Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Jádro aplikace, rozhraní webových služeb a prezentační vrstva pro systém PartyBoard

Plzeň, 2016 Bc. Antonín Neumann

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 12. května 2016

Bc. Antonín Neumann

Poděkování

Chtěl bych poděkovat Ing. Jakubovi Daňkovi za odborné vedení a rady při psaní této diplomové práce. A dále své rodině a přítelkyni za jejich podporu v průběhu celého mého studia.

Abstrakt

Tato diplomová práce popisuje analýzu a tvorbu části servisně orientovaného systému Partyboard. Naším cílem je vytvořit komplexní projekt, který bude schopný přijímat, odesílat, zobrazovat a spravovat zprávy od návštěvníků nočních klubů a barů.

Celý projekt je rozdělen na dvě části. Každá část je realizována jako samostatná diplomová práce a obsahuje několik samostatných aplikací. Tyto aplikace dohromady budou tvořit výsledný systém.

Ve své části se zaměřím na analýzu servisně orientovaných architektur, na analýzu a návrh REST API, které bylo pro komunikaci mezi ostatními aplikacemi zvoleno. A v závěru na zhodnocení a výběr MVC frameworku, který bude pohánět prezentační a administrační web projektu.

Druhou polovinou projektu se zabývá diplomová práce Bc. Jana Nováka s názvem *Databázový systém, mobilní aplikace a business plán pro aplikaci Partyboard*, v níž se věnuje zejména návrhu a implementaci databáze a business plánu celého projektu. Jeho závěrem je mobilní aplikace, která komunikuje, s mnou vytvořeným API, a umožňuje uživatelům číst a odesílat zprávy.

Abstract

This thesis describes the analysis and creation of service-oriented system PartyBoard, which aims to create a comprehensive project that will be able to receive, send, view and manage messages from visitors to nightclubs and bars.

The whole project is divided into two parts, with each part is realized as a separate thesis and includes several individual applications which will form the final system.

In my part I will focus on the analysis of service-oriented architectures. Further analysis and design REST API, which was for communication between other applications selected. And last on the evaluation and selection of the MVC framework that will drive the presentation and administration web project.

The second half of the project deals with the thesis Bc. Jan Novak named Database system, mobile application and business plan for an application PartyBoard. Which pays particular attention to the design and implementation of database and business plan of the project. And the last part is a mobile application that communicates with me and created API allows users to read and send messages.

Obsah

1 Úvod

Cílem mojí diplomové práce je realizace části projektu nazvaného PartyBoard. Jedná se o projekt, jehož hlavním účelem je příjem, správa a vizualizace SMS zpráv na televizních obrazovkách, anebo projekčních jednotkách přímo v nočních klubech, případně v aplikaci na mobilních zařízeních.

Smyslem celého projektu je nabídnout klubům možnost nového sociálního propojení, soutěžení a propagace. Pro uživatele je hlavním přínosem možnost seznámení s jinými návštěvníky klubu.

Na této aplikaci spolupracuji se svým kolegou Bc. Janem Novákem. Mojí částí je výběr technologie, návrh a realizace aplikačního rozhraní (API) pro servisně orientované webové služby, výběr vhodného frameworku pro provoz prezentační webové stránky a nakonec webová aplikace určená pro zobrazení zpráv na velkoplošných obrazovkách přímo v klubech.

Návrh API se zakládá na architektuře REST se zprávami ve formátu JSON. Implementace je následně vytvořena na platformě NodeJS s využitím frameworku ExpressJS. Výběr webového frameworku musí reflektovat potřeby systému a umožňovat snadný budoucí rozvoj prezentačních webových stránek Partyboard.cz.

Webová aplikace tzv. „Board“ je určena pro zobrazení jednotlivých SMS zpráv typicky na televizní obrazovku či promítací plátno umístěné přímo v nějakém klubu, baru nebo restauraci.

Cílem této diplomové práce je tedy porovnání a výběr technologií, které by umožňovaly systém snadno udržovat a spravovat. Implementovat příjem SMS zpráv. A dále návrh architektury celého projektu tak, aby jej bylo možné rozvíjet a doplňovat o nové funkce i v budoucnu po jeho nasazení v praxi.

2 Architektury pro tvorbu servisně orientovaných systémů

Servisně orientovaná architektura (SOA) byla pro systém PartyBoard zvolena, protože součástí toho projektu je několik samostatných aplikací, které běží na různých systémech a platformách. Níže v textu je popsaná charakteristika této architektury a její hlavní přednosti.

2.1 Servisně orientovaná architektura

Servisně orientovaná architektura (SOA) je paradigma pro organizování a distribuovaných schopností aplikací nebo programů, které mohou být pod kontrolou různých vlastníků [1].

Klíčové prvky SOA můžeme definovat jako službu, která je:

- autonomní (každá služba je navržena, vyvíjena a provozována nezávisle),
- distribuovatelná (služba může být umístěna kdekoliv na internetové síti),
- volně vázaná (každá služba je nezávislá na ostatních a může být nahrazena nebo aktualizována bez poškození aplikací, které ji využívají – pokud je zachováno původní rozhraní),
- sdílí schéma a kontrakt (rozhraní), ale nikoli implementaci.

2.1.1 Služba

Pro správné používání principů SOA je nutné definovat výše často zmiňovaný termín služba (angl. service).

Podle organizace The Open Group [2] je služba definována jako logická reprezentace opakovaných *business*[1] aktivit, které mají specifický výstup, a zároveň splňuje následující podmínky:

1 Anglický výraz business v tomto kontextu není vhodné překládat, protože jeho význam do českého jazyka překládaný jako obchodní není pro můj účel zcela korektní. Alternativně by bylo možné použít slovo doménový, ale zůstanu raději u původního anglického výrazu.

- Je soběstačná.
- Může být složena z jiných služeb.
- Je uzavřená (angl. black box) pro konzumenty služby.

