VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

PORTÁL PRO SLEDOVÁNÍ STAVU HERNÍCH SERVERŮ

GAME SERVER STATUS MONITORING PORTAL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **DOMINIK SKÁLA**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. RADEK BURGET, PhD.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Skála Dominik**
Program: Informační technologie
Název: **Portál pro sledování stavu herních serverů**
Game Server Status Monitoring Portal
Kategorie: Informační systémy

Zadání:

1. Seznamte se existujícími portály pro shromažďování údajů o herních serverech a jejich funkčností.
2. Prostudujte současné technologie pro tvorbu webových aplikací s veřejným aplikačním rozhraním.
3. Navrhněte architekturu portálu pro shromažďování a publikování informací o herních serverech různých kategorií, jejich stavu a statistikách provozu.
4. Navržené řešení implementujte pomocí vhodných technologií. Implementujte možnost vzdáleného přidávání serverů přes API, notifikace a možnost sdílení na sociálních sítích.
5. Proveďte testování výsledného řešení na přiměřeně rozsáhlé množině serverů.
6. Zhodnoťte dosažené výsledky.

Literatura:

- Swicegood, T.: Programming Node.js, O´REILLY, 2012
- Žára, O.: JavaScript - Programátorské techniky a webové technologie, Computer Press, 2015
- Gutmans, A., Rethans, D., Bakken, S.: Mistrovství v PHP 5, Computer Press, 2012

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Burget Radek, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 16. října 2018

Abstrakt

Cílem této práce je vyvinutí monitorovací aplikace herních serverů, která se zabývá sběrem a vizualizací monitorovaných dat. Práce nejprve pojednává o existujících řešeních, o jejich výhodách a nevýhodách. Na základě tohoto pozorování je tvořen návrh řešení, zejména návrh monitorovací aplikace a také návrh moderní webové aplikace s aplikačním rozhraním. Aplikační rozhraní je vyvinuto v jazyce PHP ve frameworku Yii. Samotné webové rozhraní je vyvinuto v JavaScriptové knihovně ReactJS. Nezávislou aplikací je monitorovací aplikace, která je také vytvořena v jazyce PHP. Na závěr práce je provedeno zhodnocení dosažených výsledků a efektivity monitorovací aplikace, z dosažených výsledků jsou také vyvozovány další varianty implementace monitoringu a dalších rozšíření aplikace.

Abstract

The main goal of this thesis is the development of a game server monitoring application which handles collecting and visualisation of collected data. The thesis compares and looks into existing solutions, their pros and cons. The solution is built upon this observation, specifically the design of the monitoring application and a modern web application with API access. The API is created using the Yii PHP Framework. The user interface is created using the ReactJS JavaScript library. The monitoring application itself is an independent application build in PHP. In the conclusion of the thesis, the evaluation of achieved results and the effectivity of the monitoring application is discussed. Based on the results, other implementations of monitoring applications and other extensions of web application are proposed.

Klíčová slova

Herní servery, monitoring, hosting, statistiky, vizualizace dat, sběr dat, API, PHP, JSON, ReactJS, JavaScript, HTML, SQL, DOM, interpretace dat

Keywords

Game servers, monitoring, hosting, statistics, data visualisation, data collection, API, PHP, JSON, ReactJS, JavaScript, HTML, SQL, DOM, data interpretation

Citace

SKÁLA, Dominik. *Portál pro sledování stavu herních serverů*. Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Radek Burget, PhD.

Portál pro sledování stavu herních serverů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Radka Burgeta, PhD. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal. Další informace mi poskytla společnost Hicoria Hosting s.r.o., která mi pro účely testování umožnila využít databázi svých herních serverů.

.....
Dominik Skála
12. května 2019

Poděkování

Rád bych poděkoval vedoucímu práce Ing. Radku Burgetovi, PhD. za odborné vedení a rady dodávané v průběhu studia. Také bych rád poděkoval společnosti Hicoria Hosting s.r.o., v jejíž spolupráci mi bylo umožněno vyvíjet aplikaci nejen s dostatkem prostředků, ale i s dostatkem potřebných dat.

Obsah

Kapitola 1

Úvod

Cílem této bakalářské práce je navrhnout a implementovat systém monitorování herních serverů. Tento systém je nutné implementovat pomocí moderních webových technologií využívajících moderních přístupů z pohledu uživatelských rozhraní. Do monitoringu herních služeb můžeme zařadit celou škálu informací, které lze sledovat, od doby odezvy serverů, přes momentální stav až po počet hráčů. Výčet možností sledování samozřejmě jde mnohem dále a pro každou aplikaci je povětšinou silně individuální. Valná většina aplikací se však drží určitého standardu, kterým je minimálně odezva a počet hráčů.

V dnešní době nalezneme na trhu spoustu podobných aplikací, které se zaměřují na monitorování z různých pohledů. Největší rozdělení je zde na Enterprise a „Home“ řešení. Enterprise již samy o sobě integrují větší množství služeb, které lze monitorovat.

Samotná práce je rozdělena do teoretické a praktické části. V teoretické části jsou popisovány aktuální varianty monitorování a jejich nedostatky, aktuální technologie které jsou vhodné pro použití při implementaci webových služeb a nakonec vybrané technologie, jejich popis a důvod proč byly využity zrovna ony. Praktická část je zaměřena spíše na návrh řešení, implementaci serverové a klientské části komunikující přes REST API, je v ní zahrnuto testování a postupy monitorování. V implementační části je shrnut popis využitých technologií (Server: Yii 2, Klient: ReactJs), jedna sekce je věnována problémům na které bylo naraženo při implementaci.

Kapitola 2

Monitorování služeb

Monitorování je podstatnou části našeho života, prakticky v každém odvětví našeho života se snažíme kontrolovat stav nějakého objektu. U počítačových systémů toto platí snad dvojnásob. Pokud budeme brát monitorování jako samostatnou činnost, je to odvětví, ve kterém bychom se jen v počítačových systémech ztratili. V dnešní době monitorujeme pro příklad stavy počítačových sítí, dostupnost samotných zařízení v internetu, ale můžeme i monitorovat samotná hardwarová zařízení a jejich stavy. Pro tyto účely můžeme zmínit například technologii S.M.A.R.T, která je využívána pro monitorování stavu pevných disků. Pokud se budeme bavit o počítačových sítích, můžeme zmínit postupy a služby jako SNMP, NetFlow. Dále pro příklad můžeme monitorovat jednotlivé probíhající procesy v operačních systémech. Nedílnou součástí je také aplikační monitoring, kterým můžeme sledovat nejen stav běžících procesů ale i informace o tomto procesu, které předává sama aplikace.

Důležitým faktorem je u monitoringu zpracování dat. Ve valné většině případů si společnosti vyvíjející monitorovací nástroje silně střeží způsob a formu implementace. Samozřejmě, základní nástroje pro sběr dat jsou většinu všeobecně známy a využívá se standardů jako SNMP nebo NetFlow o kterých je více popsáno v sekci 3.1, každopádně zpracování dat a operace nad nimi se snaží aplikace skrývat, aby nebylo možné jejich využití. Každá aplikace tak má své know-how nad kterým staví.

Kapitola 3

Stávající řešení

Stávající řešení na trhu můžeme rozdělit na několik segmentů. Těmi dvěma hlavními segmenty jsou soukromý a firemní sektor. Dále můžeme monitoring rozlišit na softwarový a hardwarový. Segment soukromý je převážně tvořen aplikacemi tvořenými na všeobecný způsob. Převážně se jedná o webové aplikace, které se specializují buď na jeden typ her a nebo všeobecně na více herních serverů. Jejich hlavní nevýhodou je převážně design. Z valné většiny se jedná o aplikace implementované ve starších jazycích (starší verze PHP), bez možnosti využití dynamičnosti moderních technologií. Nevyužívají tedy žádné JavaScriptové knihovny a dokonce po většinou ani žádných moderních UI postupů. Také se většinou jedná o čistě `server-server` řešení, kdy se server aktivně dotazuje na stav herních serverů bez jakékoliv samostatné aktivity herního serveru, server většinou poskytuje v rámci připojení nějaké hodnoty.

Pokud se bavíme o enterprise řešení, tyto řešení jsou po většinou řešeny na bázi internetových protokolů. Na těchto protokolech dále staví externí aplikace. Mezi základní protokoly lze zařadit SNMP a nebo NetFlow. Nad těmito protokoly případně nad jejich postupy staví externí aplikace. Mezi nejznámější aplikace patří Zabbix, ntopng a nebo NagiOS.

3.1 Monitorovací protokoly a aplikace

3.1.1 SNMP

SNMP je protokol pro práci s monitorovanými objekty, to znamená že provádí kontrolu provozu přímo na daném zařízení. Samotná definice je popsána v RFC 1157. Jak popisuje Dr. Matoušek, samotný protokol se vyvinul z protokolu SGMP[1]. [9]. Rozlišujeme u něj řídící stanici a agenta na monitorovaném zařízení. Komunikace probíhá ve valné většině případů na bázi dotazů ze strany serveru ve formě: **get** a **set**, pomocí kterých dochází ke kontrole a nastavování prvků na monitorovaném zařízení. Samozřejmě je zde i možnost komunikace přímo od klienta, těmto zprávám se říká *nevyžádané zprávy*. Jedná se o speciální typ **trap**. Tyto zprávy jsou opravdu využívány výjimečně, například k ohlášení pádů rozhraní, zaplnění front, atd.

[1] SMGP – Simple Gateway Management Protocol

3.1.2 NetFlow

NetFlow je dalším z řady protokolů sloužících pro monitorování, ovšem v tomto případě kontroluje zejména provoz na síťových prvcích. Protokol byl vyvinut společností Cisco[2], je neustále vyvíjen a dnes je již dostupný v 10. verzi. Za zmínku určitě stojí verze číslo 5 a číslo 9, které jsou jedny z nejpoužívanějších.[3]

NetFlow má hlavní 4 prvky – Exportér a Kolektor, nástroje pro export a komunikační protokol. Exportér slouží pro získávání dat o tocích, naopak kolektor slouží pro ukládání dat na síťovém prvku přes který protéká provoz. Komunikační protokol je vždy nějaká verze protokolu NetFlow. Jak jsem již zmínil výše, běžně a předně se můžeme setkat s verzemi 5 a 9, ale také s IPFIX[3] a nebo sFlow[4]. Nástroje pro export jsou již spíše dodatečným avšak neméně důležitým prvkem. Mezi takové nástroje můžeme zařadit nástroj NfSen[5].

3.1.3 Zabbix

Zabbix je klient-server aplikace, vytvořená Alexeievem Vladishevem a aktuálně je vyvíjená organizací Zabbix SIA. Jeho možnosti jsou prakticky neomezené, dokáže monitorovat téměř cokoliv od počtu procesů na procesoru, přes aktuální využití sítě až po počet IO operací na discích. Také zvládne monitorovat počet stávajících připojení. Tím ale výpis nekončí. Pokud aplikace nedostačuje, je ji možné rozšířit vlastními skripty nebo programy. A co více, aplikace je kompletně zdarma, náklady na provoz jsou tedy pouze za zařízení na kterém tato aplikace běží.[20]

Přidání je značně jednoduché – na koncovém zařízení stačí nainstalovat aplikaci *zabbix-agent*. Tato aplikace vygeneruje vlastní konfigurační soubor, ve kterém se vyplní server na kterém má probíhat sběr.

Jak již bylo zmíněno, monitorovat lze prakticky cokoliv. Jedním z důležitých prvků je síťové rozhraní, tam lze sledovat nejen stav sítě, ale i její rychlost, aktuální využití rychlosti, jak je vidno na obrázku 3.1.

3.1.4 NagiOS

NagiOS je opět open-source aplikací vyvíjenou v jazyce C poprvé vydanou v roce 1999. Od té doby se aplikace rozrostla a zaintegrovala spoustu veřejně dostupných systémů a aplikací. V dnešní době existuje několik variant, které nabízí různorodé možnosti od monitorování sítí až po kontrolu a částečnou správu aplikací služeb a dokonce i zařízení. Obrovskou výhodou je také možnost instalace vlastních rozšíření.

NagiOS obsahuje jak klientskou tak i serverovou aplikaci, přičemž server se může chovat i jako klient a měřit na nainstalovaném zařízení potřebné hodnoty. Výhodou je také možnost instalovat pluginy nejen na serverové instanci, ale také na klientech.

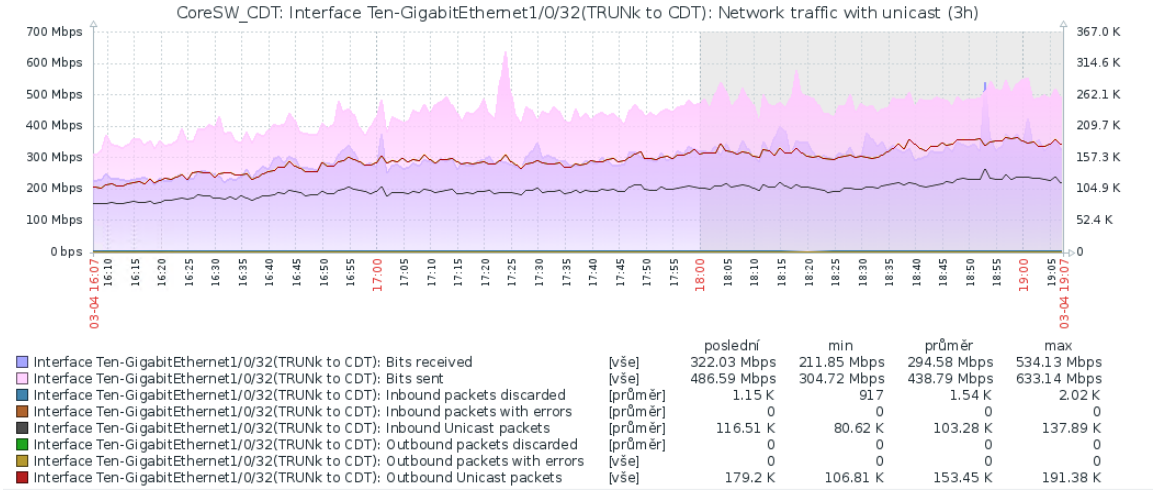
Mezi sledovatelná zařízení lze zařadit stroje s operačním systémem Windows, stroje s operačním systémem založeném na Linux/Unix. Dále zvládá monitorovat routery a switche, samotné služby běžící na koncových zařízeních (kupříkladu HTTP nebo SSH) a v neposlední řadě sem můžeme zařadit monitoring síťových tiskáren.[11]

[2] Cisco Systems, Inc. – gigant na poli síťových prvků, dnes prakticky udávající trendy ve světě internetových protokolů a hardwaru

[3] IPFIX – rozšíření 9. verze NetFlow protokolu

[4] sFlow – https://sflow.org/sflow_version_5.txt

[5] NfSen – http://nfsen.sourceforge.net/



Obrázek 3.1: Využití sítě na páteřním switchi společnosti Hicoria Hosting s.r.o.

3.1.5 ntopng

ntopng je monitorovací aplikací sledující provoz sítě. Je založena na knihovně libpcap a její použití je možné na obrovské škále zařízení sahajících od Unix zařízení, přes MacOS až po Windows servery. Mezi hlavní vlastnosti této aplikace můžeme zařadit řazení síťového provozu, zjišťování nejvíce vytížených uzlů, geolokace hostů, kompletní analýzu IP provozu. Bonusem je také skvělá podpora MySQL sloužící k exportu všech potřebných dat k možné vizualizaci v nějakém informačním systému. Samotná aplikace má několik verzí, kde komunitní verze je zdarma, Pro a Enterprise verze jsou již zpoplatněny dle licenčních podmínek.[14]

3.2 Dostupné služby

Na trhu je spousta dostupných produktů a není určitě lehké jim konkurovat. Cílem této práce je však nejen jim konkurovat ale vyvinout software, který překoná takovéto aplikace. Mezi nejznámější aplikace na trhu lze zařadit spoustu server-listů. Po důkladném zkoumání jsem se rozhodl zařadit tyto 4 následující: Minecraft-Server-List.cz, službu Listforge, Czechcraft a Gametracker. Každá tato služba působí na trochu jiném trhu, v následující části popisuji jejich výhody, nevýhody a co si lze z jejich designu odnést.

3.2.1 Minecraft-Server-List.cz

Tato webová stránka je českým listem serverů pro hru Minecraft. Jak je dle obrázku 3.2 patrné, nezakládá si příliš na vzhledu webu a o nějaké optimalizaci pro telefony si na tomto webu můžeme nechat zdát. Hlavní výhodou jsou naopak možnosti komentářů případných hráčů a nějaká základní filtrace. Komentáře ale nejsou nijak validovány ani zde nejsou žádní hlavní recenzenti, což věrohodnosti komentářů příliš nepřidává a komentář tak může přidat kdokoliv, kdo se zaregistruje.

3.2.2 Služba Listforge

Tato služba sdružuje několik různých server-listů do jednoho, přičemž každý běží na zvláštní doméně. Prakticky se jedná vždy o stejný design s jinou infografikou, tj. s jiným obrázkem v záhlaví a s jinou primární barvou webu.

Důležitým prvkem je výpis statistik počtu hráčů u jednotlivých serverů. Webové stránky navíc nejsou pouze pro vlastníky serverů, ale i pro hráče, hráč si zde může přidat svůj server do oblíbených, může si prohlédnout seznam nejhranějších serverů a co je asi nejdůležitější, přímo ovlivňuje pozici serveru ve výpisu serverů formou hlasování. Pro každý server lze hlasovat přes speciální URL kód, který si mohou majitelé vložit na svůj web a tím tak dále propagovat webové stránky tohoto listu. Navíc je ve statistikách určitá forma porovnání. Co zde však rozhodně chybí je forma propojení se sociálními sítěmi, na které je v dnešní době kladen velký důraz. Nevýhodou také je zobrazení informací pomocí tabulek v detailu serveru. Další obrovskou nevýhodou je nejednotnost designu – webové stránky jsou sice rozkopírovány, ale aktualizace očividně nejsou přenášeny mezi všemi stránkami = neudržitelnost kódu. Jak je vidět na obrázku 3.3, na jedné variantě existuje slušná mobilní responzivita, na druhé si o tom opět můžeme nechat jen zdát.

