosti směrování a poskytuje nám možnosti *redirect* nebo *sendFile*.

```
response.redirect("/public/about");
response.redirect("http://www.seznam.cz");
response.redirect(301, "http://www.seznam.cz");  // HTTP status
code 301

response.sendFile("/path/to/video.mp4");
```

6.7.3 Šablonovací systém

Obrovskou výhodou Express je možnost použít několik šablonovacích systémů. Mezi nejznámější patří například Jade nebo EJS.

```
// Start Express
var express = require("express");
var app = express();

app.engine('.html', require('ejs').__express);

app.set('views', path.join(__dirname, 'views'));

app.set('view engine', 'html');
```

Ukázka vytvoření serveru společně se šablonovacím systémem

V předchozí ukázce je příklad použití šablonovacího systému EJS. Nejprve pomocí metody app.engine řekneme, jaký systém chceme používat. Dále řekneme, že chceme používat soubory s koncovkou .html a nastavíme adresář, ze kterého se mají šablony načítat. Pokud chceme šablonu renderovat, stačí zavolat funkci:

```
res.render('template', {
  variable:16
});
```

Nyní se vyrenderuje šablona ze složky /views/template.html, ve které bude dostupná proměnná variable s hodnotou 16.

Express je tedy ideální možnost jak pohodlně a rychle vytvořit webserver společně se šablonovacím systémem v Node. Lze ho částečně přirovnat například k frameworku Rails v případě Ruby, přesto že pracuje na nižší úrovni. Pomocí tohoto modulu lze také velice snadno implementovat ověřování uživatel.

(29)

7 Produkční nasazení aplikace

7.1 Správa zdrojového kódu a systém Git

Git je velice rychlý a efektivní nástroj pro zprávu verzí vyvinutý původně pro správu verzí linuxového jádra. Uživatelé nestahují pouze nejnovější verzi souborů, ale uchovávají kompletní kopii repozitáře. Pokud dojde například ke kolapsu serveru, lze jej obnovit zkopírováním repozitáře od libovolného uživatele. Každá lokální kopie je plnohodnotnou zálohou všech dat. Lze také jednoduše a pohodlně spolupracovat na projektu s několika skupinami lidí. Systém Git je v dnešní tobě integrován do většiny vývojových prostředí, které ještě více usnadňují správu verzí projektů. Následující kapitola poskytuje základní úvod pro práci s Gitem, protože bez základních znalostí Gitu se v prostředí Node.js nelze obejít. Všechny moduly jsou zálohovány pomocí tohoto systému a práci s ním se nelze vyhnout. (31)

7.1.1 Základy vlastnosti systému Git

Většina systémů uchovává informace jako seznamy změn jednotlivých souborů, tedy jako sadu souborů a seznamů změn těchto souborů v čase. Git pracuje na odlišném principu. Soubory chápe spíše jako sadu snímků. Pokaždé, když se v systému něco změní, Git zkontroluje, jak soubory vypadají v daném okamžiku, a uloží referenci na tento okamžik. Git tedy chápe soubory spíše jako snímky, které uloží.

Díky tomu, že máme uloženou kompletní historii projektu na našem lokálním disku, jsou všechny změny a revize starších verzí projektu prakticky okamžité. Díky tomu, že Git používá ke své činnosti pouze soubory uložené lokálně, nejsou potřeba informace z jiných počítačů. Základní výhoda tohoto principu spočívá v tom, že Git stále nemusí vyhledávat informace na serveru, ale načte je ihned z lokální databáze.