2.1.2 Konzument služby

Za konzumenta služby můžeme označit aplikaci, jinou službu nebo uživatele, který od služby na základě jejího rozhraní a svého vstupu požaduje nějaký výstup.

Máme-li například službu poskytující adresy zaměstnanců, potom aplikace, která tyto údaje umí vytisknout na papír, je konzumentem této služby.

2.2 SOAP

SOAP (Simple Object Access Protocol) je protokol určený pro výměnu zpráv v decentralizovaném a distribuovaném prostředí, založený na XML (Extensible Markup Language) formátu. Protokol SOAP 1.2 je standardizován jako doporučení *(angl. W3C Recommendation)* vydané konsorciem W3C v roce 2007 [3].

Hlavní výhodou protokolu SOAP je možnost komunikace přes různé běžně dostupné síťové protokoly, například HTTP (Hypertext Transfer Protocol), SMTP (Simple Mail Transfer Protocol), TCP (Transmission Control Protocol) anebo UDP (User Datagram Protocol). Díky tomu nemá potíže s firewally[2] jako například DCOM (Distributed Component Object Model) vyvinutý společností Microsoft, jehož komunikaci je nutné na většině firewallech explicitně povolit.

Každý SOAP požadavek musí podle specifikace obsahovat obalující element *Envelope* a v něm elementy *Body* (povinný) a *Header* (nepovinný). V elementu *Body* musí být specifikováno jaký zdroj (metodu) požadujeme a jeho parametry.

2 Firewall označuje hardwarový nebo softwarový prostředek kontrolující příchozí a odchozí síťovou komunikaci, kterou na základě nastavených pravidel propouští nebo blokuje a zajišťuje tak zabezpečení síťového provozu.

Element *Header* může nést informaci například o tajném klíči, který ověřuje, zda má volající právo ke službě nebo metodě přistupovat.

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
  soap:encodingStyle="http://www.w3.org/2003/05/soap-
encoding">
  <soap:Header>
    <t:Token
      xmlns:t="http://www.example.com/token/"
      soap:mustUnderstand="true">SECRET_KEY</t:Token>
   </soap:Header>
  <soap:Body>
    <m:GetUser xmlns:m="http://www.example.com/users">
      <m:UserId>A14N0139P</m:UserId>
    </m:GetUser>
  </soap:Body>
</soap:Envelope>
```

Zdrojový kód 1: Ukázka SOAP požadavku odesílaného na server

Na výše uvedený požadavek server odpoví následujícím XML souborem.

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
  soap:encodingStyle="http://www.w3.org/2003/05/soap-
encoding">
  <soap:Body>
    <m:GetUserResponse
      xmlns:m="http://www.example.com/users">

      <m:UserId>A14N0139P</m:UserId>
      <m:UserName>Antonín</m:UserName>
      <m:UserSurname>Neumann</m:UserSurname>
      <m:UserGenger>muž</m:UserGenger>
    </m:GetPriceResponse>
  </soap:Body>
</soap:Envelope>
```

Zdrojový kód 2: Ukázka SOAP odpovědi odeslané ze serveru

2.2.1 WSDL

Pro popis webových služeb využívajících protokol SOAP se velmi často používá jazyk WSDL (Web Services Description Language). Tento dokument používá syntaxi jazyka XML, ukázka v příloze B.

Specifikaci WSDL je možné nalézt jako doporučení organizace W3C [4], a to konkrétně ve verzi WSDL 2.0. Umožňuje kromě popisu SOAP protokolu, popisovat také webové služby postavené na architektuře REST.

„WSDL dokument popisuje rozhraní webové služby na syntaktické úrovni, stejně jako .h soubory v jazyce C nebo interface v jazyce Java, tedy jako seznam jmen funkcí/metod, zde zvaných operace, spolu se jmény a typy parametrů a návratových hodnot.“ [5]

2.3 REST

REST (Representational State Transfer) je architektura popsaná v roce 2000 v disertační práci Roy T. Fieldingem [6]. REST je navržený pro distribuované systémy, jež ke komunikaci využívá zejména protokol HTTP. Komunikace probíhá pomocí tzv. konektorů (*angl. connectors*).

Primárními konektory jsou klient a server. Klient, konzument služby, který je vždy iniciátorem spojení (požadavku, *angl. request*). Server je program, který poslouchá příchozí požadavky klientů, zpracuje je a odešle odpověď (*angl. response*).

Dalšími konektory jsou *cache* (umožňují uchovávat odpovědi serveru v mezipaměti), *resolver* (překládá část nebo celý identifikátor zdroje, nejčastěji doménové jméno na IP adresu pomocí protokolu DNS [Domain Name System]) a *tunnel* (ten slouží k předávání komunikace například skrze proxy).

Základní abstrakcí informace je zdroj *(angl. resource)*. Podle práce pana Fieldinga může být takovýmto zdrojem jakákoliv informace, dokument,

obrázek, dočasná služba, kolekce jiných zdrojů, atp. Zdroj je v případě RESTu identifikován pomocí tzv. identifikátoru zdroje, kterým může být URL (Uniform Resource Locator) [7] nebo URN (Uniform Resource Name) [8]. Obě tato identifikační schémata spolu dohromady tvoří URI (Uniform Resource Identifier) [9].

```
scheme:[//[user:pass@]host[:port]][/]path[?query][#fragment]
```

Zdrojový kód 3: Ukázka obecné syntaxe URI [9]

Na rozdíl od jiných typů služeb používá REST širší spektrum metod protokolu HTTP. Konkrétně se jedná o metody *GET*, *POST*, *PUT* a *DELETE*, které odpovídají 4 základním operacím CRUD (Create-Read-Update-Delete [česky vytvoření, čtení, úprava a smazání]) prováděných nad nějakým zdrojem.

2.4 REST omezení

REST je odvozený z několika síťově založených architektur [10] a několika omezeních [6], [11], která jsou postupně aplikována na systém (v tomto případě celý WWW [World Wide Web]).