3.2.3 Czech-craft

Aplikace Czech-craft obsahuje souhrn serverů hry Minecraft. Na webové stránce je již poznat její stáří, nevyužívá moderních prvků a už vůbec se nevěnuje mobilní responzivitě. Je pravda, že Minecraft servery jsou pouze pro počítačovou verzi této hry a tak by se dalo argumentovat zda je responzivitě opravdu nutná. Je to však dnes opravdu standard a všeobecný trend je přesouvání webů na mobilní zařízení. Hlavní výhodou je zde propojení se sociálními sítěmi a to ve formě komentářů na Facebooku. Každý server má vlastní stránku na které má nějaký popis a banner[6]. Hráči také mohou ovlivňovat chod serveru hlasováním a zvyšováním tak jeho popularity.

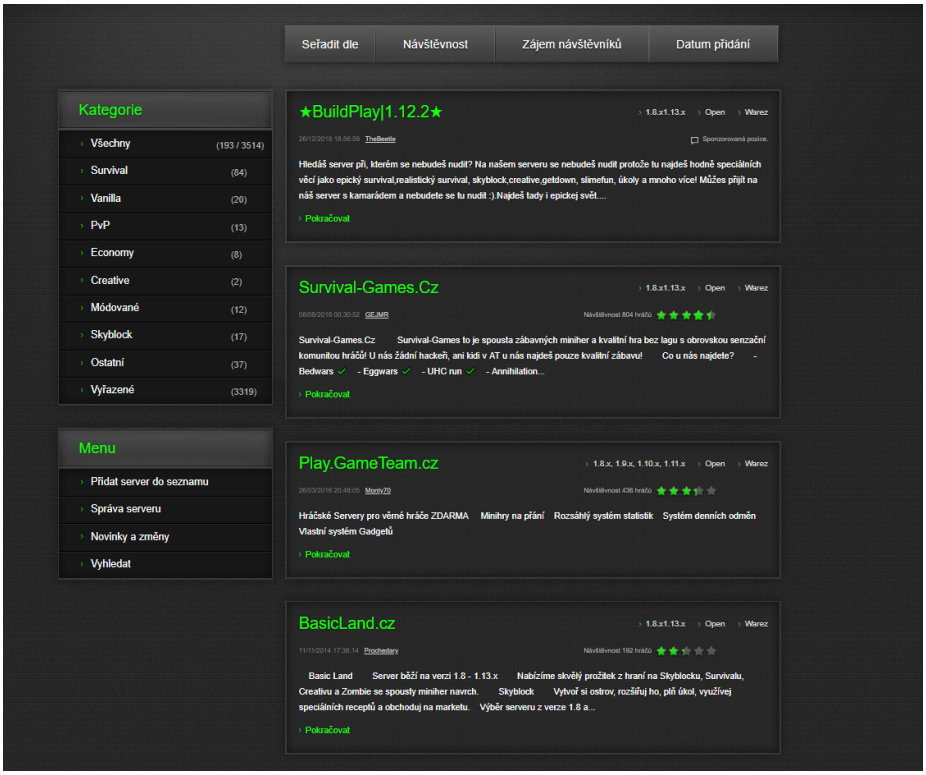
3.2.4 Gametracker

Gametracker bychom mohli označit za praotce herních server-listů. O tom i svědčí jeho design – opakované použití tabulek, tmavé barvy, neresponzivní web. Co je nutné zmínit je implementace nových služeb, které monitorují a udržení podpory pro starší služby, které už dávno nejsou vyvíjeny. Každý server má blog, kde každý uživatel má nějaký profil a uživatel si tak jednoduše může zkontrolovat zda se nejedná o falešné hodnocení serveru.

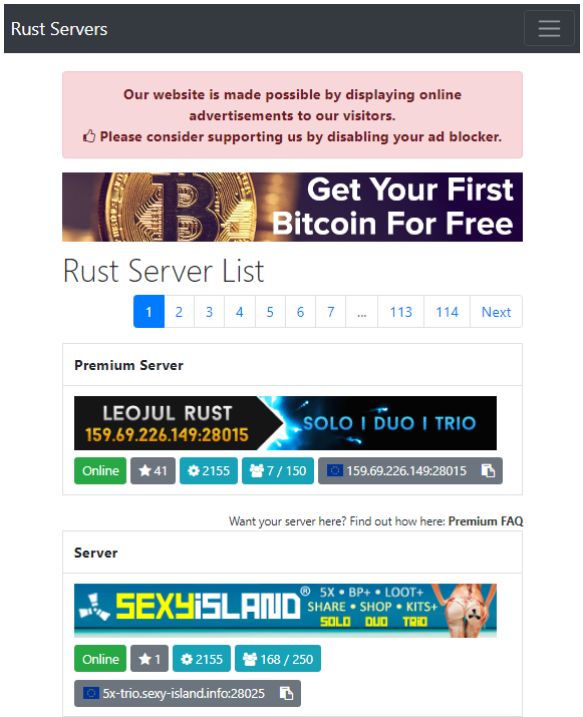
3.2.5 Zhodnocení

Důležitými prvky jsou dnes zejména sociální sítě, jak již vyplývá ze zadání této práce. Ty však ve valné většině dostupných nabídek chybí. Veškeré tyto stránky se zaměřují na svůj rozvoj a ne na rozvoj klientských serverů. Důležitými faktory jsou také responzibilita webu a moderní design. Žádný z výše zmíněných webů nenabízí unifikované řešení pro registrátory a skoro nikdo nenabízí unifikovaný systém pro získávání stavu serverů krom nutnosti využít reklamu na samotný web.

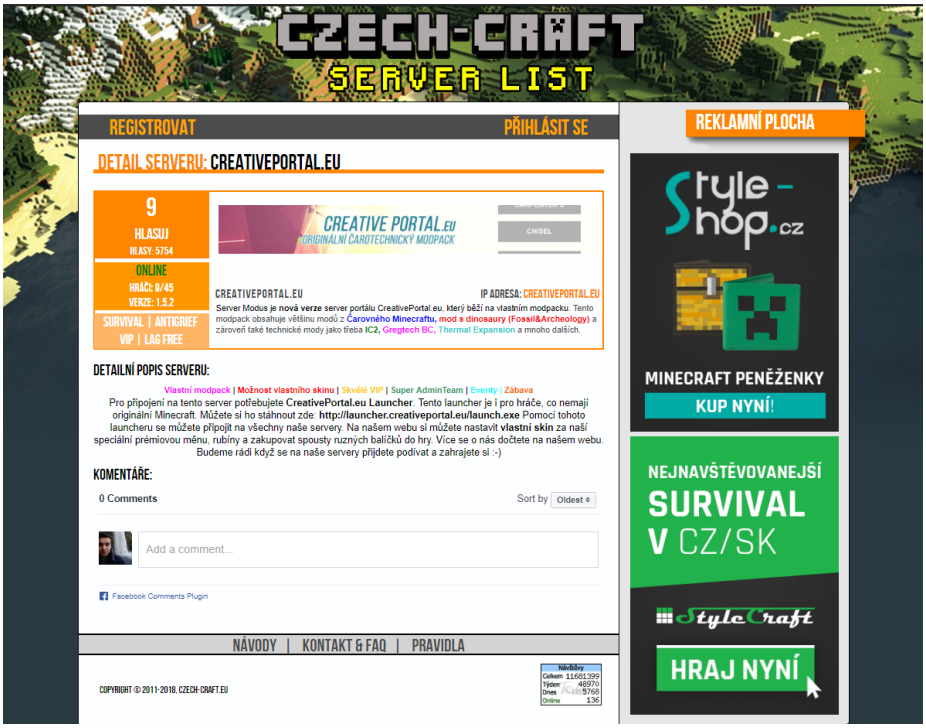
[6] Banner – reklamní obrázek/ikona obsahující data a informace o serveru



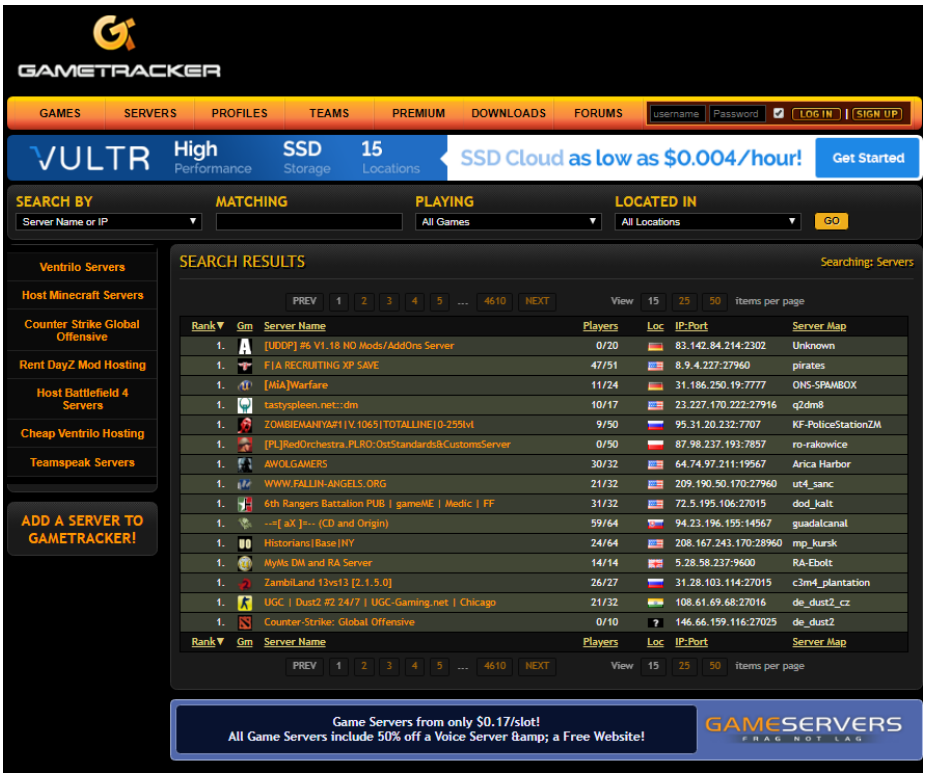
Obrázek 3.2: Screen webového rozhraní aplikace Minecraft-Server-List.cz



Obrázek 3.3: Mobilní webové rozhraní aplikace Rust-servers.net



Obrázek 3.4: Detail serveru na webu Czech-craft



Obrázek 3.5: Webové stránky Gametracker.com

Kapitola 4

Současné technologie

Mezi současné technologie sloužícím k tvorbě webových aplikací patří nespočet jazyků, metod a aplikací. V následující kapitole rozebírám jednotlivé jazyky, jejich historii a klady a zápory pro použití ve vyvíjené aplikaci. Také procházím aplikace pro tvorbu a správu relačních databází a postupy pro tvorbu nejen klientských ale i serverových částí aplikace.

4.1 HTML

HTML (zkratka pro Hypertext Markup Language) je značkovací jazyk určený a vyvinutý pro tvorbu webových stránek. Slouží pro vykreslování obsahu vytvářeného v souborech s příponou .html nebo .htm. Jeho použití je dnes většinou doprovázeno dalšími technologiemi, ať už se jedná o klientské prvky rozšiřující dynamičnost webových stránek a nebo naopak serverové prvky dovolující interakci a správu stavů.

Historie HTML

Jazyk HTML byl vyvinut na počátku 90. let minulého století Timem Berners-Leem ve spolupráci s Danielem Connollym. Primárním úkolem bylo vytvořit jednoduchý systém sdílení dokumentů mezi jednotlivými zaměstnanci CERNu. V souvislosti s tímto jazykem vytvořil Tim Berners-Lee první prohlížeč s názvem WorldWideWeb. V brzké době na tento trend začaly navazovat další společnosti, po vytvoření webových stránek CERNu došlo k rychlému rozvoji webu a bylo nutné začít definovat standardy jazyka HTML. Původní verze jazyka byla čistě textová, nepodporovala grafické rozhraní. Verze 2.0 se již dočkala podpory grafiky a interaktivních formulářů. Vydána byla v roce 1995 komunitou IETF[1]. V této době postupně vzniká W3C[2], které se do budoucna stará o standardizaci tohoto jazyka. V roce 1997 vyšla verze 3.2 doplňující HTML jazyk o tabulky, zarovnáním textů a stylové prvky více popsané v sekci 4.2.1. na konci 90. let dochází k vydání na dlouhou dobu jedné z posledních verzi HTML – v.4.01. Mělo se jednat o poslední verzi jazyka, která měla být nahrazena jazykem XHTML využívajícím prvky XML. Hlavním bodem nebo snad výhodou chcete-li mělo u jazyka XHTML být „Drakonické trestání chyb“, jak říká Pilgrim[19]. Ve výsledku to znamená, že se každá chyba ve struktuře XHTML zdrojového kódu (např. neuzavřený tag, prolínání dvou tagů) měla projevit jako kritická chyba. Pilgrim také uvádí, že dle některých statistik obsahuje až 99 % webových stránek nějakou chybu ve zdrojovém kódu a ačkoliv

[1] IETF – Internet Engineering Task Force = organizace zabývající se standardy TCP/IP a internetovými protokoly

[2] W3C – World Wide Web Consorcium = organizace standardizující jazyk HTML

si s ní většina prohlížečů umí poradit, jsou tyto chyby nežádoucí. Celkově se však XHTML neuchytilo zejména u tvůrců webových prohlížečů, kteří si založili iniciativu WHATWG. Ta si dala za úkol vyvinout novou verzi HTML, která by byla schválena W3C. Po dlouhých 15 letech od verze 4 (rok vydání – 1997) dochází k vydání verze HTML5, která přináší novinky jako přehrávání multimédií přímo v prohlížeči pomocí speciálních tagů (není nutné využít rozhraní 3. stran) nebo podpora offline webových aplikací.

4.2 Stylování a design

4.2.1 CSS

Kaskádové styly (CSS[3]) slouží pro popis způsobu zobrazování prvků v HTML souborech. Úzce tedy spolupracují s HTML jazykem. Hlavním smyslem tohoto jazyka je oddělit vykreslovaná data od zápisu jejich vzhledu. Dřívější verze jazyka HTML obsahovaly spoustu elementů mající vlastní design. To bylo však z hlediska tohoto přístupu nežádoucí. Největší nevýhodou stylů je jejich interpretace (nebo dezinterpretace) mezi různými prohlížeči. Každý prohlížeč kaskádové styly interpretuje a potažmo i vykresluje trochu jinak. Silně tedy závisí na implementaci v jádru daného prohlížeče. Ukázkovým příkladem různých interpretací mohou být prohlížeče Chrome a Edge.

Historie CSS

Postupný vývoj CSS začal v prosinci roku 1996 kdy vyšla první verze tohoto jazyka. Jak již bylo zmíněno výše, jeho hlavní snahou bylo oddělit implementaci vzhledu od elementů HTML a dát tak kontrolu nad těmito prvky vývojářům. Tvůrcem tohoto jazyka je norský zaměstnanec W3C – Håkon Wium Lie. V květnu roku 1998 vyšla CSS level 2, která rozšířila původní standard z roku 1996 o absolutní pozicování, automatické číslování nebo například o práci s textem psaným zprava doleva. Dále následovaly verze 2, 2.1 až aktuální verze 3. Ta vyšla v roce 2001, přinesla spoustu novinek jako jsou transformace, přechody nebo animace. V dnešní době se již hojně pracuje na verzi CSS4.

4.2.2 CSS preprocesory

Vylepšením jazyka CSS je kompilovaný CSS jazyk neboli preprocesor CSS, který umožňuje využívat a pracovat s proměnnými. Zjednodušuje implementaci stylů využíváním tzv. zanořování stylů. Existuje mnoho implementací a druhů těchto preprocesorů, za zmínku stojí SASS, LESS. Existují ale i nadstavby nad jednotlivými preprocesory např. extra výkonný kompilátor libSass, který je vyvíjen v jazyce C. Dle dosavadních ohlasů je až 10x rychlejší než původní SASS. Pro bližší popis se budeme věnovat SASS preprocesoru. Důležitým prvkem v SASS je fakt, že jakýkoliv starší CSS soubor je stále validním SASS souborem. Má dvě varianty zápisu – starší a novější. Starší bychom mohli připodobnit k zápisu jazyka Python, tedy zanořené prvky musí být korektně odsazeny – ovšem s hlavními rozdíly: proměnné jsou uvozeny znakem `!`, přiřazení je prováděno pomocí rovnítka namísto dvojtečky. Tyto elementy jsou právě nahrazeny v novějším zápisu, kde jsou jednotlivé prvky zapsány pomocí složených závorek, podobají se tak zápisu JSON objektů, čímž usnadňují orientaci ve zdrojovém kódu. Souhrnem tedy preprocesory umožňují zjednodušit tvorbu stylů prvků HTML, optimalizují tyto soubory stylů, přičemž mohou tedy snížit jejich velikost a zrychlit

[3] CSS – Cascading Style Sheets

jejich načítání, také umožňují kód zpřehlednit použitím proměnných nebo mixinů, které jsou parametrizované bloky kódu pro tvorbu SASS stylů.

```
table.hl {
  margin: 2em 0;
  td.ln {
    text-align: right;
  }
}

li {
  font: {
    family: serif;
    weight: bold;
    size: 1.3em;
  }
}
```

Výpis 4.1: Ukázka zanořování v SASS

4.2.3 Bootstrap

Bootstrap je nadstavbou CSS preprocesorů. Původně se jednalo o projekt vyvíjený designery a vývojáři ve společnosti Twitter, ti se však rozhodli celý tento framework poskytnout veřejnosti, která jej s radostí začala využívat. Dnes již existuje 4. verze tohoto frameworku, využívající sass preprocesoru a novinky v CSS, tzv. flexboxů. Bootstrap je vysoce nápomocným nástrojem pro tvorbu jednotného webového rozhraní s cílem na mobilní responzivitu. V dřívějších verzích nebyla responzivita prioritou, od 3. verze se však pohled změnil a prioritně se cílí na mobilní zařízení, očividně z důvodu že počet mobilních zařízení rok od roku stoupá na úkor počtu zakoupených plnohodnotných počítačů. Mezi přednosti a hlavní prvky Bootstrapu můžeme vyznačit dlaždicové rozložení, pomocí kterého využívá k pozicování prvků na webech. Dále nabízí jednoduchou možnost editace témat, čímž umožňuje vývojářům pomocí pár proměnných vytvořit prakticky nový web bez nutnosti přílišných zásahů.