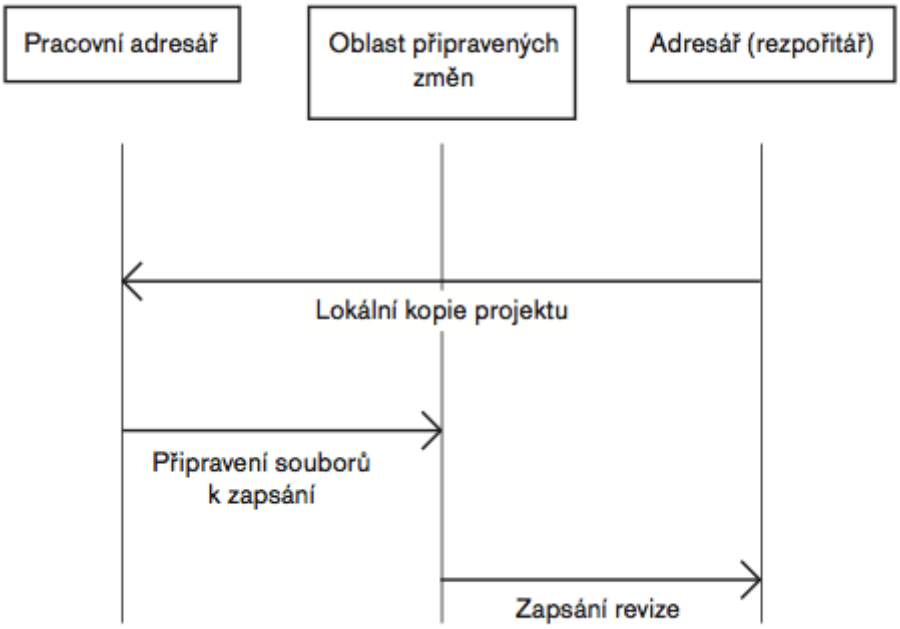
Git používá pro soubory tři základní stavy: zapsáno, změněno a připraveno k zapsání. Zapsáno znamená, že data jsou uložena v databázi. Změněno znamená, že v souboru byly provedeny změny, ale ještě nebyly uloženy do databáze. Připraveno k zapsání znamená, že soubor v aktuální verzi je připraven k zapsání v další revizi. (32)

Z těchto důvodů je projekt rozdělen v systému Git do tří částí:

- adresář systému Git
- pracovní adresář
- oblast připravených změn

Postup zápisu dat vypadá následovně:

- Změní se soubor v pracovním adresáři
- Soubory jsou připraveny tak, že jsou jejich snímky vloženy do oblasti připravovaných změn
- Revize je zapsána. Snímky souborů uložené v oblasti připravovaných změn jsou trvale uloženy do adresáře Git.



Obrázek 6 Pracovní adresář, oblast připravených změn a adresář Git (31)

Obr. Pracovní adresář, oblast připravených změn a adresář Git

7.1.2 Práce se systémem Git

Git lze v dnešní době velice pohodlně nainstalovat na všechny známé operační systémy. V následující kapitole bude popsán základní postup správy projektu.

7.1.3 Vytvoření projektu

Po vytvoření projektu pomocí webové služby využívající systému Git (např. Github nebo Bitbucket), musí být projekt nejprve inicializován. To lze provést tak, že přejdeme do složky projektu a spustíme příkaz:

git init

Tímto příkazem jsou vytvořeny všechny potřebné soubory. Následně je vhodné vytvořit soubor README, který obsahuje základní popis projektu.

touch README

Poté už jenom přiložíme soubor do projektu příkazem:

git add README

Nyní je celý projekt nastaven a uložen na lokálním úložišti. Pokud chceme projekt nahrát na vzdálené úložiště, musíme nejprve vytvořit spojení se vzdáleným serverem. K tomuto účelu slouží příkaz:

git remote add origin https://github.com/uzivatelske-jmeno/jmeno-projektu.git

Po úspěšném spojení se serverem už stačí pouze nahrát soubory příkazem:

git push origin master

7.1.4 Stáhnutí projektu

Největší síla gitu je v možnosti práce více lidí na jednom projektu a sdílení zdrojových kódů. Operace, kdy chceme použít nějaký projekt, se nazývá "*forking*". Pokud tedy chceme vytvořit kopii existujícího repozitáře, použijeme k tomu přikaz *git clone*. Tím dojde ke stažení kopie téměř všech dat, která jsou na serveru včetně všech verzí projektu. Pokud tedy někdy v budoucnosti dojde k poruše na serveru, data mohou být snadno obnovena pomocí některého z klonů. Celý příkaz je

git clone https://github.com/username/Some-project.git

Pokud chceme mít stále aktuální repozitář, je nutné spustit další příkaz. Přejdeme tedy do složky s kopií projektu a spustíme příkazy:

git remote add upstream https://github.com/octocat/Spoon-Knife.git

git fetch upstream

Prvním příkazem vytvoříme spojení na vzdálený server a druhým příkazem vždy aktualizujeme repozitář. Pokud chceme nahrát takto zkopírovaný repozitář na vzdálený server, použijeme k tomu příkaz:

git push origin master

Je důležité se uvědomit, že tímto příkazem se repozitář nahraje k uživateli, který soubor zkopíroval, nikoliv tam, odkud byl zkopírován. Pokud chceme aktualizovat repozitář tam, odkud byl stažen (chceme se například podílet na tvorbě zdrojového kódu), je nutné použít tzv. pull request.

Pokud se stane, že repozitář, který zkopírujeme je v budoucnu aktualizován, a my si nechceme přepsat změny, které jsme mezi tím v kódu provedli, lze obě verze spojit. Nejprve zavoláme známý příkaz *git fetch upstream* a potom zkopírovaný i originální projekt spojíme příkazem:

git merge upstream/master

(33) (31)

7.1.5 GitHub a BitBucket

Github a BitBucket jsou v dvě webové služby, kam lze nahrávat projekty pomocí Gitu. V následující kapitole budou shrnuty jejich základní vlastnosti.

7.1.5.1 Cena

Obě služby mají odlišně nastavené cenové podmínky. Na Githubu se platí podle počtu soukromých repozitářů, zatímco počet uživatelů není limitován. BitBucket naopak nastavuje cenu podle počtu uživatelů, kteří jsou součástí týmu, zatímco počet soukromých repozitářů není nijak omezen.

7.1.5.2 Vývojářské možnosti

Obě služby jsou si funkčně velice podobné, jsou zde však drobné rozdíly. Zajímavou vlastností Bitbucketu je možnost psát hodnocení na commity. Naopak velice využívaná vlastnost na Githubu, která nemá svůj ekvivalent na Bitbucketu, je možnost vytvoření statických stránek na subdoméně Github.io.

Ze srovnání vyplývá, že každá služba se hodí na jiný typ projektu. Pokud vývoj probíhá v malém týmu na větším množství projektů, ideální je použít Bitbucket. GitHub se více hodí pro open-source projekty. Výhoda je také možnost editování přímo v prohlížeči, naopak Bitbucket nabízí výhodnější cenovou politiku pro soukromé projekty, kdy se cena odvíjí od množství spolupracovníků. Proto je BitBucket více využíván pro korporátní účely.

(34)

7.2 Instalace a nastavení pomocí package.json

V kapitole č.6.3 jsou vidět všechny potřebné moduly pro instalaci a běh aplikaci. Přesto, že správce balíčků npm funguje velice dobře, nastavení celého projektu a instalace všech komponent postupně je poměrně nepřehledné. Proto je často do kořenového adresáře zahrnut soubor package.json. V tomto souboru jsou uvedeny všechny potřebné závislosti a verze modulů.

7.2.1 Vytvoření package.json

Základní struktura souboru package.json se skládá ze dvou částí. První je specifikace potřebných modulů a druhá je verze node. Každý modul má specifikovanou verzi. Pro názornost je uvedena následující ukázka:

```
{
  "name": "application-name",
  "description": "A package using versioning",
  "version": "0.0.1",
  "private": true,
  "scripts": {
    "start": "node app.js"
  },
  "dependencies": {
    "express": "3.4.3",
    "ejs": "*"
  }
 "engine": "node >= 0.4.1"
}
```

V předešlé ukázce je potřebné zmínit dvě věci:

- Verze node, která je aktuálně nainstalovaná je porovnána s verzí specifikovanou s package.json. Pokud nainstalovaná verze nesplňuje požadavky, instalace selže.
- V případě modulů instalace probíhá tak, že je nainstalovaná nejvyšší možná verze, která splňuje požadavky. Moduly jsou instalovány do adresáře node_modules, který se nachází v kořenovém adresáři projektu. Pokud není nalezena žádná verze, která by splňovala podmínky, instalace proběhne neúspěšně.