2.4.1 Client-Server

Prvním omezením je použití architektury klient-server, která slouží k oddělení odpovědnosti (*angl. separation of concerns*). Oddělením uživatelského prostředí (klient) od úložiště dat (server) můžeme zlepšit přenositelnost klienta a škálovatelnost serveru. Další výhodou je možnost vyvíjet obě komponenty nezávisle, a to tedy i v různých doménách.

2.4.2 Stateless

Dalším omezením je bezstavovost, tj. požadavek od klienta musí obsahovat všechny informace nezbytné k tomu, aby mu server porozuměl. Z toho vyplývá, že veškerá data vztahující se k dané relaci (*angl. session*) jsou uložena pouze na straně klienta.

Toto omezení zlepšuje spolehlivost a škálovatelnost systému. Spolehlivost se zvýší, jelikož i případná chyba jednoho requestu neovlivní žádnou další. Škálovatelnost zlepší fakt, že server nemusí ukládat žádná relační data a může rychle uvolnit své prostředky pro další požadavek. Rovněž zjednodušuje implementaci serveru.

Nevýhodou tohoto omezení je vyšší síťový provoz. Dochází totiž k přenosu stejných informací s každým požadavkem.

2.4.3 Cacheable

Pro zlepšení propustnosti sítě byla přidána mezipaměť, toto omezení vyžaduje označit data v odpovědi serveru (implicitně nebo explicitně) jako vhodná či nevhodná pro uložení do mezipaměti. Pokud je response označena jako cacheable, může jí klient v případě stejného požadavku použít, jako by se jednalo o normální odpověď serveru.

Výhodou je možnost částečně eliminovat některé interakce a zlepšit efektivitu, škálovatelnost a výkon snížením průměrné doby odezvy při sériích požadavků. Nevýhodou naproti tomu může být snížení spolehlivosti v případě, kdy se data v mezipaměti a na serveru liší.

2.4.4 Uniform interface

REST klade velký důraz na jednotnost rozhraní mezi komponentami. Jednotnost vede ke zjednodušení a lepšímu pochopení celé architektury i jednotlivých interakcí. Jednotné rozhraní definuje další 4 omezení jako svojí součást:

- Identifikace zdroje – zdroj je jednoznačně určený podle URI a samotný zdroj je oddělený od jeho reprezentace (např. konkrétní XML dokument), která je vrácena klientovi.

- Manipulace se zdroji – definuje, že pokud klient drží (např. má uloženou) nějakou reprezentaci zdroje, včetně metadat, je to dostatečné množství informací k tomu, aby s tímto zdrojem mohl manipulovat, (např. ho upravit nebo smazat).
- Samo popisné zprávy – každá zpráva (požadavek) obsahuje dostatek informací pro její zpracování (např. hlavička *Content-Type* pro určení správného parseru).
- Hypermedia jako základ pro stav aplikace – HATEOAS (Hypermedia as the engine of application state) je omezení které definuje, že klienti komunikují se serverem pouze na základě znalostí přijatých od serveru již dříve. V praxi, pokud je pro přenos použitý formát JSON, který neumožňuje přímo uchovávat hypertextové odkazy, obchází definováním API dokumentace (to není zcela validní přístup, ale je velmi častý) nebo různými dohodnutými konstrukcemi.

```
{
  "hyperlink": [
    "self": "http://example.org/foo/bar",
    "next": "http://example.org/foo/bar/2"
  ]
}
```

Zdrojový kód 4: Ukázka použití HATEOAS u JSON zpráv, zdroj: http://restcookbook.com/Mediatypes/json/

2.4.5 Layered system

Vícevrstvý systém umožňuje architektuře, aby byla složena z několika úrovní komponent, přičemž každá ví pouze o komponentách bezprostředně propojených. Tímto způsobem lze například zapouzdřit staré služby, odstínit nové služby od starých klientů, anebo škálovat systém použitím cache serverů blíže ke klientům.

2.4.6 Code-on-demand

Posledním omezením je tzv. kód na vyžádání, které je pouze volitelné, jelikož snižuje přehlednost *(angl. visibility)*. REST umožňuje, aby byla funkcionalita klienta rozšířena pomocí staženého kódu. Tím se zvyšuje rozšiřitelnost systému po jeho nasazení.

2.5 Datový formát pro výměnu zpráv

Pro výměnu dat můžeme v případě REST architektury použít různé datové formáty. Tyto formáty jsou většinou textové (jako celý HTTP protokol), snadno strojově zpracovatelné a do jisté míry i zaměnitelné. Volba konkrétního formátu je tak většinou závislá na použitém programovacím jazyce, ostatních již provozovaných službách v organizaci a na osobních preferencích programátora či týmu, který dané API navrhuje.

2.5.1 JSON

JSON (JavaScript Object Notation) je jednoduchý textový formát pro výměnu dat specifikovaný v RFC 7159 [12] a standardizovaný v ECMA-404 [13].

Tento formát je založen na objektech programovacího jazyka Javascript, a jedná se tedy o strukturu klíč-hodnota (*angl. Key-Value*). Jsou zde ovšem některé rozdíly. Formát JSON musí splňovat tyto podmínky:

- Všechny klíče musí být zapsány ve dvojitých uvozovkách.
- Povolené datové typy pro hodnoty jsou *string* (textový řetězec), *number* (číslo, nerozlišuje se celé číslo od desetinného), *object* (objekt), *array* (pole hodnot) a hodnoty *true*, *false* a *null* (prázdná hodnota).
- Řetězec musí být zapsán ve dvojitých uvozovkách.
- Číslo smí být zapsáno pouze v desítkovém formátu, pro oddělení desetinné části musí být použita tečka.