4.2.4 Material Design

Material Design je formát vyvíjený společností Google. Staví opět na principech Bootstrapu, tedy na dlaždicovém rozložení stránky, umožňuje ale využívat animací, přechodů, odsazení a také stínování efektů. Google také nabízí spoustu variant této implementace ve formě knihoven dostupných pro různé jazyky a shodou náhod tuto knihovnu nabízí i pro React. Samotné rozložení bylo ze strany Googlu od roku 2015 implementováno prakticky do všech aplikací této společnosti, včetně i webového prohlížeče Chrome. Výhodou je opět cílená mobilní responzivita, jednoduché a standardizované ovládací prvky, které lze využít napříč různými zařízeními bez obtíží.

4.3 Klientské skriptování

4.3.1 JavaScript

JavaScript je programovací jazyk určený pro tvorbu dynamických HTML stránek na straně klienta. V dnešní době se však můžeme potkat i s JavaScriptem překládaným na straně serveru, o tom ale však v další sekci. JavaScript lze označit za nenáročný, objektově orientovaný, beztřídní jazyk. Syntaxí se tento jazyk silně podobá jazykům C/C++, avšak přejímá i jisté podobnosti z Javy. Jazyk je také beztypový a hlavně multiplatformní. Jeho největším účelem je tedy dynamické načítání a manipulace s obsahem webových stránek. Pro tyto účely používá DOM[4] se kterým umí dále pracovat. Pro dnešní účely se také silně využívá minifikace nebo „uglifikace“. Minifikace je proces zhuštění JavaScriptových souborů do menších, očištěných o nepotřebné mezery, zkrácené názvy proměnných, atd. Proces „uglifikace“ nemá v českém slovníku přesný překlad. Nejlépe ho vystihuje pojem: znečitelnění. Při tomto procesu dochází k přejmenování proměnných a funkcí za účelem zhoršení čitelnosti. Tohoto procesu se využívá u JavaScriptu zpracovávaného na straně klienta. Ve výpisu 4.2 je neminifikovaná verze a poté ve výpisu 4.3 vidíme verzi znečitelněnou.

```
var x = {
    baz_: 0,
    foo_: 1,
    calc: function() {
        return this.foo_ + this.baz_;
    }
};
x.bar_ = 2;
x["baz_"] = 3;
console.log(x.calc());
```

Výpis 4.2: Nezměněný zdrojový kód

```
var x={o:0,_:1,l:function(){return this._+this.o}};
x.t=2,x.o=3,console.log(x.l());
```

Výpis 4.3: Modifikovaný, znečitelněný zdrojový kód

4.3.2 Historie Javascriptu

Základy tohoto jazyka sahají až do roku 1995, kdy Brendan Eich (později zaměstnanec Netscape Corporation) během velmi krátké doby navrhl jazyk Mocha. Tento jazyk měl konkurovat jazyku PHP a Javě. Později přejmenován na LiveScript a implementován v prohlížeči Netscape Navigator 2.0 si JavaScript získal rychle na oblíbenosti a světe div se, společnost Microsoft nelenila a vyvinula vlastní verzi pro Internet Explorer 3.0 pod názvem JScript. Na konci roku 1996 začala společnost Netscape spolupracovat s organizací ECMA[5] za účelem standardizace. Dalo by se říci, že byl JavaScript rozdělen na dvě větve. Jednou byl ECMAScript, což je základní standardizovaná verze tohoto jazyka a samotný JavaScript,

[4] DOM – Document Object Model = soubor HTML prvků webu přetransofrmovaných do objektů

[5] ECMA – European Computer Manufacturers Association = společnost zodpovědná za standardizaci nejen JavaScriptu, ale i třeba formátů dat na CD či jazyka C#

který je nadstavbou nad tímto jazykem. Od roku 1996 je tedy hlavním tvůrcem tohoto jazyka společnost ECMA International, vyvíjející a rozšiřující standard ECMA. Jak zmiňuje Žára, v roce 1999 vyšla na dlouhou dobu poslední verze tohoto jazyka.[21] Po dlouhých 9 letech se v roce 2008 ze strany společnosti Googlu objevuje novodobý prohlížeč Chrome, pro své účely obdrží vlastní výkonnou implementaci JavaScriptu, ta má pracovní název V8. V letech 2009 vzniká organizace CommonJS, jejímž účelem je odhalit možná JavaScriptová serverová rozhraní. Ve stejném roce vzniká Node.js[6]. Myšlenka čistě klientského JavaScriptu byla tak po dlouhé době překonána. Na konci roku 2009 vychází nová verze ECMAScriptu – ES5. Dnes je již standardizována devátá verze označována jako ES2018.

4.3.3 jQuery

jQuery je JavaScriptovou knihovnou určenou pro zjednodušení práce s DOM stromem. Hlavním úkolem je co nejsnadnější implementace JavaScriptu do webové stránky. Značně zjednodušuje syntax JavaScriptu a co více, umožňuje DOM elementy libovolně modifikovat. Hlavními přednostmi jsou tedy práce s DOM elementy, zpracování události z prohlížeče ale i a to zejména ajax[7] volání, která umožňují dynamicky načítat další zdroje bez nutnosti znovu načtení dané stránky. Je multiplatformní a rychle rozšiřitelné. Jediné co je dnes nutné je vložit zdrojový soubor do HTML souboru a jQuery knihovna bude automaticky stažena a načtena.

4.3.4 Angular

Angular je opět nadstavbou JavaScriptu. Jedná se o multiplatformní nástroj a je označován za JavaScript MVC[8] framework, který poskytuje strukturované metody tvorby webových stránek a webových aplikací. Je postaven na odlehčené verzi jQuery, zpracovává se na straně klienta a poskytuje relativně jednoduché prostředí pro vývoj aplikací. V dnešní době umožňuje i psaní formou TypeScriptu[9] čímž se z něj stává opravdu silný nástroj.[2]

4.3.5 React

Knihovna z dílny společnosti Facebook vyvinutá za účelem překonání problémů s obrovskými webovými aplikacemi, které jsou určeny pro správu dat. React byl vydán v roce 2013 a na první pohled se mohl každému zdát jako příliš šílený. Jak píší Banks a Porcello, po vydání této knihovny byl od Facebooku napsán článek: „Why React?“ (přeloženo: „Proč React?“), ve kterém vývojáři vysvětlovali, že je nutné Reactu dát alespoň 5 minut před úsudkem, že je přístup Reactu šílený.[1] Jeho hlavní doménou je vytvoření tzv. Virtual-DOM, což je **virtuální strom objektů HTML**. Tím že si vytváří tento virtuální strom je React schopen nad takovýmto stromem pracovat mnohem rychleji než nad standardním stromem HTML objektů. Dalším význačným prvkem Reactu je fakt, že je valná většina funkcionalit řešena externími knihovnami, které se často aktualizují, mění se, jejich vývoj může být ukončen a mohou vzniknout naopak nové. To je pro někoho silnou nevýhodou, protože se pak React nemůže označovat za framework a nenabízí žádné standardizované řešení. Opak je ale pravdou, aplikace jsou pak tvořeny na míru a web používá pouze balíčky,

[6] Node.js – sada rozhraní a knihoven umožňující spouštět JS i mimo rozhraní prohlížeče
[7] Ajax – Asynchronous JavaScript and XML
[8] MVC – Model View Controller = přístup pro programování OOP aplikací
[9] TypeScript – programovací jazyk vyvinutý společností Microsoft, silně se inspiruje C# a Javou

které jsou opravdu potřebné. Není tak nutné využívat hromadu kódu, jako je tomu u PHP frameworků, které dodávají ucelené řešení.

4.3.6 Node.js

Pro účely Reactu je nutné mít nainstalován Node.js, což je prostředí pro tvorbu JavaScriptových aplikací bez nutnosti použití prohlížeče. Přirovnat jej můžeme k fungování JVM. Použití Node.js má mnoho výhod, jednou z mnoha může být ošetření uváznutí, protože zde nejsou žádné zámky a téměř žádné funkce nečekají na hardwarová zařízení, nad kterými by se proces musel zastavit.[13] Samotný Node.js není úplně nutný pro Reactovou aplikaci, avšak pro rozšíření funkcionality je nutný **NPM**[10], což je balíčkovací nástroj sloužící nejen pro instalaci ale i údržbu JavaScriptových balíčků. Balíčky jsou uživatelské knihovny, které nejsou nijak standardizovány. Takové knihovny je také možné po instalaci rozšiřovat do vlastních balíčků a měnit tak jejich funkcionalitu.

4.4 REST API

4.4.1 Java

Java je moderním programovacím jazykem původně vyvíjeným společností Sun Microsystems původně pod názvem Oak. Jak píše Herout ve své knize, přejmenování došlo po zjištění, že existuje již jiný programovací jazyk se stejným jménem jménem. Kvůli tomuto došlo k přejmenování na jazyk Java, což mělo referencovat slovo „kafe“ z amerického slangu, jak píše Herout.[5] Dříve vyvíjená společností Sun Mirosystems, dnes již pod křídly Oracle.

Její přednosti jsou zejména v možnosti vývoje full-stack aplikací [11] a také webových aplikací. Obrovské webové aplikace jsou dnes poháněny frameworky postavených ať už na Pythonu a nebo právě Javě, protože dokáže mnohem efektivněji využívat prostředky systému. Java je však určena spíše náročnějším projektům do kterých tato práce rozhodně nespadá. Důležitým faktorem je interpretace kódu v JVM [12], což znamená, že se při kompilaci jazyka nevytváří žádné binární spustitelné soubory, nýbrž jazyk je interpretován právě ve formě mezikódu, do kterého je přeložen překladačem.

4.4.2 Node.js

Node.js byl sice zmíněn v sekci klientského skriptování 4.3, každopádně využít jej lze i pro implementaci serverového přístupového rozhraní. To je právě dáno možností překládání na straně serveru, jak bylo zmíněno v sekci 4.3.6. Výhodou tvorby vlastního aplikační rozhraní právě v Node.js je jeho jednoduchost, možnost instalace dodatečných balíčků a zejména optimalizace. Aplikace je vytvořena na míru potřebám programátora, nevyžaduje žádné externí zdrojové kódy a nenabízí nic, co programátor nepotřebuje, čímž snižuje nejen nároky na paměť ale i výpočetní nároky.

4.4.3 PHP

PHP (PHP: Hypertext Preprocessor) je programovací jazyk využívaný hojně u webových technologií v širokém měřítku. Jedná se o aktivně vyvíjený jazyk, nad kterým je posta-

[10] NPM – Node Package Manager
[11] full-stack – jedna aplikace mající jak front-endovou tak i back-endovou část
[12] JVM – Java Virtual Machine = virtuální stroj pro interpretaci zdrojového kódu

vena spousta aplikací. Takovýmto webovým aplikacím/knihovnám se přezdívá `Frameworky`. O těch ale až v další kapitole. PHP samotné je mocným programovacím jazykem postavený nad syntaxí jazyka C. Ve spolupráci s webovým serverem dokáže provádět dynamické vykreslování HTML souborů. Pracuje samozřejmě na straně serveru a již vykreslený obsah předává klientovi.[18] Výhodou PHP je i možnost tvorby konzolových aplikací. Nejsilnější stránkou tohoto jazyka je jednoduchá přenositelnost zdrojových souborů a obrovská rozšiřitelnost různými knihovnami. Právě nad tímto staví své chování PHP frameworky.

4.4.4 Historie PHP

PHP bylo vytvořeno Rasmusem Lerdorfem, původně pouze pro jeho osobní účely – proto název Personal Home Page Tools, což se zkráceně zapisovalo jako PHP Tools. Účelem těchto nástrojů bylo sledování návštěv na jeho online životopis. Postupem času se však rozhodl aplikaci dále a dále rozšiřovat, přibyly možnosti připojení k databázovým serverům, což zapůsobilo na spoustu vývojářů, kteří pomocí tohoto nástroje mohli vyvíjet malé dynamické aplikace.

V roce 1995 se rozhodl Rasmus učinit zdrojové kódy veřejnými, čímž se z PHP stal komunitní jazyk, do kterého mohl kdokoliv přispívat a opravovat v něm chyby. Postupem času se z PHP stal jazyk vhodný pro tvorbu formulářů, označovaný jako FI, podporoval také HTML syntax a silně se podobal jazyku Perl. Na jaře roku 1996 došlo ke kompletnímu přepsání tohoto jazyka na plnohodnotnou sadu nástrojů umožňujících komunikaci s databázemi, tvorbu formulářů, práci s cookies, také přibyla podpora uživatelsky definovaných funkcí.

Postupem času se vyvíjely další a další verze, za zmínku stojí PHP 3, které již připomíná PHP tak jak jej známe nyní s podporou objektově orientovaných jazyků. Dále určitě stojí za zmínku verze 5, která se tu udržela po dlouhých 15 let, než ji překonala verze 7. [17]

4.4.5 PHP Frameworky

PHP Frameworky jsou podstatnou částí vývoje webových stránek. Jejich použití umožňuje vývojářům standardizovaným způsobem vytvářet webová rozhraní. Framework je v podstatě soubor knihoven, které spolu nějakým způsobem interagují a vytvářejí jednu aplikaci. Nabízí základní sadu objektů, nad kterými lze webové rozhraní postavit. Mezi nejznámější PHP Frameworky patří Laravel, Nette, Yii a další. Valná většina těchto frameworků využívá architektury MVC – model, view, controller.

Laravel

Laravel je open source PHP framework pro webové aplikace. První verze tohoto frameworku se datuje k únoru 2012. Využívá softwarové architektury MVC, je uzpůsobeným pro tvorbu velkých webových aplikací, usnadňuje práci při tvorbě autentizace, směrování, sezeních a cachování dat. Cílem vývojářů bylo vytvořit framework tak, aby bylo jeho použití nanejvíc snadné a „bezbolestné“. Inspiraci vývojáři čerpali v dalších frameworcích stejného jazyka, ale i v Ruby on Rails, ASP.NET MVC a nebo třeba v Sinatra. Jeho velkým bonusem je také opravdu obsáhlá dokumentace. [16]

Nette

Nette je daším z řady PHP frameworků. Jedná se o aplikaci vyvíjenou českými vývojáři, která je velice oblíbenou. Mezi hlavní významné prvky dozajista patří úsporná a zapamatovatelná syntax, jednoduchá použitelnost, česká obsáhlá dokumentace, integrace AJAX a HTML5. Na tomto frameworku také staví spousta českých ale i světových webů, jako například DHL, Slevomat, Mladá fronta a další.[12]

Yii

Yii je aktivně vyvíjeným ruským PHP frameworkem. Dnes však již na vývoji pracují vývojáři z celé Evropy a tak je přídomek „ruský“ spíše přežitkem. Je jedním z nejvhodnějších kandidátů pro implementaci REST API služeb, vlastních portálů, fór, CMS a dalších služeb. Důležitým prvkem jsou zde komponenty, které usnadňují a umožňují snadno rozšiřovat nejen funkčnost ale i životnost celé aplikace. Za zmínku stojí vysoká optimalizace aplikace, využívání standardního MVC přístupu a hlavně fakt, že je neustále vyvíjen větší skupinou vývojářů. V dnešní době je označován jako Yii 2, kde 2 je číslo hlavní verze avšak v přípravě je již třetí verze tohoto frameworku.

4.5 Databáze

Za účelem zachování dat a sběru statistik je nutné využívat nějaký systém souborů. Pro tyto účely jsou nejvhodnější SQL[13] databáze. Nejdůležitějšími faktory databáze jsou její rychlost a možnosti. Rychlost jde samozřejmě ruku v ruce s návrhem a strukturou dat (tabulek). Pokud je návrh proveden nekvalitně, může silně ovlivnit rychlost získávání dat, což poté ovlivňuje rychlost přístupových bodů, která se poté propaguje až do klientů.

Na poli databázových serverů máme v dnešní době spoustu klientů, mezi nejznámější můžeme zařadit MySQL, Oracle nebo NoSQL.

4.5.1 MySQL

Jedna z nejoblíbenějších a dalo by se i říci nejoblíbenější varianta databázového serveru. MySQL je open-source[14] aplikací. Na počátku nového tisíciletí se zdála být malým konkurentem, kterému se i velcí hráči na poli databázových serverů ušklíbli, jak píše Gilfillan ve své knize[4]. Dnes však již spousta z nich na trhu nepůsobí.