7.2.2 Způsob verzování modulů

Často je dobré vyjádřit požadované verze více explicitně. Platí totiž jistá konvence v číslování modulů. Například modul 0.2.16 znamená:

- hlavní, stabilní verze - major version (0)

- minoritní verze (2)
- opravná verze - patch version (16)

Může se ovšem stát, že jednotlivé verze spolu nebudou kompatibilní. Proto lze nahradit konkrétní verzi modulu například 0.2.x. Tímto říkáme, že pokud dojde k vydání nové opravné verze, bude nainstalována. Hlavní a minoritní verze ovšem stále zůstává stejná.

Po vytvoření všech závislostí a nastavení, stačí přejít do kořenového adresáře a spustit příkaz *npm install* a dojde k instalaci všech potřebných závislostí.

(35)

7.3 Produkční nasazení Node.js

V dnešní době existují hostingy jako Heroku nebo Microsoft Azure. Lze však použít i vlastní server. Je však potřeba ošetřit některé krizové události (výpadek serveru, restartování). K tomuto účelu slouží framework Stagecoach. Tento framework poskytuje mechanismus pro běh několika aplikací nezávisle na jednom serveru, jejich automatické restartování při změnách na serveru a úpravu URL (změna portu). Aplikace tedy mohou bez problémů běžet společně se serverem Apache. Další výhodou vlastního serveru je rychlost, jelikož spousta služeb neposkytuje MongoDB server a proto pro připojení k této databázové službě je potřeba využít jinou službu. V tomto případě může docházet k latenci. Další výhodou stagecoach je také to, že obsahuje nástroj pro pohodlné přesunutí aplikace na server.

(36) (37)

7.3.1 Požadavky

Stagecoach vyžaduje pro svůj běh Unix server, na kterém musí být nainstalované potřebné programy (podrobně popsáno v kapitole o instalaci 7.3.3). Pro distribuci Ubuntu je vytvořen jednoduchý instalátor, který se postará o všechny závislosti. Společně s tímto frameworkem je vhodné nainstalovat Node modul forever. Lze to provést příkazem:

```
npm install -g forever
```

forever je nástroj, který zajišťuje, že proces je automaticky restartován, pokud dojde k chybě.

7.3.2 Stagecoach framework

Nejdůležitější části tohoto frameworku jsou sc-deploy a sc-proxy. Sc-deploy je jednoduchý skript určený pro správu projektů. Je založen na technologii rsync, což je jednoduchý program pro Unixové systémy, který synchronizuje adresáře mezi jednotlivými místy za použití co nejmenšího přenosu dat. Sc-deploy s každou nahranou verzí projektu vytvoří novou složku a pokud vše proběhne v pořádku, vytvoří na ni odkaz a data se budou načítat z této poslední verze projektu. Ve výchozím nastavení je udržováno 5 posledních verzí projektu. V případě chyby lze tedy velice snadno přenastavit verzi projektu, která má být použita.

Sc-proxy je proxy server pro webové aplikace, které naslouchají na několika portech. Vychází z modulu node-http-proxy. Tento nástroj je ideální pro běh a testování několika Node projektů na jednom produkčním serveru a zároveň umožňuje běh aplikací na portu 80.

7.3.3 Instalace - základní postup

Před instalací je nutné zajistit následující podmínky:

- Unix server (ideálně Ubuntu 10.04 nebo vyšší)
- nainstalovaný správce balíčků npm
- databáze MongoDB

Kromě výše zmíněných podmínek je také vhodné mít nainstalovaný systém Git. V Ubuntu lze snadno provést příkazem

```
sudo apt-get install git-core
```

Následně už lze stáhnout celý framework těmito příkazy do adresáře */opt*:

```
cd /opt
git clone git://github.com/punkave/stagecoach.git
```

Nyní je vhodné využít přiložené skripty pro instalaci všech potřebných balíčků. Tu lze provést i z oficiálních balíčků obsažených v ubuntu nebo jiné linuxové distribuce, ty ovšem nemusí být aktuální. V případě ubuntu lze kompletní instalaci provést příkazem:

```
sc-proxy/install-node-and-mongo-on-ubuntu.bash
```

Nyní je potřeba překopírovat soubor s nastavením /opt/stagecoach/settings.example do /opt/stagecoach/settings a změnit parametr USER na uživatele, který bude spouštět aplikace. Dále je potřeba nastavit uživatelská práva a zpřístupnit složku */opt/stagecoach/apps*, ze které se budou spouštět všechny aplikace pro uživatele uvedeného v settings.example.