- Objekt je zapsán mezi složené závorky {} a pole mezi hranaté závorky [].
- Jednotlivé hodnoty objektu i pole jsou oddělené čárkou, za poslední hodnotou čárka být nesmí.

```
{
  "name": "Antonín",
  "lastname": "Neumann",
  "email": ["neumann@students.zcu.cz", "neumann@example.cz"],
  "password": "SECRET",
  "address": {
    "street": "Ulice",
    "number": 123,
    "town": "Plzeň"
  }
}
```

Zdrojový kód 5: Ukázka třídy User ve formátu JSON

2.5.2 XML

XML je značkovací jazyk vyvinutý z univerzálního jazyka SGML (Standard Generalized Markup Language). Specifikace XML 1.0 byla vydána, jako *W3C Recommendation* organizací W3C v roce 2008 [14].

XML dokument se skládá z různých elementů (značek, tagů, *angl. tags*), každý může pro svůj účel vytvářet tyto elementy libovolně. Pokud ovšem chceme, aby náš dokument dokázala parsovat běžná knihovna některého programovacího jazyka, musíme dodržet definovaná pravidla. XML splňující tato pravidla se nazývá správně strukturovaný *(angl. well-formed)*:

- Každý XML dokument začíná deklarací.
- XML dokument obsahuje vždy jen jeden kořenový element a všechny ostatní jsou obsaženy v jeho těle.
- Každý element je řádně ukončen (párový i nepárový).
- Je-li element obsažený v těle jiného dokumentu, pak je zde celý.

- Jednotlivé elementy, resp. jejich značky, se nesmějí křížit.

Dále je možné definovat přesnou strukturu dokumentu pomocí XML schématu, označovaného zkratkou XSD (XML Schema Definition), a to hlavně z důvodu, aby odesílatel i příjemce chápali data obsažená v dokumentu stejným způsobem.

Dále s použitím XML úzce souvisí i WSDL, zmíněné v kapitole 2.2.1, které se používá pro popis rozhraní webových služeb provozovaných na protokolu SOAP i postavených na architektuře REST.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<User>
  <name>Antonín</name>
  <lastname>Neumann</lastname>
  <emailList>
    <email>neumann@students.zcu.cz</email>
    <email>neumann@example.cz</email>
  </emailList>
  <password>SECRET</password>
  <address>
    <street>Ulice</street>
    <number>123</number>
    <town>Plzeň</town>
  </address>
</Users>
```

Zdrojový kód 6: Ukázka třídy User ve formátu XML

2.5.3 YAML

Formát YAML (YAML Ain't Markup Language) je formát určený pro serializace dat s velkým důrazem kladeným na možnost číst tento formát lidmi. Formát YAML jako jediný zde uvedený pracuje se syntaxí založenou na odsazení (na rozdíl například od jazyka Python nepoužívá tabulátor, ale pouze mezery). Vnořený blok musí být od jeho rodiče oddělen právě dvěma mezerami. Od uvedení specifikace 1.2 je YAML technicky nadmnožinou formátu JSON, takže jakýkoliv dokument zapsaný ve formátu JSON je i platným YAML dokumentem, což velice usnadňuje přechod z formátu JSON. [15]