Výhodou této aplikace je zejména to, že se jedná o open-source projekt a tak je samotná aplikace zdarma. Povětšinou ji můžeme nalézt spolu s Apache serverem a PHP serverem, kde se jim na linuxových serverech přezdívá zkratkou LAMP server.[15]

4.5.2 Oracle Database

Jedna z nejvýkonnějších variant vyvíjena korporací Oracle, jak již vypovídá název. V roce 1977 byla založena společnost Software Development Laboratories, které se později v letech 1983 přejemnovaly na Oracle Systems Corporation, později se název zkrátil na Oracle Corporation. První komerčně dostupnou variantou byla Oracle V2, což byl první komerčně

[13] SQL – Structured Query Language
[14] open-source – aplikace s otevřeným zdrojovým kódem, do které může kdokoliv způsobilý přispívat
[15] LAMP – Linux Apache MySQL PHP server

dostupný SQL RDBMS [16]. Vývoj pokračoval a v roce 1992 byl spolu se sedmou verzí této databáze vydán jazyk PL/SQL [17]. Po třiceti letech vývoje se můžeme setkat již s jedenáctou verzí této aplikace. Samotná aplikace je vhodná pro Big Data [18] a nebo obrovské relační databáze na které již nestačí MySQL či jiné služby [19].[15] Aplikace je však určena pro komerční užití a její vlastnosti a zejména komerční licence jsou pro tuto práci nadbytečné.

4.5.3 MariaDB

MariaDB je novější alternativou k MySQL. Opět se jedná o aplikaci zdarma vyvinutou původními vývojáři MySQL. Je zde garantována open-source. Jejími největšími přednostmi jsou rychlost, škálovatelnost, robustnost a široká škála pluginů, pomocí kterých lze funkcionalitu rozšiřovat.[8]

4.5.4 NoSQL

NoSQL je databázovým serverem umožňujícím využívat moderních přístupů pro práci s relačními daty. Dle popisu, standardní relační databáze nebyly a nejsou uzpůsobeny k tomu, aby zvládaly agilní metodiky vývoje, mnohonásobné změny ve struktuře databáze, nebo moderně využívané datové typy. Jednou z dalších výhod je možnost horizontálního škálování serverů, tj. přidávání dalších databázových serverů sloužících pro stejnou aplikaci, namísto přidávání dalšího výpočetního výkonu jednomu serveru (též označováno jako vertikální škálování). [10]

[16] RDBMS – Relational Database Management System = relační systém řazení báze dat
[17] PL/SQL – Programming Language/SQL = nadstavba jazyka SQL vyvinutá společností Oracle
[18] Big Data – souhrn dat který je již nad schopnosti zpracování běžně dostupnými HW a SW prostředky
[19] obrovské relační databáze – databáze mající tabulky čítající desítky milionů řádků

Kapitola 5

Návrh architektury

V této kapitole pojednávám o veškerých poznatcích, které ze zadání vyplynuly rozvahami nebo schůzkami s vedoucím práce. Následně zde popíši detailně případy užití, návrh databáze a v poslední části popíši návrh celé architektury.

Jednotlivé body zadání jsem rozebral následovně:

- Existence přístupového API
- Vzdálené přidávání serverů přes API
- Notifikace
- Možnosti sdílení na sociálních sítích

Ze zadání je jasné, že je nutné využít přístupového API. V případě že již takové API je, bylo vhodné jej jak dle mého, tak i dle názoru Dr. Burgeta rozšířit natolik, aby bylo možné využít rozdělení na front-endovou a back-endovou část aplikace. Tedy aby na jednom stroji běžel pouze webový server obsluhující chování uživatelů a na jiném stroji běžel samotný back-end.

Z původních návrhů jsme s vedoucím došli k možnosti využít nějakého PHP Frameworku pro back-endovou část aplikace a pro front-endovou část využít JavaScriptového frameworku. Ze strany Dr. Burgeta zazněly zejména Angular a React. Já jsem se osobně rozhodl pro React. React je větší výzvou a umožňuje si naprogramovat vše co je potřeba dle své libosti.

Co se týká PHP Frameworku, naskytovaly se varianty jako Laravel, Nette nebo Yii. V konečném rozhodování jsem si vybral právě framework Yii, právě proto že v něm již přibližně rok pracuji, mohl jsem tak zužitkovat své zkušenosti z programování v daném frameworku. Následovalo rozhodnutí jakou verzi použít – Yii framework má dnes již 3 hlavní verze, kde verzi 1.1 končí v brzké době podpora. Následuje verze 2, která má plánovanou podporu až do roku 2025. V neposlední řadě se naskytovala verze 3, ta je však nyní ještě v ranných fázích vývoje a tak došlo k výběru verze 2. Ta se silně liší od svého předchůdce, zejména co se týká podpory REST API.

Pro účely databázového serveru jsem se rozhodl využít databáze MySQL. Při původním návrhu jsem počítal s tabulkami o cca 300 000 řádcích, pro které by měla MySQL databáze stále ještě silně dostačovat.

Architektura aplikace sestává z databázového serveru, který přímo komunikuje s API serverem. Tento API server je tvořen PHP Yii frameworkem. Databázový server běží na stejném stroji jako aplikační rozhraní. Na stejném stroji také funguje se spolupráci s API

monitorovací aplikace, která provádí neustále měření herních serverů. Na separátním stroji již běží samostatný Node.js server spolu s Webpack aplikací, který provádí překlad Javascriptové React aplikace do zabalíčkované, nečitelné formy. Tato aplikace dále komunikuje s aplikačním rozhraním. Komunikaci z React aplikace obstarává knihovna axios.

5.1 Případy užití

Kompletní případy užití popisuje schéma 5.1, avšak stručný komentář si případy zaslouží.

Z případů užití plyne několik podmínek rozšiřujících zadání. Aplikace musí být použitelná i pro osoby neregistrované. Musí být tedy všichni schopni vyhledat server, sledovat statistiky a samozřejmě, musí mít možnost registrace.

Uživatelé již musí mít možnost přihlášení se, dále pak musí mít stejné možnosti jako neregistrovaný uživatel. Také musí být servery mazat a případně i upravovat. Dále pak existuje rozšířený uživatel, který má možnost registrace klíčů pro aplikační rozhraní. Tento rozšířený uživatel je označen jako registrátor – takový registrátor může tento aplikační klíč použít pro přidání serverů z vlastní aplikace. Může tak provést integraci do svého webu. Zaměstnanec poté musí mít možnost smazat statistiky, přidávat nebo odebírat hry a samozřejmě stejná práva jako registrátor.

V neposlední řadě je zde přítomný čas, který je reprezentován aplikací CRON. Ten provádí samotný sběr statistik a zasílání notifikací uživatelům.

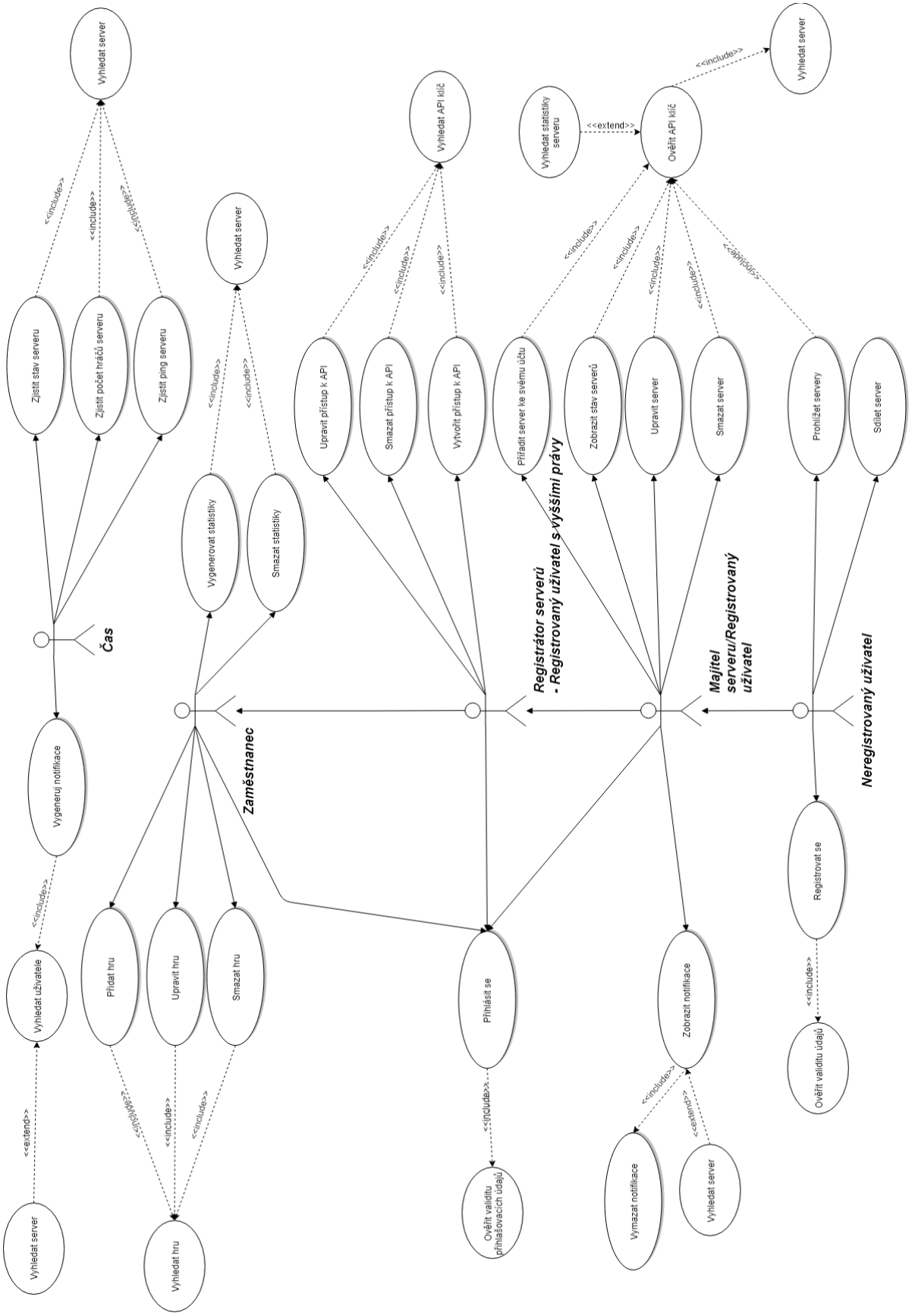
5.2 Návrh databáze

Jak již bylo zmíněno v sekci 5, pro databázový server je využita aplikace MySQL Server. Návrh samotné databáze je prakticky tou nejdůležitější částí. Pokud by byla struktura navržena špatně, mohla by silně ovlivnit výkon a rychlost aplikace.

V aplikaci jsou dva úhlavní prvky – server a uživatel. Tabulka serveru obsahuje název, popis, obrázek, informace o připojení k němu jako IP adresu a port. Na základě těchto informací se k serveru připojuje monitorovací aplikace 7.1. Každý server má také vlastníka. Tím je právě uživatel.

Uživatelská tabulka je sice strohá, ale o to asi jedna z nejdůležitějších. Obsahuje přihlašovací jméno a heslo v hašované formě. Samozřejmostí je jméno a příjmení za účely personalizace aplikace. V tabulce uživatele jsou také udržovány hodnoty časových razítek kdy bylo naposledy aktualizováno heslo a e-mail. V neposlední řadě je zde udržován souhlas se zpracováním osobních údajů a souhlas s podmínkami používání aplikace.

Každý uživatel který se přihlašuje přes přihlašovací tokeny, které musí být samozřejmě nějakým způsobem ukládány a validovány. Pro tyto účely slouží tabulka login_token, která obsahuje náhodný string, datum vydání, datum expirace a ID uživatele, pro kterého je vytvořen. Každý token je tak jedinečný. Další důležitou tabulkou, která se pojí k uživateli je tabulka user_notification. Ta obsahuje datum vygenerování notifikace, uživatelské ID pro kterého byla vytvořena, informaci o tom zda byla přečtena nebo ne, a také pole objektů ze kterého se poté v klientovi může a také skládá URL adresa. Výhodou tohoto přístupu je možnost jednoznačně určit objekt pro který se notifikace vztahuje, zároveň se jedná o univerzální řešení, notifikace není vázána přímo přes cizí klíč k jedné tabulce, ale vytváří abstrakci nad vazbami mezi tabulkami. Více o tomto přístupu je popsáno v sekci 6.5.



Obrázek 5.1: Případy užití

Kapitola 6

Implementace serverového rozhraní

Jak již bylo zmíněné v kapitole 5, aplikační a administrační (webové) rozhraní je tvořeno v PHP frameworku zvaném Yii. Aplikační rozhraní je propojené se samotným webovým rozhraním - ve webovém rozhraní je vytvořena základní administrace pro jednoduchou správu a editaci potřebných dat. Aplikační rozhraní dodává přístupové body pro práci s daty.

6.1 Yii 2 – PHP Framework

PHP Framework **Yii 2** je v tomto případě využit pro tvorbu **API**[1] a k tvorbě a udržování dokumentace pro rozhraní **API**. Základním stavebním prvkem je třída *ApiController*, ze které dědí všechny ostatní controllery použité pro přístupové body. Tato třída implementuje postupy pro validaci uživatelů, zjištění konkrétně použitých validačních metod a také udržuje seznam všech domén, ze kterých se lze přihlašovat přes uživatelské rozhraní. Každý controller má také jasně určenou verzi, pro kterou je vytvořen.

API je strukturováno do jednoduchého modulu s názvem: **v1** obsahující složky controllerů a modelů. Takový přístup je do budoucna vhodný pro snadnou rozšiřitelnost aplikace. V případě, že bude nějaká metoda zastaralá, není nic jednoduššího než vytvořit nový modul (např. s názvem: **v1.1**), který bude dědit ze všech tříd v modulu **v1**, avšak tyto třídy rozšíří o vlastní metody. Je tak zaručena zpětná kompatibilita se staršími implementacemi.

Pro tvorbu přístupových bodů **API** je nutné nastavit správné cesty k těmto souborům. To vše řeší pravidla cest komponenty *UrlManager* v aplikaci **Yii 2**. *UrlManager* provádí snadné směrování cest v **API** k patřičným controllerům. Dovoluje spolu s controllery vytvářet vlastní přístupové body.

6.2 Směrování

Jak již bylo výše zmíněno, o směrování v aplikaci se stará komponenta *UrlManager*, která umožňuje využít již vytvořených metod, které nabízí samotný framework pro tvorbu REST přístupových bodů. Výhodou těchto přístupových bodů je možnost jejich zanořování a předávání parametrů. Takto lze vytvářet parametrizované cesty a v závislosti na parametrech jsou poté načítaná rozdílná data, jak je možné vidět na výpisu 6.1.

[1] API – Application Programming Interface = aplikační programovací rozhraní

```
[
  [
      'class' => 'yii\rest\UrlRule',
      'controller' => ['v1/service'],
  ],
  [
      'class' => 'yii\rest\UrlRule',
      'controller' => ['servers' => 'v1/server'],
      'prefix' => 'v1/services/<parent:\d+>',
  ],
  [
      'class' => 'yii\rest\UrlRule',
      'controller' => ['stats' => 'v1/stats'],
      'prefix' => 'v1/services/<service_id:\d+>/servers/<parent:\d+>',
  ],
]
```

Výpis 6.1: Ukázka cest v route.php konfiguračním souboru

6.3 Zpracování požadavků

Jednotlivé požadavky jsou *UrlManagerem* směrovány do jednotlivých controllerů. Tyto controllery jsou určeny na indexu **controller** v poli cest. Zpracování samotné je poté v metodách **controllerů**, které se označují jako **actions**[2].

Jelikož se jedná o klient-server aplikaci, kde jak klient tak i server běží na jiných doménách, je nutné vyřešit **CORS policy**[3]. Pro správné fungování systému **CORS** je nutné mít pro každý přístupový bod aktivní akci *OPTIONS*, která zařízení dotazujícímu se na tento přístupový bod sdělí veškeré informace o dalších dostupných akcích a zároveň pomocí *HTTP hlavičky*[4] ověří hodnotu v prvku: *Access-Control-Allow-Origin*. Ta se musí shodovat s doménovým jménem webu, ze kterého je požadavek zaslán.

Samozřejmostí v případě přístupového rozhraní je nutné provádět nějakou formu autentizace uživatelů. U **GET** a **HEAD** akcí se neprovádí žádné ověřování uživatele. To je dáno tím, že tyto data lze ze struktury aplikace získávat volně. Není v nich nic, co by veřejnost nesměla vidět. Na tomto principu staví rozšíření pro získání stavu serveru pro klienty8.5.3. Existuje více variant jak řešit autentizaci uživatelů u **PUT**, **DELETE** a případně dalších požadavků. Jednou variantou je předávání tzv. *HTTP Bearer token* položky v *HTTP hlavičce*. To je bohužel pro tuto práci vcelku nevhodné. Důvod je prostý, prvek neumí uložit případná další data k tokenu a určit tak jasně o jaký typ tokenu se jedná. Aplikace totiž umožňuje přístup k API jak formou webového rozhraní, tak poté pomocí API požadavků mimo webové rozhraní. Tedy zpracování na pozadí. Pro tyto účely je nutné ze strany klienta, který se na API dotazuje posílat v **POST** datech položku: *login_token* nebo *registrator_token* v závislosti na tom, jak je aplikace použita. O přihlášení více ale v další sekci.