Nyní je možné vytvořit vzorovou aplikaci tak, že vytvoříme složku potřebnou adresářovou strukturu.

```
mkdir -p /opt/stagecoach/apps/example_app/data
```

Zbývá vytvořit soubor hosts, kde bude uvedena adresa, na které aplikace poběží. Soubor se nachází ve složce data. Tato složka se nikdy nepřepisuje při nahrávání nové verze aplikace, proto je vhodné do ní umístit kromě konfiguračních souborů například datové složky.

7.3.4 sc-proxy - nastavení portu

Nyní je aplikace připravena a zbývá nastavit proxy. K tomu musíme přejít z kořenového adresáře do sc-proxy a zkopírovat obsah souboru *config-example.js* do souboru *config.js*. V tomto souboru je vygenerován vzorový modul pro směrování. Pokud ovšem chceme naslouchat na portu 80, je potřeba změnit port na serveru Apache, pokud bude používán společně se serverem Node. K běhu sc-proxy je potřeba také nainstalovat všechny potřebné moduly a závislosti. Díky balíčkovacímu systému npm stačí v adresáři spustit příkaz *npm install*, který se o vše postará. V tuto chvíli je všechno nastaveno a můžeme manuálně spustit sc-proxy příkazem:

```
sudo node server.js &
```

A potom už zbývá jenom spustit všechny aplikace příkazem:

```
./sc-start-all
```

Nicméně existuje lepší způsob pro spouštění aplikací. Lze vytvořit startovací skript, který povolí automaticky zapínat sc-proxy po každém startu serveru. Pro tento skript stačí zkopírovat potřebné soubory do /etc/init:

```
cp upstart/stagecoach.conf /etc/init
```

Teď stačí spustit příkaz:

```
start stagecoach
```

7.3.5 sc-deploy - správa projektů

Druhou částí instalace je sc-deploy. K tomu je potřeba znova stáhnout celý framework a vytvořit symbolický link, aby bylo možné v terminálu použít příkaz *sc-deploy*. K tomu slouží příkaz:

```
sudo ln -s /opt/stagecoach/bin/sc-deploy /usr/local/bin/sc-deploy
```

Potom je potřeba nastavit soubor *settings*, ve kterém je nastavení pro připojení ke vzdálenému serveru. Parametr USER je jméno uživatele, který má práva ke složce s projekty a parametr SERVER je adresa serveru. Pokud vše nastavíme, stačí použít příkaz

```
sc-deploy production
```

a vše bude nahráno na server. Po nahrání by měla být aplikace ihned dostupná. Při každém dalším spuštění stačí pouze zadat příkaz sc-deploy production a všechny změny budou nahrány na server. Na serveru je uchováváno několik posledních verzí každé aplikace, které je možné přepínat.

8 Závěr

V této diplomové práci byla vytvořena aplikace pro ovládání dalekohledu využívajícího systém RTS2. Aplikace je plně funkční a může být ihned nasazena v reálném provozu. Celý projekt je založen na stále populárnější technologii Node.js, což umožňuje běh JavaScriptu na serveru. Aplikace dokazuje, že je v dnešní době bez problémů možné vytvořit kompletní aplikace pomocí jednoho jazyka.

Výsledná knihovna pro komunikaci se serverem je vytvořena s důrazem na maximální flexibilitu a ukazuje obecný způsob, jak mohou zařízení komunikovat v reálném čase.

V první části je proveden teoretický rozbor týkající se všech použitých technologií a poskytuje nutné znalosti pro práci s Node.js a pro vytvoření mobilní webové aplikace pomocí JavaScriptu. Tato část obsahuje výběr frameworku pro vývoj aplikace, způsob komunikace s RTS2 a popis základní popis všech použitých technologií.

V praktické části je vysvětlen postup vývoje všech potřebných částí. Tato část je psána jako návod pro práci s Node.js a je zde popsán způsob instalace modulů, vytvoření webserveru a způsob komunikace pomocí WebSocket. Součástí návodu je také popis instalace a nastavení Node.js na produkčním serveru.