```
user:
  name: Antonín
  lastname: Neumann
  email:
    - neumann@students.zcu.cz
    - neumann@example.cz
  password: SECRET
  address:
    street: Ulice
    number: 123
    town: Plzeň
```

Zdrojový kód 7: Ukázka třídy User ve formátu YAML

3 Analýza technologií pro implementaci webových služeb

V předchozí kapitole byly definovány a popsány principy a možnosti pro realizaci webových služeb. V této kapitole je uvedena analýza a výběr konkrétních technologií.

Výběr vhodné technologie pro implementaci API je důležitý zejména z pohledu znalosti daného programovacího jazyka. Dále z času potřebného na impelementaci a dostupnosti nástrojů pro její usnadnění (frameworky a knihovny zjednodušující práci s přenosovými datovými formáty). Rychlost a robustnost následné implementace, stejně jako jednoduchost a náklady na provoz, jsou dalšími významnými faktory.

Ve své práci jsem se rozhodl použít pro implementaci API architekturu REST, dále se věnuji pouze této technologii pro tvorbu servisně orientovaných služeb.

Každý analyzovaný programovací jazyk je sice samostatně zcela funkční, ale pro provoz webových aplikací potřebuje webový server. Webserver je program, který se stará o komunikaci skrze protokol HTTP(S). Přijímá požadavky od klientů, předá je k vyřízení a odpověď odešle zpět.

Webový server	Procentuální zastoupení
Apache	46%
nginx	25%
ostatní	16%
Microsoft IIS	11%
Google	2%

Tabulka 1: Tržní podíl webových serverů milionu nejnavštěvovanějších stránek z března 2016, zdroj: Netcraft (http://news.netcraft.com/archives/2016/03/18/march-2016-web-server-survey.html)

3.1 PHP a Apache

PHP je skriptovací programovací jazyk, který vytvořil Rasmus Lerdorf jako, Perl/CGI skript pro počítání návštěvnosti jeho vlastních stránek. Pojmenoval jej jako *„Personal Home Page Tools“*. Později byl rozšířen o nové možnosti, přepsán do jazyku C a publikován jako PHP 2.0, též Personal Home Page/Form Interpret. Nakonec od verze PHP 3 je používám jako název pouze PHP s významem rekurzivního akronymu PHP: Hypertext Preprocessor. V současnosti je poslední stabilní verzí jazyka PHP 7.0.6. [16]

Jazyk PHP má v sobě přímo zahrnutý jednoduchý webový server, jenž je určen převážně pro testování. Pokud chceme webové aplikace postavené na PHP provozovat v tzv. produkčním prostředí, potřebujeme k tomu webový server, který je k tomuto účelu navržený.

Mezi nejznámější webservery pro tyto účely, lze zcela jistě zařadit Apache HTTP Server Project vyvíjený společnosti The Apache Software Foundation, dále nginx vytvořený a spravovaný společností NGINX Inc. a IIS (Internet Information Services) nabízený společností Microsoft.

Apache HTTP Server Project je dle posledního měření od společnosti Netcraft nejpoužívanější webový server na světě. Jedná se o svobodný a otevřený software, který je možné dále šířit nebo upravovat. Je distribuovaný pod licencí Apache License 2.0.

Z těchto důvodů jsem se rozhodl jej použít jako webový server pro programovací jazyk PHP.

3.2 Javascript a Node.js

JavaScript (zkráceně JS) je v poslední době velmi využívaným jazykem pro tvorbu webových aplikací. Jedná se o objektově orientovaný skriptovací jazyk, který v roce 1995, během deseti dnů, vytvořil Brendan Eich. Tento jazyk je

interpretovaný a používá se zejména na straně klienta, tj. převážně přímo ve webovém prohlížeči.

Existuje ovšem několik možností jak provozovat JavaScript na straně serveru. Mezi nejznámější patří node.js, Rhino nebo Nashorn. Rhino, spravovaný organizaci Mozilla Foundation, a Nashorn, od společnosti Oracle, jsou javascriptové engine napsané v programovacím jazyce Java. Oba slouží spíše pro provozování JS v kombinaci s Javou než pro provoz samostatných JS aplikací.

Naproti tomu projekt node.js umožňuje interpretovat javascript na straně serveru naprosto samostatně. Node.js používá interpret V8 JavaScript engine od společnosti Google a několik standardních knihoven, jejichž součástí je i server a modul pro práci se souborovým systémem.

Přestože node.js obsahuje modul, jak pro webový server, tak pro práci se soubory, je lepší využít k tomuto účelu framework, který za nás řeší některé základní požadavky a přidává vyšší míru abstrakce.

Často využívaný framework pro tvorbu webserveru na platformě node.js je express. Počet jeho stažení prostřednictvím správce balíčků (angl. package manager) za měsíc duben přesáhl 6 miliónů [17]. Express poskytuje snadný způsob tzv. *routingu*, tedy směrování ve smyslu definování přípojných bodů *(angl. end points)* aplikace v podobě URI, logiky zpracování požadavku a odeslání odpovědi. Rovněž podporuje zasílání odpovědí přímo ve formátu JSON. Pro express není problém odpovídat zprávami ve formátu XML nebo i dalšími.

3.3 Java a GlassFish

Java je objektově orientovaný jazyk vyvinutý v roce 1995 firmou Sun Microsystems (v současné době vlastněný společností Oracle). Java se

kompiluje do tzv. bytekódu *(angl. bytecode)* a následně se tento zkompilovaný kód interpretuje pomocí JVM (Java Virtual Machine). Díky použití JVM a bytekódu jsou programy napsané v Javě velmi snadno přenositelné mezi různými platformami. Nevýhodou je potřeba výrazně více paměti oproti jiným programovacím jazykům.

Java poskytuje pro tvorbu webových aplikací několik API, která jsou souhrnně označována jako JavaEE (Enterprise Edition). Pro běh JavaEE je zapotřebí aplikační server, který implementuje daná API. Těchto aplikačních serverů existuje velké množství. Od těch menších, například Jetty nebo Apache Tomcat, které implementují jen vybranou podmnožinu z definovaného API JavaEE, až po ty komplexní které implementují API celé. Ty poskytují většinou i nějakou funkcionalitu navíc v podobě například administrátorské konzole, logování atp. Mezi tyto větší aplikační servery může zařadit WildFly (dříve JBoss), nebo GlassFish, který je rovněž referenční implementací aplikačního serveru.

Pro webové služby postavené na REST architektuře poskytuje JavaEE rozhraní JAX-RS (Java API for RESTful Web Services**)**. Opět existuje několik různých implementací tohoto rozhraní, například Apache CXF nebo Jersey, který je referenční implementací JAX-RS rozhraní a najdeme ho přímo zahrnutý v aplikačním serveru GlassFish.

3.4 Ostatní technologie

Programovacích jazyků a platforem, ve kterých lze vytvořit a provozovat REST API, existuje velká spousta. Prakticky každý běžně používaný programovací jazyk má v dnešní době podporu pro tvorbu webových služeb, od těch běžných až po ty nové nebo méně časté:

- C# a .NET framework s použitím ASP.NET webového frameworku.
- Ruby a framework Ruby on Rails.

- Python a framework django nebo Flask.
- C++ s knihovnou C++ REST SDK, Restbed nebo DumaisLib.
- Go spolu s frameworkem Revel, Gorilla web toolkit nebo Goji.
- Dart, který sám obsahuje podporu pro tvoru REST API.
- A mnoho dalších.

3.5 Testovací API

Pro testování implementace v jednotlivých programovacích jazycích jsem vytvořil velmi jednoduchý návrh REST API se zprávami ve formátu JSON, který tvoří pouze 3 základní objekty *Uživatel, Peníze* a *Test*. Všechny objekty podporují základní operace CRUD. Níže je uvedena ukázková podoba objektů *User* a *Money*.

```
{
  "userId": 1,
  "name": "Antonín",
  "lastname": "Neumann",
  "birthday": "1989-09-07T00:00:00.000Z",
  "gender": "female",
  "money": [ __money objects__ ]
}
```

Zdrojový kód 8: Objekt User ve formátu JSON

Hodnotou proměnné *money* je pole objektů s penězi relevantních danému uživateli.