[2] **actions** – akce, speciální pojmenování metod v controllerech

[3] **CORS policy** – Cross-origin resource sharing = mechanismus řešení přístupu k datům mezi různými doménami[6]

[4] HTTP hlavička – první část zprávy, která se zasílá v odpovědi, druhou částí jsou data

6.4 Přihlášení

Jak již bylo zmíněno, tato práce umožňuje přihlašování dvěma formami. První formou je přihlášení přes webové rozhraní, které je popsáno v implementaci klientské části8.3. Pro tuto formu si uživatel sám vkládá servery, které chce monitorovat a sdílet na svém webu přes unifikované výstupy. Varianta využívá přihlašování přes přihlašovací tokeny, tedy *login_token*, který je ve formě **JWT**[5] uložen na straně klienta. **JWT** token obsahuje vždy přihlašovací token uložený v databázi serveru, datum vystavení tokenu, datum expirace tokenu, doménové jméno serveru který token může použít a doménové jméno serveru, který token vystavil. Token je poté jako celé pole spojen do jednoho řetězce a zašifrován pomocí SHA256. Takový řetězec je poté poslán klientovi na webovém rozhraní, které jej dešifruje, zkontroluje datum vystavení a expiraci. Pokud jsou hodnoty správné, je tento token zasílání rozhraním při operacích potřebných jako součást těla zprávy ve formě jak je k vidění na výpisu 6.2

```
{
    "login_token": "_XjP_PFV1Y7XmtDufTb7g6Dq0wNdSY3S",
    .
    .
}
```

Výpis 6.2: Formát POST dat zasílaných z webového rozhraní na API

Při komunikaci s API server včasně upozorní klienta na expiraci tokenu a ten si vyžádá nový token, který bude platný. Standardní doba platnosti tokenu je 1 měsíc.

Druhou variantou je přihlašování přes aplikační rozhraní. Tuto variantu mohou využívat registrátoři[6]. Pro aktivaci tohoto modu je nutné si v nastavení účtu vygenerovat aplikační token, kterým je náhodně vygenerovaný řetězec. Doba expirace se udržuje pouze v databázi aplikačního rozhraní. Tokeny tak mohou mít neomezenou platnost. Princip je jednoduchý a silně se podobá aplikačním rozhraním bankovních společností. Při každém požadavku se tento token zašle a ověří se tak jeho validita a stav expirace. Výhodou tohoto přístupu je možnost kontrolovat počet požadavků pro určitého registrátora. Každý tento požadavek se může zaznamenávat. Takový postup využívají společnosti, které své API zpoplatňují a umožňují určitý počet požadavků zdarma. Příkladem může být například společnost Google a jejich Google Maps API.

Token registrátora se vždy v datech označuje jako: *registrator_token*, formát je poté stejný jako u uživatelského tokenu. Hlavním rozdílem je rozdílná ochrana tohoto tokenu. Takovýto token je vždy dovoleno užít jen z určité sady IP adres, kterou uživatel po vygenerování tokenu zadá v uživatelském rozhraní. Pokud se bude pokoušet o připojení z jiné IP adresy, bude tento pokus odmítnut.

```
{
    "registrator_token": "YBJDZp_cXQTKKgi_IJW3vBxfuevfVJxi",
    .
    .
}
```

Výpis 6.3: Formát POST dat zasílaných ze strany registrátora při implementaci API

[5] **JWT** – **J**SON **W**eb **T**oken = forma kódování přihlašovacích tokenů

[6] registrátoři = registrátorem se může stát prakticky kdokoliv, kdo chce přes aplikační rozhraní vzdáleně vytvářet a upravovat servery, nejvhodnější případ užití jsou ale hostingové společnosti

Ověření na straně serveru probíhá ve statické metodě *findByAccessToken*, její logika je k vidění ve výpisu 6.4. konstanty API_LOGIN a USER_LOGIN (druhá varianta, která je skrytá v `else` větvi zdrojového kódu) určují typ přihlášení dle zadaného přihlašovacího tokenu. Konstanta LOGIN_TOKEN_KEY je speciální tajný klíč, podle kterého se veškerá data šifrují a zpětně dekódují v **JWT** tokenech.

```
public static function findByAccessToken($validationMethod, $data)
{
        $modelName = '';
        $model = null;

        if ($validationMethod === self::API_LOGIN)
        {
                $model = RegistratorToken::findByToken($data);
                if (!$model)
                        throw new ApiException(401, 'User not existing');

                if ($model->isExpired())
                        throw new ApiException(403, 'Token expired');

                $ip = \Yii::$app->request->getUserIP();
                if (!$model->checkIPAddress($ip))
                        throw new ApiException(412, 'Not allowed from this location');
        }
        else
        {
                $token = JWT::decode(
                           $data,
                           LoginToken::LOGIN_TOKEN_KEY,
                           array("HS256")
                    );
                $model = self::findAccessToken($validationMethod, $token->token);
                if (!$model || $model->isInvalid())
                        throw new ApiException(403, 'Token is expired.');
        }
        return $model->user;
}
```

Výpis 6.4: Validace uživatelských tokenů

Přihlášený uživatel poté může přidávat nové servery ve webovém rozhraní. Ty se po registraci automaticky stávají monitorovatelnými a v následujícím cyklu monitorování se provede sledování stavu statistik. Pro přidání serveru je nutné zaslat požadavek na aplikační rozhraní obsahující strukturu v těle zprávy podobnou jako na výpisu 6.5.

```
{
    "login_token": "_XjP_PFV1Y7XmtDufTb7g6Dq0wNdSY3S",
    "server": {
        "name": "JMENO",
        "ip": "123.123.23.23",
        "port": 12345,
        "description": "POPIS"
    }
}
```

Výpis 6.5: Ukázkový obsah těla zprávy sloužící pro vytvoření nového serveru přes API

6.5 Notifikace

Součástí aplikace je také notifikační systém. Tento systém je součástí uživatelských koncových bodů. Každá notifikace se vždy váže přímo na jednoho uživatele. Pro účely generování těchto notifikací jsem vytvořil statickou metodu v modelu UserNotification, který se váže na tabulku notifikací. Tato metoda má za účel jediné, chová se prakticky jako konstruktor jednotlivých notifikací. Ve snaze o vytvoření jednotného rozhraní jsem do metody přidal jako první parametr sadu uživatelských ID, kterým se má zaslat tato notifikace. Pokud je toto pole prázdné, vygeneruje se notifikace všem uživatelům. V tento moment se dají tedy generovat notifikace manuálně, nebo automaticky právě touto metodou. Generování provádím v tento moment při vytvoření nebo úpravě nějakého serveru, dochází tak k oznámení všem uživatelům pro účely prezentace této funkcionality. Nejdůležitějším prvkem notifikace je dozajista atribut **objectArray**. V tomto atributu ukládám JSON objekt, pomocí kterého ve front-end části aplikace vytvářím jednoduchým for cyklem URL adresu, jsou jednotlivé přístupové body určeny klíčem a identifikace objektů hodnotou. Například adresa `/conditions` by byla do atributu **objectArray** uložena ve formátu: {"conditions": ""}, kdežto adresa: `/services/1/servers/52` by byla uložena jako: {"services": 1, "servers": 52}.

```
public static function notify($userIds = [], $title, $message, $data)
{
        if (empty($userIds))
        {
                $userIds = new ActiveDataProvider([
                        'query' => User::find()
                ]);
        }

        else {
                $userIds = new ActiveDataProvider([
                        'query' => User::find()
                                ->andWhere(new InCondition('id', 'IN',
                                    $userIds)),
                ]);
        }

        foreach ($userIds->getModels() as $user)
        {
                $n = new UserNotification;
                $n->user_id = $user->id;
                $n->content = $message;
                $n->date = new Expression('NOW()');
                $n->title = $title;
                $n->objectArray = $data;
                $n->save();
        }
}
```

Výpis 6.6: Metoda pro generování notifikací

Kapitola 7

Monitoring – konzolová aplikace

Hlavním bodem zadání je samozřejmě monitoring serverů, nad kterým je postaveno celé API a samozřejmě jej využívá i React klient. Tuto monitorovací aplikaci jsem se rozhodl implementovat jako součást API. Pro tyto účely je vhodná konzolová aplikace frameworku Yii, která umožňuje vytvořit ve standardním controlleru vlastní metodu. Takovou metodu je poté lehce možné zavolat nejen uvnitř framewrku, ale také externě přes příkazový řádek.

7.1 Návrh implementace

Při implementaci aplikace jsem se snažil vymyslet způsob, jakým zařídit co nejsnáze rozšiřitelnou variantu monitorování v závislosti na integraci do webové aplikace. Jak již bylo zmíněno výše, výběrem byla konzolová aplikace Yii. S jejím použitím bylo možné využít stávajících modelových a databázových struktur pro získávání stavů a informací o jednotlivých serverech.

Každý server spadá pod jednu službu, každá tato služba (v nynějším případě pouze hry) má v databázi určen krátký název, podle kterého se určují třídy spojené s monitoringem pro danou službu. Každá služba má v databázi ve sloupci *stats_list* uloženo pole všech dostupných monitorovaných hodnot. Pro začátek jsou využívány hodnoty: **PingStat**, **PlayersStat**, **StatusStat**. Jméno každé sbírané statistiky se skládá vždy z názvu hodnoty, která se udržuje a z všeobecného názvu **Stat**. Celé jméno poté určuje název modelu, který slouží k uložení dat do databáze pro daný typ. Výhodou takovéhoto přístupu je lehká modifikovatelnost aplikace. V reálném prostředí stačí přidat hodnotu pro sledování do databáze k dané službě, následně pouze vytvořit model (třídu) pro korektní ukládání dat.

7.2 Použité knihovny

Samozřejmě data je nutné odněkud vygenerovat. Aktuální herní servery využívají systém Query, což je systém přístupových bodů, které jsou k dispozici bez nutnosti připojení se herním klientem k serveru. Takovéto systémy jsou často velice dobře specifikovány v dokumentacích těchto serverů. Naskytly se tedy dvě varianty řešení – buď implementovat kompletně sám celou query knihovnu pro specifikovaný server a nebo využít již vytvořené řešení pro připojení ke Query serveru pro získání dat. Pro účely práce jsem se rozhodl nejprve zkusit najít stávající řešení, než implementovat sám dotazování serverů. Urychlil se tak podstatně vývoj aplikace samotné. K mému štěstí již existují podobné aplikace, z toho důvodu jsem

se rozhodl využít stávajících dostupných knihoven a pouze rozšířit jejich funkčnost pro své účely.

V práci využívám aktuálně dvě knihovny – knihovnu PHP Source Query[1] a PHP Minecraft Query[2]. Minecraft Query získává bohužel status pouze k Minecraft[3] serverům. Narozdíl od ní Source Query zvládá získávat status serverů, které jsou postaveny na Source Enginu[4]. Takových her je opravdová spousta a pro účely a ukázání funkčnosti jsem se rozhodl implementovat pouze herní servery pro aplikace Counter-Strike: Global Offensive[5] a RUST[6].

Rozšíření je provedno jednoduchým try – catch blokem, který nejprve zajišťuje a zkouší komunikaci na server. Pokud se spojení v pořádku naváže, obdrží statstiky, které si uloží. Samotné dotazování probíhá podobně jako na výpisu 7.1. Samozřejmě se jedná o pouhý dotaz na serverovou aplikaci, data se však v tomto stavu ještě nikam neukládají, pouze se předávají dále. Jejich zpracování je již dále rozvedeno v sekci 7.3 a ve výpisu 7.4.

```
class MCQuery extends BaseQuery
{
        public static function query($server)
        {
                $query = null;
                $queryResult = [];
                try {
                        $query = MCPing::check($server->ip, $server->port, 5);
                } catch (MCPingException $e) {
                        $Exception = $e;
                } finally {
                        if (isset($query->ping))
                                $queryResult['ping'] = $query->ping;
                        if (isset($query->players))
                                $queryResult['players'] = $query->players;
                        if (isset($query->max_players))
                                $queryResult['max_players'] = $query->max_players;
                        if (isset($query->online))
                                $queryResult['status'] = $query->online ? 1 : 0;
                        if (isset($query->favicon))
                                $queryResult['image_url'] = $query->favicon;
                }
                return $queryResult;
        }
}
```

Výpis 7.1: Ukázka zjištění stavu z herního serveru

[1] PHP Source Query – https://github.com/xPaw/PHP-Source-Query

[2] PHP Minecraft Query – https://github.com/xPaw/PHP-Minecraft-Query/

[3] Minecraft – počítačová sandboxová hra určená pro stavění vlastního světa, dnes vlastněná společností Microsoft

[4] Source Engine – Herní engine vyvinutý společností Valve

[5] Counter-Strike: Global Offensive – akční FPS střílečka vyvíjená společností Valve

[6] RUST – survival strategie vyvíjená společností Facepunch Studios

7.3 Integrace do webové aplikace a generování

Každý server má metodu *generateStatistics*, která provádí generování každé dostupné statistiky pro daný typ služby. Každá statistika má základní model s názvem: **StatModel**, tento model má nachystané univerzální metody pro předgenerování statistiky. Z této třídy jsou následně rozšířeny třídy PingStat, PlayersStat a StatusStat. Vygenerování je zařízeno metodou *generateStat*, která provádí validaci obdržené hodnoty a její uložení. Veškerá data jsou pro každý server udržována po dobu 14 dní. Při každém generování statistik dochází ke kontrole již nepotřebných statistik (starších než 14 dní) metodou *destroyOldStatistics*, která provádí i jejich mazání. Náhled na část kódu obsluhující monitoring serverů je na výpisu 7.2. Je zde dozajista prostor pro vylepšení, jehož popisu se budu dále věnovat v závěru této práce.

Důležitým faktorem je zde získání všech aktuálně dostupných serverů. Dále se vygeneruje aktuální datum a čas pro ušetření alespoň části práci databáze. Pokud nedojde k nalezení žádného serveru, ukončí se zpracovávání monitorování chybou. Po nalezení potřebných serverů se inicializuje prázdné pole serverů, které skončily při generování hodnot chybou a následně se v cyklu nejprve smažou staré statistiky a následně vygenerují nové.

```
public function actionGenerate()
{
	$startDate = (new \DateTime());
	$startDate = $startDate->format('Y-m-d H:i:s');
	$servers = Server::find()->all();
	if (!$servers)
		return ExitCode::NOINPUT;

	$failedServers = [];

	foreach ($servers as $server)
	{
		$server->destroyOldStatistics();
		if (!$server->generateStatistics($startDate))
			$failedServers[] = $server;
	}
}
```

Výpis 7.2: Ukázka funkčnosti monitorovací aplikace

Jelikož si u každé dostupné služby udržuji jasně veškeré dostupné statistiky, je poté mazání starých statistik vcelku triviální metodou. Jak je k vidění na výpisu 7.3, pro model Server je dostupná metoda *destroyOldStatistics()*, která nejprve zjistí statistiky dostupné pro Server (respektive pro danou službu) a následně jednoduchým cyklem provede jejich smazání z databáze. Matoucí může být metoda *getClassPath()*. Ta zaručuje vygenerování správné cesty ke třídě dané statistiky. Takto lze provést dynamické volání tříd z dané složky, čímž mohu rozšiřovat typy sbíraných hodnot teoreticky až do nekonečna.

```
public function destroyOldStatistics()
{
        $stats = $this->getAvailableStatistics();

        foreach ($stats as $stat)
        {
                $className = $this->getClassPath().$stat;
                $className::destroyOldStatistics($this->id);
        }
}
```

Výpis 7.3: Mazání starých statistik

Druhým, avšak o to důležitějším krokem monitorování je samotný sběr statistik. Ten již provádí metoda *generateStatistics()* třídy Server. Obsahuje parametr **$startDate** a **overrideSaving**. Parametr **$startDate** je použit za účelem ušetření práce databáze zbytečným negenerováním aktuálního času a za druhé umožňuje určit jednoznačně čas v jakém bylo spuštěno monitorování. Toto dopomáhá k zjednodušení filtrace dat při jejich zpracování na výstupu. Druhý parametr **overrideSaving** je použit v pro účely opakování monitorování – pokud dojde k chybě při sledování stavu serveru, jeho stav se při prvních dvou pokusech neukládá, ukládá se vždy až třetí pokus o vygenerování statistik. Tímto postupem dochází k odfiltrování masivní valné většiny stavů, kdy mohlo dojít pouze k dočasné ztrátě spojení nebo mohl dočasně vzdálený server mé připojení schválně odmítnout. Jak popisuje výpis 7.4, nejprve se vyhledají veškeré dostupné statistiky pro službu daného serveru. Poté se vyhledá objekt služby serveru. Dále je nutné vygenerovat název třídy rozšiřující funkčnost knihovny pro získání informace o stavu serveru, která je použitá pro vygenerování statistiky. Tato třída vždy obsahuje statickou metodu query, kterou jsem si vytvořil sám jako nadstavbu pracující s původními knihovnami. Výsledek je uložen do proměnné **$result**. V další části se provede jednoduchý cyklus, který pro každou statistiku dostupnou pro danou službu získá z proměnné **$result** výsledky proměnných, které jsou sledovány. Z těchto proměnných poté vygeneruje potřebné modely a uloží je do databáze. Důležitá je okamžitá kontrola položky status z výsledku, kterou ověřuji zda je server dostupný, respektive zda dotaz skončil úspěšným nebo neúspěšným získáním hodnot.

```
public function generateStatistics($startDate, $overrideSaving = false)
{
        $stats = $this->getAvailableStatistics();
        $service = Service::findById($this->service_id);
        $queryClass = $this->queryNamespace.'\\'.$service->getQueryClass();
        $result = $queryClass::query($this);
        $failedGeneration = 0;


        foreach ($stats as $stat)
        {
                $className = $this->getClassPath().$stat;
                echo "Generating {$stat} statistics for server {$this->id}:
                    {$this->name}\n";
                if (!$overrideSaving && isset($result['status']) && $result[
                    'status'] == 0)
                {
                        \Yii::debug("Stat: {$stat} could not be generated for
                             server {$this->id}: {$this->name} because server
                            is OFFLINE.");
                        return false;
                }
                else if (is_null($status = $className::generateStat(
                    $startDate, $this->id, $result)))
                {
                        \Yii::debug("Stat: {$stat} could not be generated for
                             server {$this->id}: {$this->name}");
                        $failedGeneration++;
                }
        }
        if ($failedGeneration == count($stats))
                return false;
        return true;
}
```

Výpis 7.4: Ukázka zdrojového kódu tvorby statistik

V průběhu vývoje aplikace jsem prošel několik způsobů integrace a generování statistik. Původní návrh spočíval v generování pouze v případě, že se uživatel připojí k webu a při pokusu o načtení dat o serverech se vygenerují statistiky. Probíhalo by tak generování v reálném čase. Po testování aplikace jsem došel k závěru, že takové chování není vhodné obzvláště v případech, kdy některé servery nemusí být dostupné a při neúspěšných dotazech může aplikace takto zjišťovat stavy třeba i 10 – 20 vteřin pouze pro 10 serverů (s předpokladem že máme 2 vteřiny na doba ukončení spojení). Tento návrh jsem se rozhodl předělat na automatické generování každé 2 hodiny s dodatečným generováním aktuálních statistik pro právě načtené servery. Takový systém byl však stále silně neefektivní. Po konzultaci s vedoucím práce jsem se rozhodl implementovat statistiky generovat každých 10 minut pro

všechny servery uložené v databázi. Pokud u nějakého nastane chyba, dojde k uložení do pole **$failedServers**, ve kterém se ve dvou podobných cyklech, které se opakují ve po dvou minutách pokusí aplikace znovu získat stav serverů a až při třetím neúspěšném pokusu ukončí hledání a uloží neúspěšný stav do databáze. Desetiminutové opakování je zajištěno pomocí aplikace CRON[7]. Dvouminutové čekání je zajištěno přímo ve skriptu pomocí funkce *sleep()*.