Citovaná literatura

1. Malý, Martin. Kometa přináší web v reálném čase. *Zdrojak.cz.* [Online] 23. srpen 2010. http://www.zdrojak.cz/clanky/kometa-prinasi-web-v-realnem-case/.

2. —. Web Sockets. *Zdrojak.cz.* [Online] 14. prosinec 2009. http://www.zdrojak.cz/clanky/web-sockets/.

3. Kubánek, Petr. *Robotic Telescope Network for Transients.* 2013.

4. XML-RPC vs. Other Protocols. [Online] http://tldp.org/HOWTO/XML-RPC-HOWTO/xmlrpc-howto-competition.html.

5. JSON-RPC 1.0 Specifications. *JSON-RPC.* [Online] leden 2012. http://json-rpc.org/wiki/specification.

6. *XML-RPC.* [Online] UserLand Software, Inc, 14. červen 1999. http://xmlrpc.scripting.com/default.html.

7. Využití webových služeb a protokolu SOAP při komunikaci. [Online] http://www.kosek.cz/diplomka/html/websluzby.html.

8. Mikoluk, Kasia. JSON vs XML: How JSON Is Superior To XML. [Online] 16. srpen 2013. [Citace: 13. prosinec 2013.] https://www.udemy.com/blog/json-vs-xml/.

9. jQuery Mobile. [Online] 14. duben 2013. http://jquerymobile.com/.

10. Hartmann, Stefan. Sencha Touch vs. jQuery Mobile. [Online] 26. červen 2012. http://www.fusonic.net/en/blog/2012/06/26/sencha-touch-vs-jquery-mobile/.

11. Mrozek, Jakub. JavaScript na serveru: Patří budoucnost Node.js? [Online] 21. září 2012. http://www.zdrojak.cz/clanky/javascript-na-serveru-patri-budoucnost-node-js/.

12. Blohowiak, Aaron. Creating Custom Modules. [Online] 28. červenec 2010. http://howtonode.org/creating-custom-modules.

13. Kiessling, Manuel. The Node Beginner . [Online] 17. duben 2012. http://www.nodebeginner.org/#finding-a-place-for-our-server-module.

14. Eiji, Kitamura a Malte, Ubl. Introducing WebSockets: Bringing Sockets to the Web. *HTML5.* [Online] 20. říjen 2010. http://www.html5rocks.com/en/tutorials/websockets/basics/.

15. Sikora, Martin. Node.js & WebSocket - Simple chat tutorial. [Online] 2011. Node.js & WebSocket - Simple chat tutorial.

16. Patrik, Janáček a Martin, Šimeček. Web Storage (lokální úložiště) – HTML5. *Programujte.com.* [Online] 26. březen 2012. http://programujte.com/clanek/2012022801-web-storage-lokalni-uloziste-html5/.

17. IndexedDB. *Mozilla Developer Network.* [Online] leden 2013. https://developer.mozilla.org/en-US/docs/IndexedDB.

18. HTML5 - Web SQL Database. *Tutorials Point.* [Online] 2011. http://www.tutorialspoint.com/html5/html5_web_sql.htm.

19. Simms, Christian. HTML5 Storage Wars - localStorage vs. IndexedDB vs. Web SQL. [Online] 18. květen 2011. http://csimms.botonomy.com/2011/05/html5-storage-wars-localstorage-vs-indexeddb-vs-web-sql.html.

20. How To Choose Between SVG and Canvas. *Internet Explorer Dev Center.* [Online] Microsoft Corporation, leden 2013. http://msdn.microsoft.com/en-us/library/ie/gg193983(v=vs.85).aspx#HTML5_Graphic_Technologies.

21. Malý, Martin. SVG, nebo Canvas? Vyberte si. *Zdroják.* [Online] 27. říjen 2010. http://www.zdrojak.cz/clanky/svg-nebo-canvas-vyberte-si/.

22. Stanislav, Heller a Tomáš, Volf. Datastore MongoDB Stub pro Google App Engine SDK. [Online] 14. říjen 2013. http://www.datakon.cz/media/2013/Datakon-2013-Heller_Vof-prezentace.pdf.