```
{
  "moneyId": 1,
  "amount": 1000,
  "description": "Some description",
  "spending": true,
  "date": "2015-12-31T15:30:00.000Z",
  "userId": 1
}
```

Zdrojový kód 9: Objekt Money ve formátu JSON

Objekt *Test* slouží pouze pro základní ověření implementace všech základních HTTP metod používaných pro komunikaci s API v architektuře REST (konkrétně GET, POST, PUT a DELETE), a jako odpověď vždy posílá hlavičku, která označuje úspěšný HTTP request.

Pro změření času odpovědí jednotlivých API serverů byla vytvořena jednoduchá aplikace v Javascriptu, která serveru pomocí technologie AJAX (Asynchronous JavaScript and XML) posílá jednotlivé requesty a měří čas pro jejich zpracování a vyřízení.

Vzhledem k náročnosti takové analýzy byly vybrány pro testování pouze 2 programovací jazyky: Javascript a PHP. Důvodem k výběru těchto dvou jazyků byla hlavně jejich znalost pro tvorbu webových aplikací. V případě PHP též velká podpora ze strany poskytovatelů webových serverů, tzv. hostingů. V případě správy vlastního serveru v budoucí fázi provozu jistě i jednodušší konfigurace webových serverů a menší paměťové nároky.

3.5.1 Testovací prostředí

Server, na kterém běžely vzorové implementace API, je notebook Lenovo ThinkPad E330 s těmito parametry:

- procesor Core i53210M 2.5GHz,
- operační paměť 8GB RAM,
- operační systém Windows 10 64bit,
- síťová karta RealTek Semiconductor RTL8168/8111 PCIE Gigabit Ethernet.

Softwarové prostředí pro běh PHP serveru bude představovat:

- Apache ve verzi 2.4.7 (32bit).
- PHP verze 5.5.8.

- Framework Slim [18].

Pro Javascriptový server bude použitý:

- Nodejs 4.2.3 LTS + NPM 2.14.0.
- Coffeescript 1.9.3.
- Framework ExpressJS [19].

3.5.2 Způsoby měření

Aby bylo možné vybrat technologii pro implementaci konečného API systému PartyBoard, definoval jsem tato měřící kritéria:

- rychlost odpovědí na základě stoupajícího počtu požadavků (simulace připojení více uživatelů),
- čas potřebný k implementaci v jednotlivých programovacích jazycích (speciální neopakovatelná veličina, která je velmi subjektivní).

Vzhledem k tomu, že AJAXové requesty jsou ze své podstaty prováděny asynchronně, bylo nutné zvolit, jakým způsobem budeme výsledný čas měřit.

Pokud budeme měřit čas od odeslání požadavku do jeho přijetí, získáme tím docela přesný výsledek, který ale příliš neodpovídá realitě, jelikož za dobu návratu jednoho requestu se reálně stihne provést a vrátit requestů několik.

Další možností je měřit celkovou dobu, kterou klient čekal na vyřízení všech jím poslaných requestů.

Čas pro implementaci jsem rozdělil celkově na tři samostatně měřené části. První část je základní seznámení s tvorbou REST API v těchto jazycích (ne více než pro potřeby testovací implementace), dále jsou problémy ohledně CORS, a čas samotné implementace testovacího serveru.

3.5.3 Naměřené výsledky

Všechny requesty během měření byly odesílány ihned a najednou, je tedy možné o tom uvažovat jako o připojení mnoha klientů současně.

V případě PHP začne prohlížeč přibližně od 5000 requestů (u Javascriptu jde přibližně o hodnotu 40000 requestů) vracet chybu *ERR_INSUFFICIENT_RESOURCES. Ta* poukazuje na překročení zdrojů webového prohlížeče, který již nezvládá alokovat paměť pro tolik dat. Aby se daly jednotlivé implementace vzájemně porovnávat, jsou níže uvedené výsledky maximálně pro 4000 požadavků.

Počet requestů	Javascript	PHP
800	250 309	1 825 019
1 200	411 876	5 978 861
1 600	564 524	7 462 934
2 000	771 165	3 788 518
2 400	913 447	4 847 682
2 800	1 056 296	2 231 911
3 200	1 536 401	3 786 511
3 600	1 680 391	3 952 018
4 000	1 824 721	5 015 028

Tabulka 2: Součet času jednotlivých requestů v milisekundách

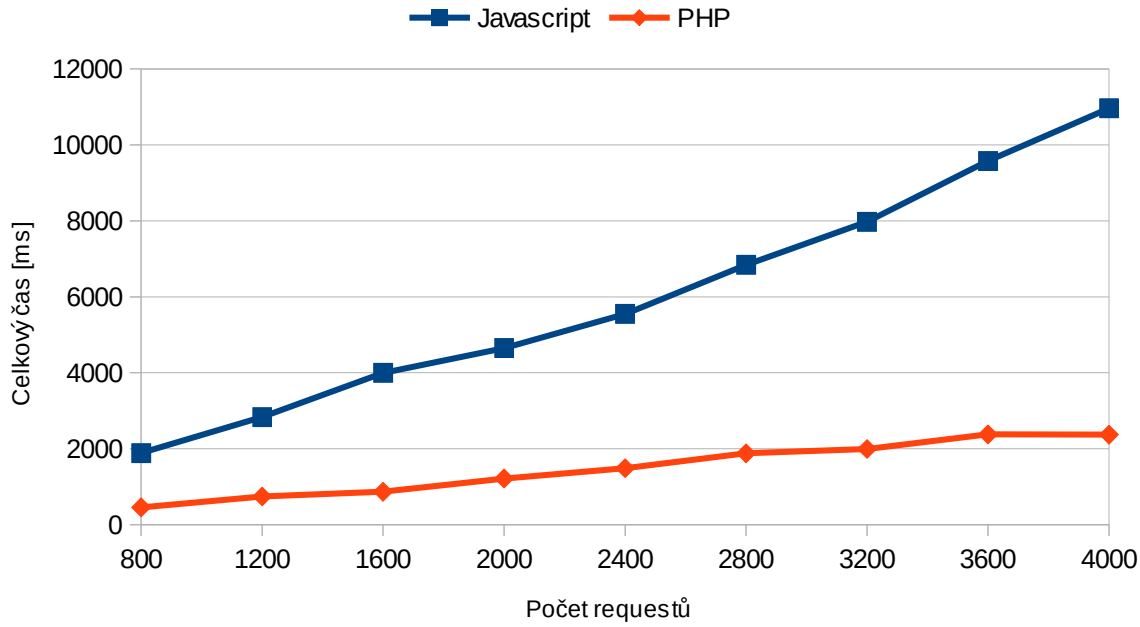
Počet requestů	Javascript	PHP
800	1 885	456
1 200	2 832	744
1 600	3 997	866
2 000	4 649	1 217
2 400	5 548	1 487
2 800	6 841	1 878
3 200	7 975	1 994
3 600	9 578	2 378
4 000	10 963	2 371

Tabulka 3: Celkový čas pro vyřízení všech requetů v milisekundách

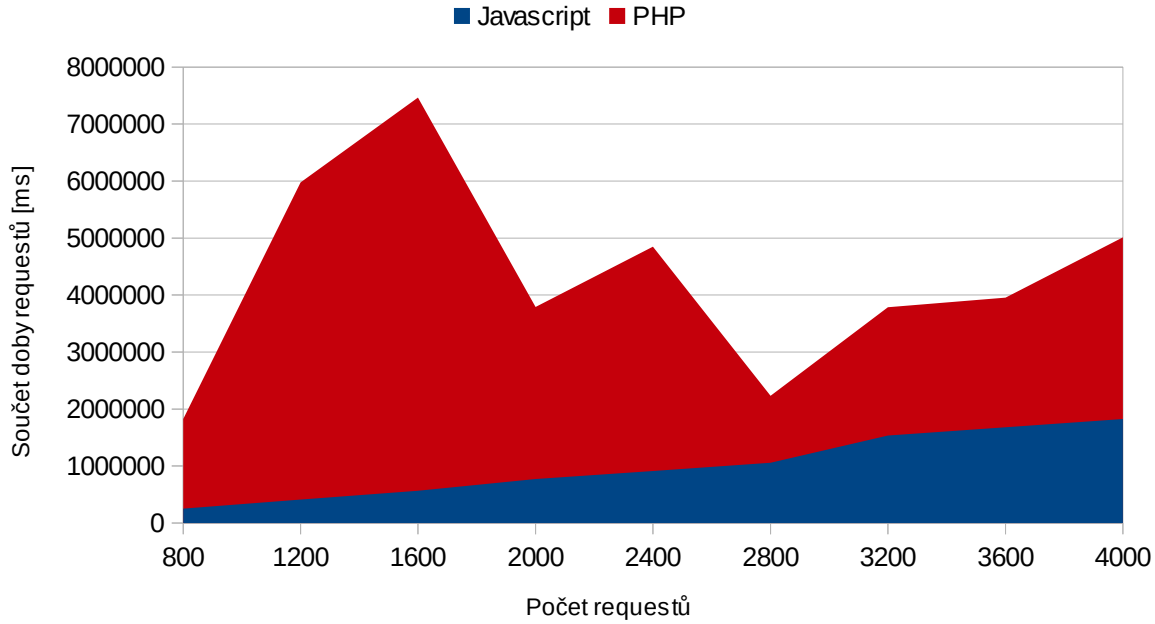
Typ úlohy	Javascript	PHP
Úvodní seznámení s nástroji	42 minut	1 hodina 16 minut
Řešení problémů s CORS	5 hodin 46 minut	2 hodiny 52 minut
Vlastní implementace API	4 hodiny 26 minut	2 hodiny 28 minut

Tabulka 4: Čas implementace API serverů (měřeno pomocí nástroje Toggle[3])

3 https://toggl.com



Graf 2: Celkový čas pro vyřízení všech requestů v milisekundách



Graf 3: Součet jednotlivých requestů

3.5.4 Závěr

Z tabulky č. 3 vyplývá, že odezva PHP serveru pro zpracování všech requestů je výrazně nižší, než u Javascriptu. To je dáno tím, že Apache dokáže zpracovávat requesty paralelně, na rozdíl od Node.js, které pracuje pouze s jedním vláknem. Tento nedostatek by šel vyvážit provozem několika instancí téhož node.js serveru a nasazení tzv. load balanceru. Server by jednotlivé requesty přijímal a podle svých interních statistik rozděloval rovnoměrně mezi tyto běžící instance.

Podle tabulky č. 2 byl součet časů potřebný na vyřízení všech requestů u Javascriptového serveru výrazně nižší, než u serveru implementovaného v PHP. To znamená, že javascriptová implementace má průměrný čas pro vyřízení jednoho requestu výrazně kratší, než je tomu u PHP serveru, a to 3 až skoro 8 násobně.

Celkový čas potřebný pro vytvoření serveru byl u Javascriptu přibližně dvojnásobný. To lze vysvětlit hlavně tím, že implementace serveru v Javascriptu proběhla jako první a některé obecné problémy nebo postupy implementace REST serveru, které bylo potřeba nastudovat a vyřešit, se promítly do času pouze u této implementace.

Též paměťové nároky systému postaveného na node.js jsou výrazně nižší. A nakonec i velmi příjemná práce s coffeescriptem mě dovedla k rozhodnutí zvolit pro výslednou implementaci REST API serveru pro systém Partyboard s kombinací platformy node.js, programovacího jazyka javascript/coffeescript a frameworku ExpressJS.

4 Analýza existujících MVC frameworků

V této kapitole analizuji a vybírám vhodný framework pro potřeby systému Partyboard. Tento framework bude sloužit pro tvorbu webových stánek určených k prezentaci celého systému, a též jeho správě. V závislosti na výběru vhodného nástroje je realizována základní podoba stránek, která bude dále rozšiřována již nad rámec této práce.

MVC (Model-View-Controller) je softwarový návrhový vzor, který od sebe odděluje jednotlivé logické celky aplikace na datový model, uživatelské rozhraní a řídící logiku do tří samostatných celků.

V modelu je umístěna celá *business* logika, tedy způsob ukládání dat nebo používání *služeb*. Nejčastěji se tato vrstva stará o ukládání dat v databázovém serveru. V případě systému PartyBoard je v modelu obsažena logika, která se stará o komunikaci s API serverem.

Uživatelské rozhraní, představuje část aplikace, která se stará o zobrazení dat uživateli. V případě webových stránek se jedná zejména o HTML, CSS a Javascriptové soubory, obrázky a specifická data poskytnutá modelem.

Kontroler *(angl. controller)* odděluje model od uživatelského rozhraní a řídí způsob, jakým aplikace reaguje na uživatelské vstupy. Jedná se o vstupní bod návrhového vzoru. Každý požadavek, který uživatel prostřednictvím webového prohlížeče a HTTP protokolu odešle, jde nejdříve právě sem [20].

4.1 Srovnání PHP frameworků

Na základě přečtení několika názorů na internetových fórech (např. StackOverflow), jsem dospěl celkem k 4 kandidátům, Nette, Symfony, Zend a vlastní řešení. Nette používá např. zpravodajský server Root.cz a mnozí

další [4]. Symfony framework používají např. projekty Drupal a phpBB[5]. U frameworku Zend se mi nepodařilo nalézt webové stránky nebo projekty, které jej využívají, ale samotný Zend podporují např. firmy jako je Google a Microsoft.

4.1.1 Nette

Je původem český framework vyvíjený od roku 2008 Davide Grudlem [21]. Projekt je šířený jako open source pod několika licencemi: New BSD (Berkeley Software Distribution) nebo též 3 bodová BSD licence [22] a GNU (GNU's Not Unix!) General Public License (GPL) verze 2 i 3 [23], ze kterých je možné si vybrat tu, která nejlépe vyhovuje potřebám daného projektu.

Aktuální stabilní verzi Nette 2.3.10 je možné stáhnout prostřednictvím PHP správce závislostí *(angl. Dependency Manager)* Composeru. Celý framework je komponentově orientovaný. Každou komponentu je tedy možné stáhnout, provozovat samostatně nebo použít všechny jako kompletní framework.