[7]CRON – tzv. démon umožňující spouštět v daných časových intervalech určité procesy

Kapitola 8

Implementace klientského rozhraní

Uživatelské prostředí, klient chcete-li, je tvořené aplikací běžící nezávisle na funkčnosti serveru. Jedná se o tzv. `fat-client` implementaci, kdy je webová aplikace plnohodnotnou funkční aplikací. Z aplikačního rozhraní si aplikace pouze stahuje potřebná data formou API požadavků, která jsou uložena ve formě JSON. Implementace probíhala v JavaScriptové knihovně React.

8.1 React – Úvod

Jak bylo zmíněno v kapitole 4.3.5, React je Javascriptovou knihovnou sloužící opět k tvorbě interaktivních webových rozhraní. Pro tyto účely využívá VirtualDOM, což je programovací koncept, který v paměti udržuje reálný strom objektů. Funguje nezávisle na tomto stromu a tím dokáže mnohonásobně zvýšit rychlost práce s takovýmto stromem. Úhlavním prvkem Reactu jsou komponenty o kterých pojednávám více v sekci 8.1.1.

8.1.1 Komponenty

Stavebním kamenem Reactu jsou komponenty (v angl. originále Component). Existují funkční a třídní komponenty, přičemž varianta třídní je dostupná od ECMAScriptu verze 6 (ES6). Takovéto komponenty jsou si ekvivalentní a jejich použití lze zaměnit. Přístup k elementům v takovýchto komponentách se však značně liší. Zatím co funkční komponenta musí jako parametr obdržet proměnou do které jsou vkládány předávané parametry, třídní komponenta tuto proměnnou obdrží automaticky.

Příklady 8.1 a 8.2 ilustrují tento rozdíl.

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

const element = <Welcome name="Sara" />;
```

Výpis 8.1: Zápis a předávání hodnot do funkčních komponent

```
class Welcome extends React.Component {
    render() {
        return <h1>Hello, {this.props.name}</h1>;
    }
}

const element = <Welcome name="Sara" />;
```

Výpis 8.2: Zápis a předávání hodnot do třídních komponent

V této práci je využíván zejména zápis třídní. Rozhodl jsem se tak z důvodu, který jasně popisuje i název. Třídní zápis se více blíží třídám z klasických OOP jazyků a je dle mého i lépe čitelný. Veškerá data sloužící k jedné komponentě jsou v jedné třídě a je tak možné sdružit i více tříd do jednoho souboru. Je sice doporučeno mít rozděleny třídy dle oficiální dokumentace na systém: 1 třída = 1 soubor. V tomto případě mi ale bylo bližší rozdělení – jedna větev stromu přístupových bodů API = 1 soubor. Aplikace není rozsáhlá na tolik, aby nebylo možné se v ní orientovat a zejména aplikaci programuji sám.

V rámci jednotlivých tříd existují vnitřní stavy, atributy chcete-li, tak jako u klasického OOP jazyka. Tyto atributy lze vytvářet vždy v konstruktoru dané třídy, který musí být přetížen. Příklad takového přetížení je na výpisu 8.3

```
  class Account extends Component
  {
    constructor(props)
    {
        super(props);
        this.classes = props;
        this.state = {
            servers: [],
        }
    }
        render()
    {
      .
      .
    }
}
```

Výpis 8.3: Přetěžování konstruktoru u komponent

Tím se dostáváme k úhlavnímu prvku komponent a to je **stav komponenty** (v angl. originále component state). Tento stav slouží právě pro kontrolu nutnosti rerenderingu[1] komponenty. Pokud dojde k jeho změně, je prakticky vždy vyvolán rerender komponenty. Tento proces lze upravit, ve valné většině případů to však není nutné. Stav komponenty je nutné vždy upravovat přes metodu *setState*, která zaručuje korektní předání informací o změně stavu dalším závislým komponentám. Životní cyklus komponenty Reactu je znázorněn na obrázku 8.1.

Metody lze v Reactu zapisovat v třídách dvěma způsoby. Existuje starší zápis, který známe z klasických programovacích jazyků, který je k vidění na výpisu 8.4. Druhým, nověj-

[1] rerender – proces při kterém dojde k znovu vykreslení pouze určité části aplikace → komponenty

ším zápisem je zápis tzv. *šipkový* (v angl. originále jsou tyto metody označovány jako *arrow functions* nebo *arrow methods*). Rozdíl je zejména v rychlosti zápisu takového zdrojového kódu, dále pak i v rychlosti zpracování. V neposlední řadě je nutné zmínit i lepší předávání identifikátorů. Všeobecně známý je problém rozsahu proměnné **this** v JavaScriptu, tento problém z valné většiny umí tento styl zápisu vyřešit. Zápis tzv. šipkový můžeme vidět na výpisu 8.5.

```
class Account extends Component
{
    renderServers( params )
    {
        let { servers } = this.state;
        let data = [];
        if (servers.length)
        {
            .
            .
        }
        return (data);
    };
        render()
    {
      this.renderServers();
    }
}
```

Výpis 8.4: Standardní zápis metod

```
class Account extends Component
{
    renderServers = ( params ) =>
    {
        let { servers } = this.state;
        let data = [];
        if (servers.length)
        {
            .
            .
        }
        return (data);
    };
    render()
    {
      this.renderServers();
    }
}
```

Výpis 8.5: Šipkový zápis metod

Šipkový zápis metod je definován v ECMAScriptu verze 6. To ale přináší nejen spoustu výhod, ale i spoustu problémů. Starší prohlížeče si s novějšími funkcionalitami jazyka ECMA nejsou schopny poradit a v nejlepším případě kód nezkompilují, v horším jej špatně interpretují a celá aplikace se tak může chovat nevyzpytatelně. Takové chování je ale silně nežádoucí. Jsou však způsoby jak lze problém řešit a dokonce i vyřešit → odpovědí na něj je **Webpack**.

Webpack

Webpack je speciální aplikace, která umožňuje sjednotit veškeré JavaScriptové soubory do jednoho balíčku. Spolu s tím dokáže provádět nad tímto sjednoceným souborem další operace. Těmi může být již dříve zmíněná minifikace a „uglifikace“ 4.3.1, ale dokonce dokáže provádět rekompilaci jazyka vůči staršímu standardu jazyka ECMASCript, čímž vytváří pro vývojáře unifikované prostředí – v práci jsem tak mohl využít novinek jazyka ES6 a přitom byl zdrojový kód dostupný a přeložitelný i v prohlížečích jako je Microsoft Edge nebo ve starších verzích Google Chrome. Webpack také umožňuje linkovat stylizační soubory a dokonce i obrázky pro možnost užití v Reactu.

8.2 React – Aplikace - routování, linkování, skladba (navigace, tělo, footer)

8.2.1 Skladba

Zdrojová struktura také odráží vzhled aplikace, to znamená že existuje složka layout[2], ve které jsou uloženy prakticky neměnné komponenty Footer a Header. Header v sobě vykresluje další komponentu Navigation. Navigation je tedy komponenta starající se o vykreslování navigačních prvků na stránce. Rozhraní tvoří navigační panel obsahující uživatelskou část a tělo aplikace. Uživatelská část zobrazuje vždy jméno, které uživatel zadal při registraci pro částečnou personalizaci webu a ikonku. Při prokliku přes ikonku se uživatel dostane do uživatelského nastavení. V navigačním panelu také nalezneme notifikační panel, který po rozkliknutí zobrazí posledních 5 notifikací, které si uživatel může přečíst. Notifkační systém je více rozepsán v sekci 6.5. Aplikace zobrazuje vždy aktuální stránku, ve které je uživatel nasměrován. Patička stránky obsahuje odkaz na podmínky použití a popis služby.

8.2.2 Routování a linkování

Tyto části jsou vykreslovány v App komponentě, kterou obaluje komponenta BrowerRouter. Ta slouží ke směrování URL tras v singlepage[3] webu. Jednotlivé trasy se poté vytváří pomocí komponenty Route, jejich definice je vyobrazena ve výpisu 8.6.

```
<Route exact path="/" render={props => (
    <Services />
)} />
<Route path="/auth" component={Auth}/>
<Route path="/services/:id" component={servers} />
<Route path="/account" component={Account} />
<Route path="/servers/add" component={ServerForm} />
```

[2] layout – rozložení

[3] singlepage - jednostránkový web, obsah je načítán pouze uvnitř dané stránky, nedochází k načítání celých stránek ale jen překreslení části aplikace

Výpis 8.6: Nastavení URL tras v React aplikaci

Z výpisu je patrné, že má každá cesta vlastní `cestu`, tzv. path. Pro každou cestu je vhodné specifikovat komponentu, která se na této trase má vykreslovat. Ta se předává parametrem `component`. U jedné `cesty` je také použito klíčové slov `exact`. To zaručuje, že se tato trasa vykreslí opravdu když je v názvu pouze `/`. Pokud by u prvotní cesty nebylo toto klíčové slovo použito, vykreslovala by se tato komponenta u všech ostatních cest spolu s jejich komponentami kvůli tomu, že se znak `/` objevuje ve všech dalších cestách. Samotné hledání cesty probíhá podle nejlepší shody, pokud není použito klíčové slovo `exact`, pak se naopak vybírá přesná shoda.

8.3 React – Uživatelé a autentizace

Autentizace uživatelů je asi jednou z největších překážek, pokud je implementujeme v Reactu. Samotná aplikace vyžaduje, aby byl uživatel nejen schopen se registrovat a přihlašovat se, ale také pro účely bezpečnosti byly navrženy metody odhlášení a odhlášení ze všech možných zařízení a hlavně, přidávání, úpravy a mazání jednotlivých serverů. Samozřejmostí je v dnešní době možnost úpravy údajů, které zadali při registraci, jak požaduje nařízení o ochraně osobních údajů.

Uživatel při registraci zadává uživatelské jméno, heslo, email a jméno a příjmení. Tyto údaje jsou potřebné nejen pro správnou funkčnost a ověřování uživatele (uživatelské jméno a heslo) ale také pro vylepšení uživatelského rozhraní. Uživatelský účet je dostupný přes jméno ikonku se jménem uživatele v navigačním panelu. Po rozkliknutí uživatelské sekce je uživateli prezentováno uživatelské nastavení ve kterém má výpis informací o svém účtu, má zde dostupné možnosti odhlášení a také má dostupný výpis všech jím přidaných serverů.

Jak již bylo zmíněno v sekci 6.4, uživatelé se autentizují pomocí JWT tokenů, které jsou uloženy v local storage. Nevýhodou tohoto způsobu ukládání je možnost editace a čtení těchto hodnot. Právě pro tyto účely se využívají JWT tokeny, které jsou zašifrovány speciálním klíčem, jehož zašifrovanou verzi si může přečíst klient, ale při editaci tokenu již nebudou data sedět vůči dešifrované verzi. Tento token obdrží React aplikace po úspěšném přihlášení nebo po úspěšné registraci a provede znovu-vykreslení všech komponent využívajících této uživatelské struktury. Aby bylo možné přenést veškeré informace přes celou aplikaci a nemusel se uživatel několikrát autentizovat pro každou sekci zvlášť, je nutné využít kontextu (v angl. originále Context), což je speciální proměnná, do které je obalen zdrojový kód aplikace. Uvnitř jednotlivých komponent je poté nutné definovat statickou konstantu *contextType*, čímž lze poté přistupovat k uživatelskému kontextu přes unifikované rozhraní: *this.context.prop*, kde prop je proměnná, která je ve stavu třídy **UserContext**.

Samozřejmě není nutné využívat kontext a je možné předávat proměnné do jednotlivých tříd přes jejich proměnnou **props** 8.2, v každé třídě je pak však nutné samostatně zpracovat tuto hodnotu, v každé komponentě ji ukládat do vnitřního stavu komponenty a takto rekurzivně předávat dále. Při úpravě zdrojového kódu je pak nutné metody a proměnné přepisovat ve všech zanořených komponentách zvlášť. Takový postup není žádoucí.

Kontext také umožňuje jednoduše definovat a volat metody „zvenčí“ stejným postupem jako bylo popsáno výše: *this.context.prop*. Definice metod a jejich užití poté může být obdobná jako na výpisu 8.7. Všimněme si, že se metody ukládají do vnitřního stavu komponenty do objektu user a uvnitř něj do objektu actions. Dále si všimněme toho, že jsou předávány pouze názvy metod, bez volání těchto metod a bez výpisu parametrů. Jsou totiž

předávány reference na metody. Pokud bychom zde vložili volání, při inicializaci komponenty by došlo ke zpracování metod a již by nedocházelo k volání metod, nýbrž by došlo k vrácení již uloženého výsledku. Dalším faktem který musím zmínit je fakt, že jsou sice metody dostupné a ukládají se do vnitřního stavu kontextu, pokud ale chceme nějak pracovat s jejich výstupem a na základě jejich změny provádět znovu vykreslení, je nutné jejich výstup do stavu kontextu opravdu ukládat standardní metodou *setState*.

```
export const UserContext = React.createContext();

export class UserProvider extends Component
{
    constructor(props)
    {
        super(props);
        this.state = {
            user: {
                    actions: {
                        checkExpiration: this.checkExpiration,
                        isOwner: this.isOwner,
                    }
                }
        }
    }

    checkExpiration = () =>
    {
        .
        .
        return false;
    }

    isOwner = ( server ) =>
    {
        .
        .
        return true;
    }
}
```

Výpis 8.7: Předávání metod z kontextu

8.4 React – Struktura

Vzhled front-endové části aplikace kopíruje rozvržení API. Po otevření webové stránky je uživateli okamžitě dostupný výpis všech služeb, pro které jsou dostupné servery. Jednotlivé služby spadají do různých kategorií, ty ale nejsou momentálně využity, protože je na webu pouze jedna kategorie služeb a tím jsou herní servery.

Po výběru určité služby dojde k načtení části serverů do výpisu, načte se prvních 12 serverů. Výpis serverů je rozdělen do 2 sloupečků za účelem maximalizace místa a zároveň za účelem zjednodušení designu. Důležité informace jsou dostupné po rozkliknutí serveru, přičemž na výpisu všech serverů je dostupný obrázek serveru (pokud je přidán), název serveru, jeho IP adresa, kterou si uživatel může zkopírovat do schránky a perex[4]. Samozřejmě jsou zde přítomné odznáčky (v angl. originále badges), což jsou speciální prvky HTML, které umožňují zobrazit jednoduché, avšak důležité informace. Pro příklad je nyní vždy zobrazen stav serveru a pokud je server dostupný, vypíše se i aktuální počet hráčů. Způsob generování a získávání těchto statistik je popsán v kapitole 7 a proces filtrace je v sekci 8.5.

8.5 React – Detail, statistiky a jejich interpretace

Detail obsahuje již podrobnější informace, včetně bližšího popisu serveru, kompletní náhledové fotky a také statistiky. Ze strany serveru jsou vždy poslány všechny statistiky dostupné pro danou aplikaci, klient (připojený na API) tak může se statistikami pracovat a interpretovat je tak jak si přeje. Toho právě využívá i React prostředí. Po stažení dat serveru se dodatečně načítají i serverové statistiky. Pokud se žádné nestáhnou, nic se neděje, prostě se nezobrazí, uživatel to nijak nepostřehne. Načítání statistik je kompletně odděleno od načítání samotného detailu, uživatel tak nečeká na případné zdlouhavé načítání dat.

8.5.1 Sociální sítě

Co se týká sociálních sítí, aktuálně je na detailu serveru vždy zobrazeno tlačítko pro sdílení na Facebooku a Twitteru. Tato tlačítka automaticky generují text obsahující adresu serveru a reklamní text, pomocí kterého by po sdílení měl uživatel nalákat další hráče. U severu se také udržují odkazy na facebookové stránky, přes které se může uživatel dostat až na samotnou stránku daného herního portálu.

8.5.2 Statistiky

Samotný sběr statistik, který provádí monitorovací aplikace není tou poslední částí monitoringu. Tou asi nejdůležitější částí tohoto procesu je interpretace těchto dat. Každý server by v případě získávání statistik každých 10 minut a udržování jich po dobu 14 dní měl obdržet až 2000 hodnot pro každou sbíranou informaci. Samozřejmě, takovýto počet hodnot je obrovský a nějaká orientace v takovémto výpisu by byla opravdu ne příliš dobrá. Pro tyto účely jsem se rozhodl z každé sbírané informace grafy této statistiky za posledních 14 dní. Po načtení detailu serveru se ze serveru dalším požadavkem na API stáhnou veškeré statistiky, což je v aktuálním stavu aplikace až cca 6000 objektů. Každý takovýto objekt obsahuje datum a čas statistiky a hlavně hodnotu této statistiky. Samotné statistiky jsou staženy jako pole JSON objektů.

Nad těmito staženými daty se poté provádí určitá redukce dat, čímž provádím určité zosobnění výsledků. Z klientské aplikace se zjistí aktuální klientský čas a následně jsou klientovy vypsány aktuální statistiky serveru a také porovnání s minulými dny. Vypíše se zde nejvyšší dostupná hodnota za posledních 14 dní ve stejnou hodinu, ve kterou se uživatel snaží získat statistiky. Toto platí však pouze pro počet hráčů. Důležitou částí tohoto procesu je právě redukce těchto dat. Klientovi může v *nejhorším* možném případě dorazit až 6000 hodnot, 2000 pro každou statistiku. Takové množství vykreslovat opravdu

[4] perex – krátký popisek

není vhodné a klientovi toho příliš neřekne než, že pouze vypíše holá data. Navíc orientovat se v takovém grafu by bylo šílené. Proto je provedena nejprve agregace těchto výsledků - získají se statistiky z aktuální hodiny jednoduchým filtrem. V druhé řadě jsou statistiky shlukovány do skupiny, podle dne ve kterém byly vytvořeny. Nad takovýmito daty se již dá pracovat. Tento malý filtr dokáže z 2000 hodnot odfiltrovat circa 1800. Po shlukování nám zbude 15 polí po 6 hodnotách. Tyto hodnoty jsou poté pro účely různých funkcí různě počítány. Celkový proces filtrace a procesu porovnání dat s reálným časem je blíže popsán na výpisu 8.8.

Vykreslené statistiky je možné vidět na výpisu 8.2.