23. Mrozek, Jakub. JavaScript na serveru: MongoDB, Mongoose a AngularJS. *Zdrojak.* [Online] 26. říjen 2012. http://www.zdrojak.cz/clanky/javascript-na-serveru-mongodb-mongoose-angularjs/.

24. Klobása, Pavel. Letmý technologický pohled na MongoDB. [Online] 23. leden 2012. http://vsadnajavu.cz/2012-01/databaze/letmy-technologicky-pohled-na-mongodb/.

25. AJAX Navigation. *jQuery Mobile.* [Online] 2012. http://jquerymobile.com/demos/1.3.0-rc.1/docs/demos/widgets/ajax-nav/.

26. mongojs. *GitHub.* [Online] https://github.com/mafintosh/mongojs.

27. Embedded JavaScript templates. *Node.js.* [Online] https://npmjs.org/package/ejs.

28. Rauch, Guillermo. Socket.io. [Online] 2012. http://socket.io.

29. Hahn, Evan. Understanding Express.js. [Online] 29. květen 2013. http://evanhahn.com/understanding-express-js/.

30. How to serve static files. *nodejitsu.* [Online] září. 26 2011. http://docs.nodejitsu.com/articles/HTTP/servers/how-to-serve-static-files.

31. Chacon, Scott. *Pro Git.* Praha : l CZ.NIC, z. s. p. o., 2009. 978-80-904248-1-4.

32. Základy práce se systémem Git - Nahrávání změn do repozitáře. [Online] 2012. http://git-scm.com/book/cs/Z%C3%A1klady-pr%C3%A1ce-se-syst%C3%A9mem-Git-Nahr%C3%A1v%C3%A1n%C3%AD-zm%C4%9Bn-do-repozit%C3%A1%C5%99e.

33. Fork A Repo. *GitHub.* [Online] 10. září 2013. https://help.github.com/articles/fork-a-repo.

34. Github vs BitBucket: The determination. [Online] 23. červenec 2013. http://nudowdeployer.wordpress.com/2013/07/23/github-vs-bitbucket-2/.

35. Robbins, Charlie. Package.json dependencies done right. *nodejitsu.* [Online] 24. květen 2011. http://blog.nodejitsu.com/package-dependencies-done-right.

36. Deploying node.js and mongodb in production with stagecoach. *justjs: node.js tutorials.* [Online] 14. červenec 2012. http://justjs.com/posts/deploying-node-js-and-mongodb-in-production-with-stagecoach.

37. Host multiple Node apps on your Linux servers. *GitHub.* [Online] https://github.com/punkave/stagecoach.

Příloha A: testovací dalekohled

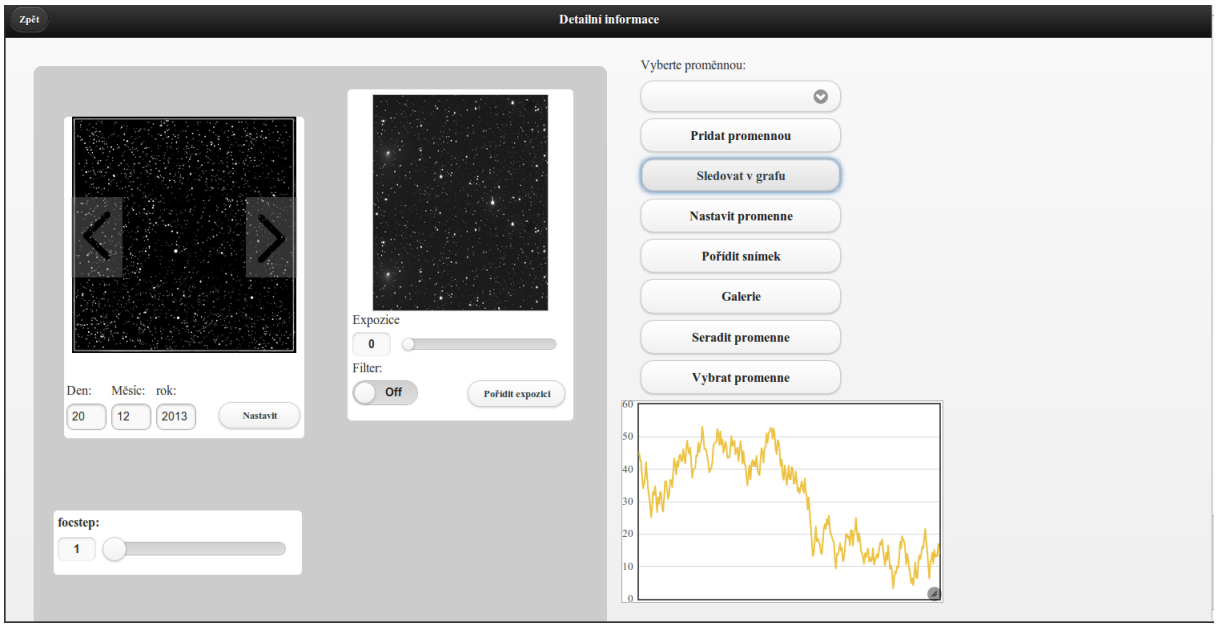
Aplikace bude komunikovat s dalekohledem CPC 925 GPS, který bude ovládán pomocí počítače systémem RTS2. Dalekohled podporuje standartní komunikační protokol NextStar. Komunikace mezi dalekohledem a řídícím počítačem probíhá přes sériovou sběrnici RS232.