```
composer require nette/nette
```

Zdrojový kód 10: Příkaz pro stažení frameworku Nette pomocí nástroje Composer

Pro komunikaci s databází používá Nette komponentu *Database*, jejíž základy jsou postavené nad standardní PHP knihovnou pro práci s databází PDO (PHP Data Objects).

Pro vytváření uživatelského rozhraní používá Nette vlastní šablonovací systém Latte. Ten ošetřuje vstupy před případným XSS útokem a umožňuje použití různých maker a filtrů, které výrazně zjednoduší tvorbu prezentační vrstvy aplikace. Je ovšem nutné naučit se syntaxi šablonovacího systému a porozumět jeho základním principům.

4 https://builtwith.nette.org/

5 http://symfony.com/projects

Nastavení celé aplikace se provádí v konfiguračních souborech ve formátu Neon. Tento formát, navržený přímo pro Nette, je hodně podobný formátu Yaml popsaném v kapitole 2.5.3.

4.1.2 Symfony

Framework Symfony [24] je vyvíjen od roku 2005, jeho současná verze je 3.0. Distribuovaný je pod svobodnou licencí MIT license a jeho původním autorem je Fabien Potencier. Jedná se, podobně jako u Nette, o komponentově orientovaný framework, jehož jednotlivé komponenty je možné používat samostatně bez závislosti na ostatních.

Pro instalaci celého framework včetně základní struktury je nutné nainstalovat nástroj nazývaný Symfony Installer skrze, který je následně možné framework stáhnout. Alternativní možností je opět použít nástroj Composer.