```
processStats = (data) => {
    let _keys = Object.keys(data);
    let values = Object.values(data);
    let stats = [];

    _keys.map((key, id) => {
            stats[id] = {title: name, key: key, items: values[id]};
            const currentHour = new Date().getHours();
            const statsFromCurrentHour = stats[id].items.filter(d => new
                Date(d.date).getHours() === currentHour);
            let filteredArray = [];
            const data = groupBy(statsFromCurrentHour, it => new Date(it.
                date).getUTCDate());
            data.forEach(key => {
                filteredArray.push({
                        date: key[0].date,
                        avg: Math.round(key.reduce((prev, cur) => prev + cur.
                            value, 0) / key.length),
                        max: Math.round(key.reduce((max, p) => p.value > max
                            ? p.value : max, key[0].value)),
                    });
            });
            stats[id]['filteredArray'] = filteredArray;
        }
    });
    return stats.filter(value => Object.keys(value).length);
};
```

Výpis 8.8: Ukázka části filtrace vstupních dat

8.5.3 Generátor widgetů

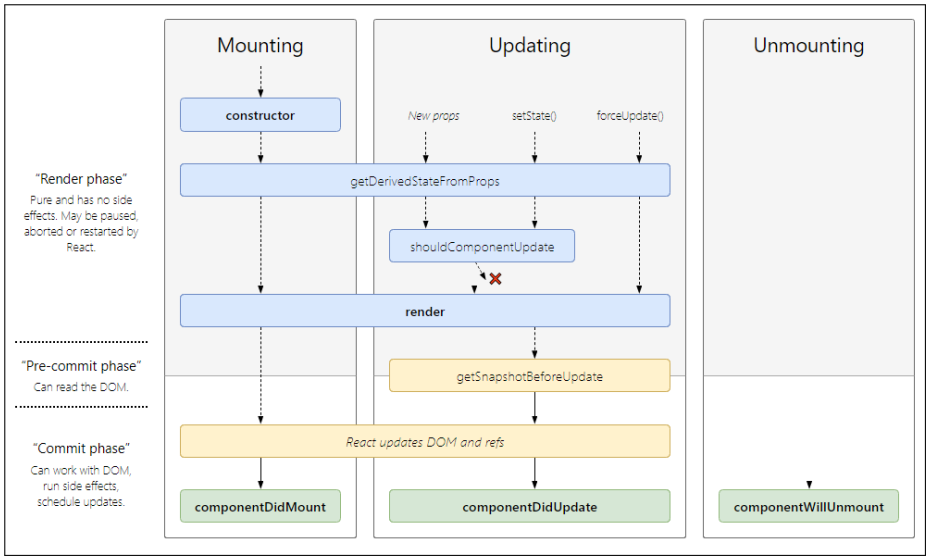
Rozhodl jsem se také pro zjednodušení a zvýšení popularity nabídnout automatické generování webových widgetů[5] pro získávání stavu serveru. Tento widget se dá získat z uživatelského rozhraní, ve kterém si uživatel vybere jeden ze svých serverů a jazyk do kterého chce widget vygenerovat (momentálně je podporován pouze jazyk PHP). Aplikace mu pak nabídne sama stažení souboru s daným widgetem. Generování a stažení takových souborů není v Reactu vůbec složité, je však důležité dávat pozor na escapování[6] hodnot.

```
let data = `<?php class Widget() {
    .
    .
    public function render()
    {
        .
        .
    }
}

$widget = new Widget();
echo $widget->render()

?>`;