Základní vlastnosti dalekohledu:

- Možnost vzdáleného ovládání
- Hledáček, který pomáhá přesně vyhledat pozici
- Ergonomický design umožňující snadnou manipulaci s dalekohledem
- Rozsáhlá databáze obsahující všechny známé objekty na obloze

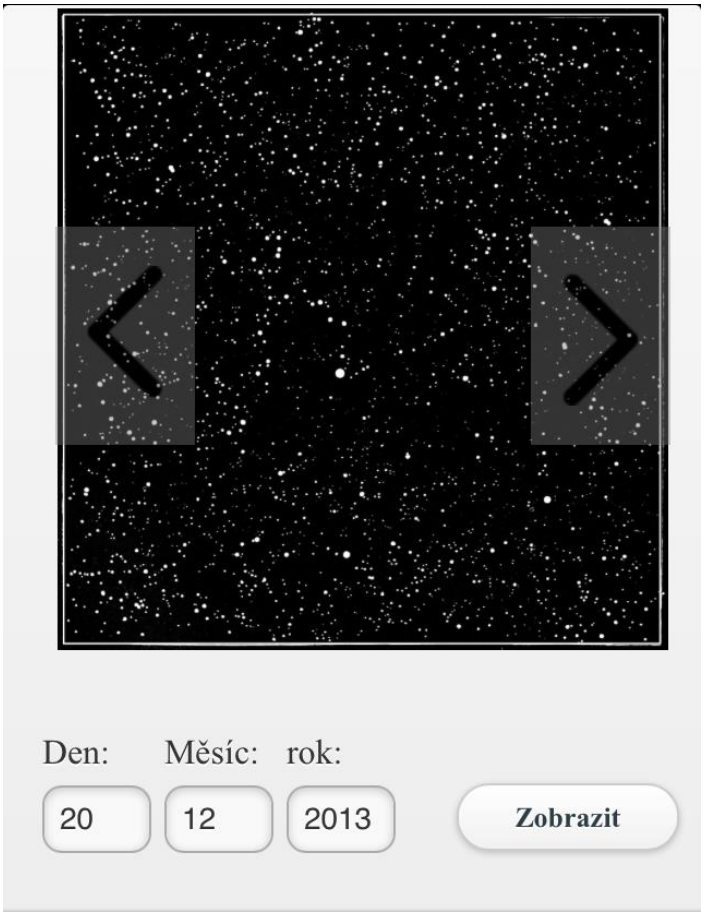
Obrázek 7 Dalekohled CPC 925

9 Příloha B: Obrázky aplikace



Obrázek 8 Aplikace na počítači

Obrázek 9 Menu nastavení na mobilním zařízení



Obrázek 10 Galerie na mobilním zařízeni

Obrázek 11 Menu pro pořízení snímku na mobilním zařízení