const file = new Blob([data], {type: 'text/plain'});
let fileURL = window.URL.createObjectURL(file);
let link = document.createElement('a');
link.href = fileURL;
link.setAttribute('download', "Widget."+extension);
link.click();
```

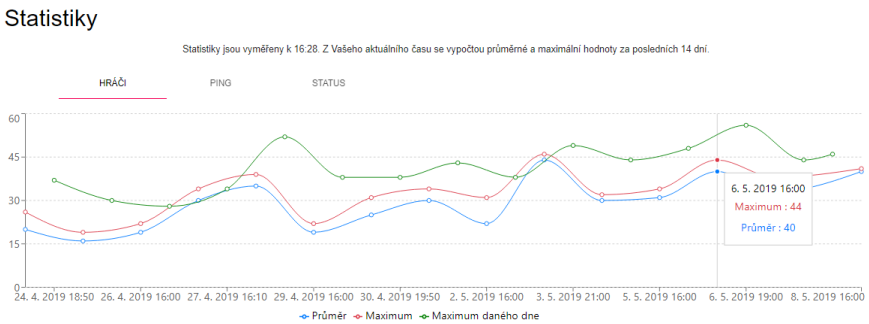
Výpis 8.9: Ukázka generování widgetů pro externí webové stránky

[5] widget – miniaplikace, kterou lze vložit přímo do webu

[6] escapování je slangový výraz pro `escape sequence`, které umožňují do řetězců vkládat speciální znaky



Obrázek 8.1: Životní cyklus komponenty v Reactu[7]



Obrázek 8.2: Interpretace statistik v uživatelském rozhraní

Kapitola 9

Testování služby

Součástí práce je i značné testování služby. V rámci vývoje aplikace jsem se spíše řídil W-modelem vývoje kdy jsem při vývoji postupoval po jednotlivých částech, které jsem postupně navrhl, postupně je implementoval a zároveň i testoval. V poslední fázi jsem z valné většiny dost věcí ověřoval a přepisoval. Nejdůležitějšími prvky, které byly v práci nutné otestovat byly nejen funkčnost samotného API, ale i omezení požadavků na přístup pouze z určitých domén, řádné otestování přihlašování a odhlašování a hlavně otestování přidávání serverů. Dále se tedy zaobírám popisem postupů jednotlivých testů. Veškeré testování probíhalo manuálně s využitím doprovodných testovacích nástrojů pro komunikaci s REST API službou.

9.1 Testování API

Při implementaci aplikačního rozhraní docházelo po každém naprogramovaném přístupovém bodu řádné testování. Důležité bylo nejen ověřit funkčnost výstupu, ale i ověřit správné výstupy jednotlivých modelů. Uživatel by neměl obdržet jak v dotazech a následných odpovědích ze serveru tak i v uživatelském rozhraní data, která mu nenáleží. Do těchto dat spadají informace o pozici některých služeb, spadají sem případná hesla, která by mohla být potřebná pro získávání stavů serverů a samozřejmě sem spadají interní údaje jako hašovaná hesla. Toto vše se však dá vcelku jednoduše ve frameworku Yii vyřešit metodou *fields*, která jasně určí, které atributy a data lze pro daný model zaslat. Nesmím také opomenout fakt, že bylo nutné zasílat informace o statistikách spolu s modely serveru a dále i generovat aktuální stav serveru, který se v databázi neudržuje. Řádně otestována byla i CORS pravidla, o nichž je více v sekci 6.3, která byla nutná z důvodu komunikace mezi různými doménami. Samotné aplikační rozhraní běží na doméně: `https://api.server-list.cz`, kdežto uživatelské rozhraní má doménu: `https://server-list.cz`. Pro každý požadavek, který na server dorazí bylo nutné zařídit, že se zašlou korektní informace o povoleném přístupovém rozhraní.

Pro testování přístupových bodů jsem využíval aplikace Restlet Client, která slouží pro zasílání požadavků na vybrané přístupové rozhraní. Umožňuje také vybírat typy metod a dokonce vkládat proměnné do HTTP hlaviček.

9.2 Testování autorizačních služeb

Testování autorizačních služeb bylo jednou z náročnějších částí, ale zdaleka ne tou nejnáročnější. Jelikož je uživatel přihlašován vlastním systémem přihlašovacích tokenů, bylo nutné řádně otestovat, že se data uživatelů nedostanou do špatných rukou. Každý přístupový bod který provádí jakoukoliv úpravu dat nebo získává data citlivá, ke kterým by se nikdo jiný než ten správný uživatel neměl dostat, ověřuje správnost přihlašovacího tokenu. Za účelem testování jsem také zkoumal možnosti zneužití z cizích / podvržených webů, testoval a experimentoval s možnostmi posunu času u klienta, kvůli kterým může docházet k chybám autorizace. I tyto případy byly ošetřeny právě kontrolou všech tokenů na přístupovém rozhraní.

9.3 Testování přidávání serverů

Testování přidávání serverů již byla ve výsledku hračka. Samotné přidávání se opět řídí snadnými podmínkami – uživatel musí být přihlášen, musí mít platný přihlašovací token, musí přidávat server, který ještě neexistuje. Tyto podmínky vždy kompletně ošetřuje přístupové rozhraní, které vrací jasné výstupy chyb na vstupu. Tímto tak lze nejen ušetřit čas při vývoji ověřování formulářů a vstupů, ale také se takto zvyšuje přehlednost kódu – validace je psána pouze v jednom jazyce a jedním postupem, není nutné tak psát více variant validace, což a to hlavně šetří práci programátora.

9.4 Testování monitorovací aplikace

Testování monitorovací aplikace probíhalo v několika fázích. Původní systém byl navržen pouze pro jeden server. Dále jsem prováděl rozšíření na sledování circa 100 serverů, jejich monitoring probíhal stabilně po cca 14 dní. Po úspěšném ověření funkčnosti nad rozumnou várkou serverů, byly mi poskytnuty další servery, jejich počet se celkem vyšplhal na 800. S tímto počtem tedy aplikace prováděla od února 2019 sledování stavu těchto serverů ve dvouhodinových intervalech. Po následném rozšiřování za účelem využití nedávných výsledků jako momentálních výsledků došlo k přidání 10 minutových intervalů, ve kterých aplikace měří servery a následně došlo k rozšíření na 3-stupňový monitoring, kdy se první dva pokusy o monitoring neukládají, ukládá se až třetí pokus. Tyto všechny stavy byly důkladně testovány, bylo testování automatické čištění dat, aby nedošlo k úplnému zahlcení databáze. Kompletní seznam všech serverů nad kterými bylo provedeno testování je však příliš dlouhý, je tedy přiložen spolu s dalšími daty v příloze A. Samozřejmě, jak bylo zmíněno v sekci 8.5, monitorovací aplikace se neskládá pouze z části generující data, ale i z části provádějící jejich interpretaci a výpis. V detailu serveru jsem tedy musel sepsat speciální metody, které redukují data na výstupu. Důležité bylo otestovat různé výstupy pro různé statistiky. Data jsem reálně porovnával s aktuálními výsledky a kontroloval pravost výsledků.

Kapitola 10

Závěr

Hlavním cílem práce bylo vytvořit portál serverů s funkční monitorovací aplikací s možností přidávání serverů přes vzdálené API. Myslím si, že implementace takovéhoto systému se podařila více než zdárně. Monitorovací systém provádí sledování v 10 minutových intervalech, zjišťuje stavy serverů nezávisle na funkčnosti webového rozhraní. Aplikační rozhraní bylo také zdárně implementováno. Součástí a doplňkem bylo sdílení na sociálních sítích a notifikační systém. Webové rozhraní interpretuje správně veškeré statistiky a správně je i vypisuje. Nad samotnými daty je provedena zdařilá filtrace. Je nutné zmínit, že se filtruje přibližně 4 - 6 000 záznamů pro každý herní server, což v konečném počtu dává přes 1,5 milionu záznamů v databázi pro všechny servery. Není to malé číslo a operace nad tímto počtem dat, jako třeba vyfiltrování nejobsazenějšího serveru není úplně nejjednodušší operací. SQL dotazy jsem však na několikrát přepisoval, filtrace dat je tak opravdu rychlá jak na back-endové části, tak i na front-endové.

Pro vytvoření práce jsem využil nejen moderních vývojových přístupů, ale také i aktuálních technologií, přičemž s mnoha z nich jsem pracoval poprvé. Pro tvorbu front-endové části aplikace jsem využil JavaScriptové knihovny React, ve které jsem se učil. Samotnému programování předcházelo načtení si oficiální dokumentace ale i přečtení si anglické knihy Learning React, která vcelku bravurně vysvětluje základy tohoto jazyka. Ke konci vývoje aplikace jsem však zjistil, že spoustu věcí, které jsem řešil svým způsobem, jsem mohl vyřešit mnohem efektivněji a rád bych se do budoucna věnoval přepsání části aplikace. Co se týká monitorování, snažil jsem se spojit znalosti z prostředí aplikace Zabbix a znalosti z herního světa. Ve výsledku si myslím, že je aplikace sepsána kvalitně. Pro 800 serverů sice aplikace bohatě postačuje, rád bych se však v budoucnu věnoval rozšíření aplikace na nějaký modulární systém, kdy budou jednotlivé servery monitorovány v separátních vláknech. Předběžně jsem si spočítal, že mi přibližně při 2000 serverech bude docházet ke zpožděním s monitorovanými servery, čemuž bych právě zabránil rozdělením procesů do různých vláken.

Aplikace má také spoustu míst, kam lze expandovat a již při nynější konzultaci s hostingem jsem dospěl k závěru, že bych rád vytvořil pokročilé možnosti filtrování serverů, zvýšil integraci sociálních sítí a samotné sociální sítě propojil s notifikačním systémem. Při každém novém příspěvku na stránce daného profilu by tak mohla přijít uživatelům notifikace o novince na Facebooku pro daný server. Rád bych také uživatelům nabídl možnost tvorby vlastních URL k serverům, čímž zjednoduším přístup uživatelům k jejich statistikám.

Když se ohlížím za časovým rozložením práce, určitě mohu říci, že nejvíce času jsem strávil nad monitorovací aplikací a front-endovou částí aplikace. Samotný sběr statistik bylo nutné řádně otestovat, front-endovou část jsem musel rozumným způsobem nadesignovat a také napojit na aplikační rozhraní, což zabralo asi nejvíce času. Osobně je mi bližší logika

než-li design a tak jsem dle mého názoru strávil zbytečně více času návrhy a designováním aplikace než bylo třeba. Rozhodně jsem si vyzkoušel možnosti monitorovacích knihoven, otestoval jsem varianty dostupné na trhu a myslím si, že jsem vytvořil aplikaci, která je relativně jednoduše schopná jim konkurovat.

Literatura

[1] Banks, A.; Porcello, E.: *Learning React: functional web development with React and Redux*. O'Reilly Media, 2017, iSBN: 1491954590.

[2] Dayley, B.: *Learning AngularJS*. Addison-Wesley, 2015, iSBN: 978-0-134-03454-6.

[3] Netflow, nová éra monitorování počítačových sítí. Online; [navštíveno: 13.04.2019]. URL https://www.flowmon.com/cs/solutions/use-case/netflow-ipfix

[4] Gilfillan, I.: *Myslíme v MySQL4*. Grada Publishing, a.s., 2003, iSBN: 80-247-0661-X.

[5] Herout, P.: *Učebnice jazyka Java*. Nakladatelství KOPP, 2015, iSBN: 978-80-7232-392-2.

[6] Cross-origin resource sharing - Wikipedia. Online; [navštíveno: 1.4.2019]. URL https://en.wikipedia.org/wiki/Cross-origin_resource_sharing

[7] Maj, W.: React lifecycle methods diagram. Online; [navštíveno: 3.5.2019]. URL http://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/

[8] About MariaDB - MariaDB.org. Online; [navštíveno: 10.4.2019]. URL https://mariadb.org/about/

[9] Matoušek, P.: *Síťové služby a jejich architektura*. Publishing house of Brno University of Technology VUTIUM, 2014, ISBN 978-80-214-3766-1, 396 s. URL http://www.fit.vutbr.cz/research/view_pub.php.cs.iso-8859-2?id=10567

[10] NoSQL Databases Explained. Online; [navštíveno: 15.04.2019]. URL https://www.mongodb.com/nosql-explained

[11] About Nagios. What is Nagios? Online; navštíveno: 11.04.2019]. URL https://www.nagios.org/about/

[12] Seznámení s Nette Frameworkem. Online; [navštíveno: 16.04.2019]. URL https://doc.nette.org/cs/3.0/getting-started

[13] About | Node.js. Online; [navštíveno: 11.04.2019]. URL https://nodejs.org/en/about/

[14] ntopng – ntop. ntop – High Performance Network Monitoring Solutions based on Open Source and Commodity Hardware. Online; [navštíveno: 12.04.2019]. URL https://www.ntop.org/products/traffic-analysis/ntop/

[15] Introduction to Oracle Database. [Online; navštíveno 5.4.2019].
URL
https://docs.oracle.com/cd/E11882_01/server.112/e40540/intro.htm#CNCPT939

[16] Otwell, T.: Laravel - The PHP Framework For Web Artisans. Laravel. Online; [navštíveno: 16.4.0219].
URL https://laravel.com/docs/4.2/introduction

[17] PHP: History of PHP - Manual. Online; [navštíveno: 16.4.2019].
URL https://www.php.net/manual/en/history.php.php

[18] PHP: What is PHP? - Manual. Online; [navštíveno: 11.04.2019].
URL https://www.php.net/manual/en/intro-whatis.php

[19] Pilgrim, M.: *Ponořme se do HTML5*. CZ.NIC, z.s.p.o., 2015, iSBN: 978-80-905802-6-8.

[20] What is Zabbix [Zabbix Documentation 4.2]. Online; [navštíveno: 15.04.2019].
URL https://www.zabbix.com/documentation/4.2/manual/introduction/about

[21] Žára, O.: *JavaScript: programátorské techniky a webové technologie*. Computer Press, 2015, iSBN: 978-80-251-4573-9.

Příloha A

Obsah DVD

- soubor **readme.txt** obsahující informace o projektu, návod k instalaci a přístupové údaje
- složka **latex**
  - zdrojový kód technické zprávy
- složka **server-list.cz**
  - soubor **readme.txt** obsahující návod ke spuštění projektu a informace o repozitáři a projektu
  - zdrojové soubory front-endové aplikace
  - konfigurační soubory
- složka **server-list.cz-api**
  - souboor **readme.txt** obsahující návod ke spuštění projektu a informace o repozitáři a projektu
  - zdrojové soubory back-endové části aplikace včetně monitorovací aplikace
  - konfigurační soubory
- složka **db**
  - soubor **export.sql** obsahující data jednoho testovacího uživatele a sadu testovacích dat pro spuštění monitoringu nad nimi
- složka **data**
  - soubor **servery.xls** obsahující výpis všech serverů nad kterými byli prováděn monitoring

Příloha B

Dokumentace aplikačního rozhraní

- webové rozhraní aplikačního rozhraní obsahuje dokumentaci přístupových bodů na adrese: https://api.server-list.cz
- koncové body aplikačního rozhraní jsou vždy prefixovány adresou: https://api.server-list.cz/v1

B.1 Přístupový bod: User

- **POST /register** Vytvoření nového uživatelského účtu. V těle požadavku je očekáván přihlašovací token a data uživatele.
- **POST /login** Přihlášení uživatele, v těle zprávy je očekáváno uživatelské jméno a heslo. Uživatel zpět obdrží přihlašovací token.
- **POST /relogin** Znovu přihlášení uživatele - prodlouží expiraci stávajícího tokenu. V těle požadavku je očekáván přihlašovací token.
- **POST /server/<id>** Kontrola vlastnictví serveru s ID: **<id>**. V těle požadavku je očekáván přihlašovací token.
- **POST /logout** Odhlášení uživatele z aktuálního zařízení. V těle požadavku je očekáván přihlašovací token. Tento token je ponechán k expiraci a nelze jej použít.
- **POST /logoutAll** Odhlášení uživatele ze všech zařízení. V těle požadavku je očekáván přihlašovací token. Veškeré tokeny uživatele jsou ponechány k expiraci a nelze je použít.
- **POST /notifications** Získání všech notifikací, které uživateli dorazily. V těle požadavku je očekáván přihlašovací token a data uživatele.
- **POST /notification/<id>** Označení notifikace s ID: **<id>** jako přečtené. Notifikace již nebude vyskakovat jako nepřečtená. V těle požadavku je očekáván přihlašovací token.

B.2 Přístupový bod: Service

- **GET /services** Výpis všech služeb dostupných pro přidávání serverů v aplikaci.
- **GET /services/<id>** Detailní informace o službě s ID: **<id>**.

B.3 Přístupový bod: Server

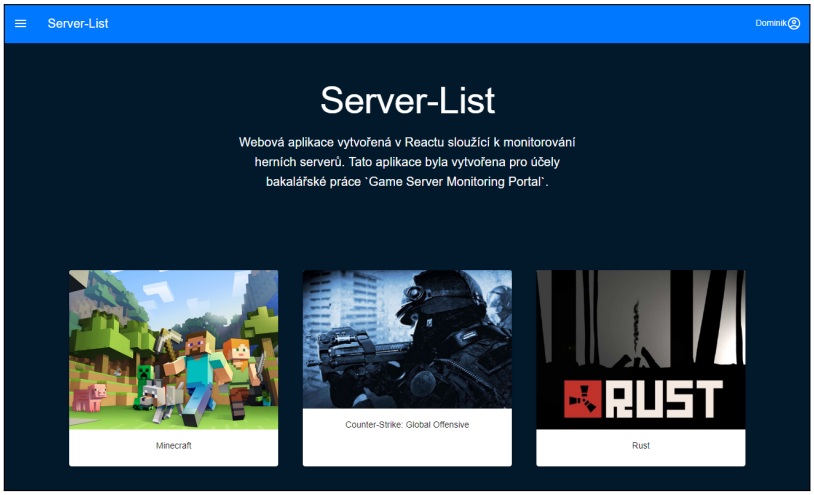
- **GET /services/<id>/servers** Výpis všech serverů v aplikaci.
- **GET /services/<id>/servers/<server_id>** Detailní informace o serveru s ID: **<server_id>**. ID služby je určeno prvkem: **<id>**. Jedná se o stromovou strukturu.
- **POST /services/<id>/servers** Vytoření nového serveru u služby určené ID: **<id>**. V těle zprávy je očekáván přihlašovací token a data serveru.
- **PATCH /services/<id>/servers/<server_id>** Úprava serveru s ID: **<server_id>**. ID služby je určeno prvkem: **<id>**. V těle zprávy je očekáván přihlašovací token a upravená data serveru.
- **DELETE /services/<id>/servers/<server_id>** Smazání serveru s ID: **<server_id>**. ID služby je určeno prvkem: **<id>**. V těle zprávy je očekáván přihlašovací token.

B.4 Přístupový bod: Stats

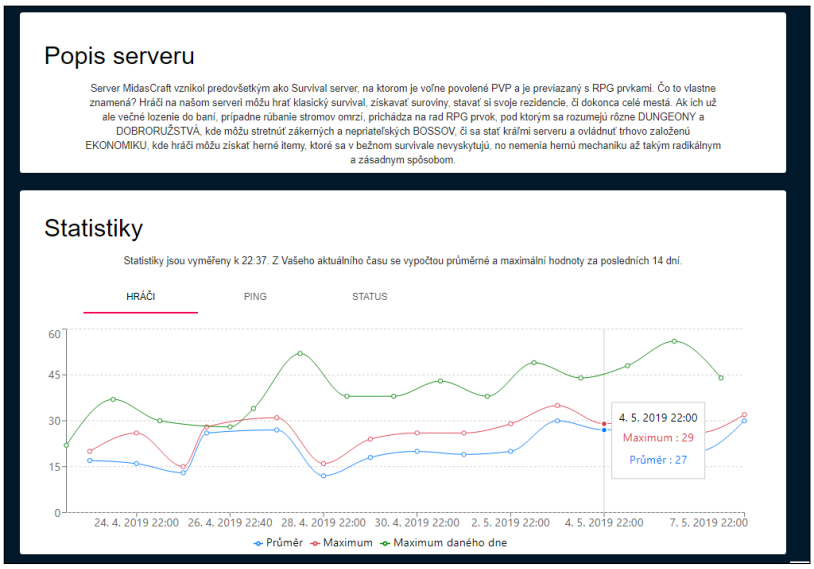
- **GET /services/<id>/servers/<server_id>/stats** Výpis všech dostupných statistik k serveru s ID: **<server_id>**zjištěných z databáze. Je vráceno pole obsahující pole objektů statistik v závislosti na počtu sledovaných hodnot.

Příloha C

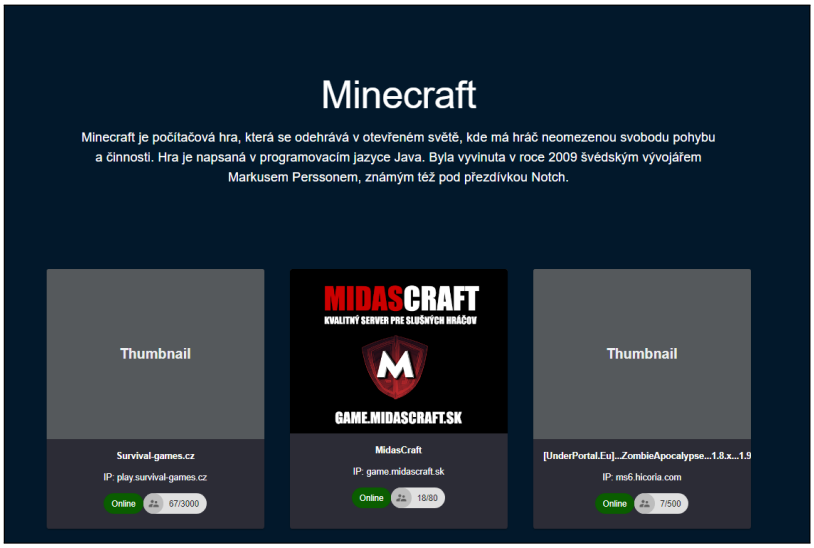
Fotografie a snímky z průběhu vývoje a finální aplikace



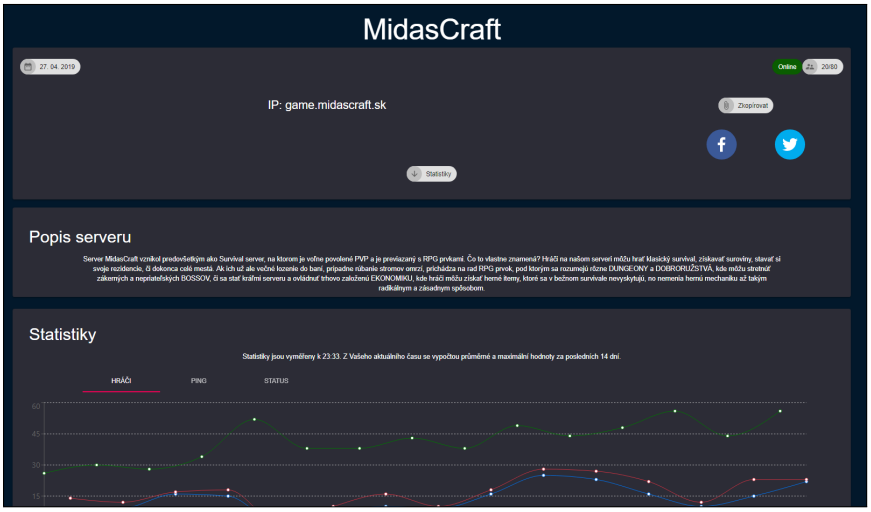
Obrázek C.1: Výpis všech služeb



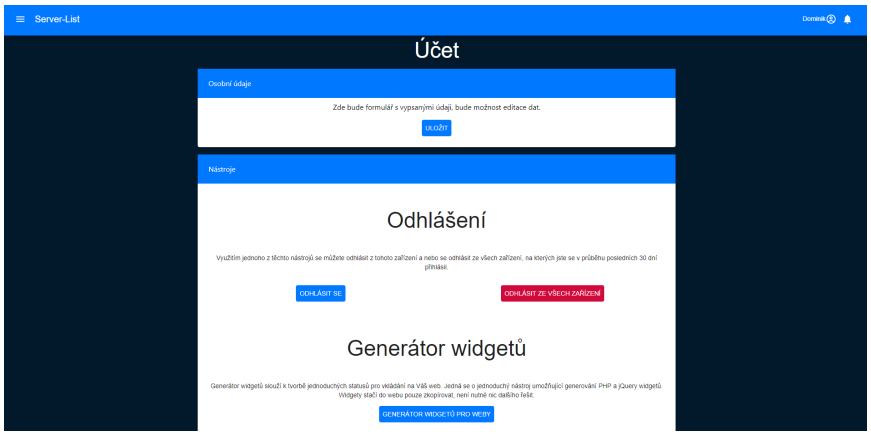
Obrázek C.2: Výpis statistik a popisu serveru



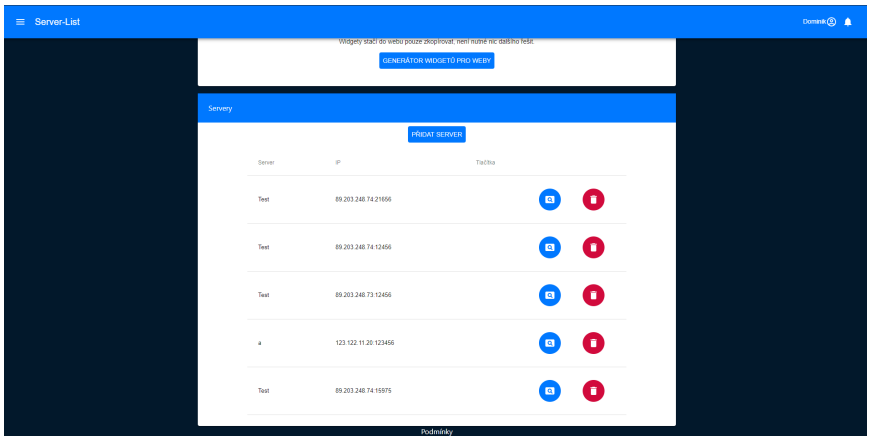
Obrázek C.3: Výpis serverů jedné služby



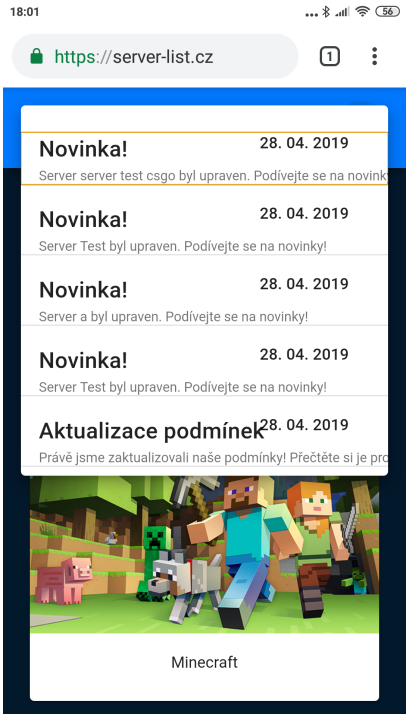
Obrázek C.4: Detail serveru



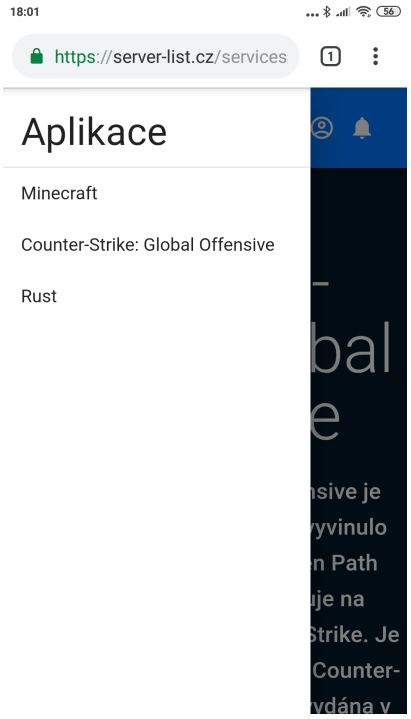
Obrázek C.5: Výpis nastavení a generátor widgetů



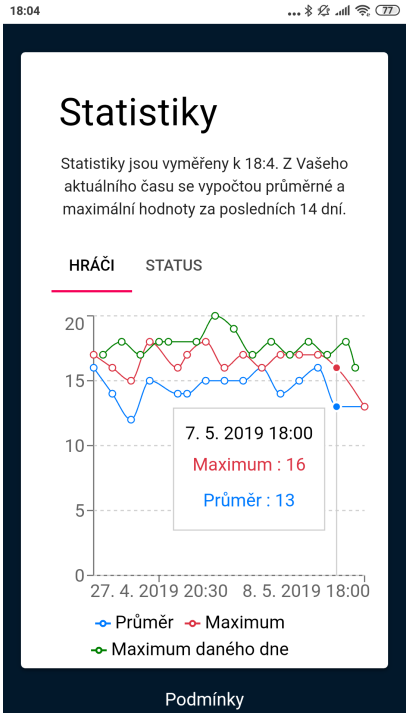
Obrázek C.6: Výpis serverů v nastavení



(a) Mobilní aplikace a notifikace



(b) Panel dostupných aplikací



(c) Responzivita mobilních statistik