```
symfony new my_project
composer create-project^
      symfony/framework-standard-edition my_project_name
```

Zdrojový kód 11: Příkazy pro stažení frameworku Symfony pomocí Symfony Intalleru a pomocí Composeru

Pro práci s databází nenabízí Symfony žádnou vlastní komponentu, místo toho doporučuje použití Doctrine ORM (Object-relational mapping), což je sada knihoven pro objektově-relační mapování a abstrakční databázové vrstvy.

Pro vytváření prezentační vrstvy používá tento framework vlastní šablonovací systém Twig. Ten stejně jako Latte (a většina dalších systému pro tvorbu šablon) automaticky ošetřuje vstupy a poskytuje nezbytná makra a užitečné filtry.

Symfony disponuje programem nazývaným *Console*, pomocí něhož lze aplikace vytvářet, nastavovat a spravovat. Veškerá nastavení jsou ukládána ve formátu YAML, takže i pokud jsou vytvořena pomocí *Console,* jde se v nich snadno vyznat a případně je ručně upravit.

4.1.3 Zend framework 2

Zend framework [25] je plně objektově orientovaný framework vyvíjený od roku 2005 a jeho současná verze 2 byla vydána v roce 2010 pod licencí New BSD license.

Zend je možné stáhnout pomocí Composeru, a také pomocí nástroje Pyrus. Pyrus správce PHP balíčků (komponent, knihoven) postavený nad repositářem PEAR (PHP Extension and Application Repository).

```
pyrus.phar . install zf2/Zend_Framework#Standard
composer require zendframework/zendframework
```

Zdrojový kód 12: Stažení Zend frameworku pomocí nástroje Pyrus nebo Composer

Pro přístup k databázi tento framework používá komponentu Zend_Db, která je podobně jako u Nette nadstavbou nad standardní PDO knihovnou jazyka PHP.

Zend nepoužívá žádný šablonovací systém, vystačí si pouze se základními možnostmi jazyka PHP a sadou funkcí, které jsou při vytváření UI (User Interface) často používané. Samozřejmě není problém používat Zend s jakýmkoliv jiným šablonovacím systémem.

Pro nastavení webové aplikace používá Zend komponentu nazvanou Zend_Config. Od této třídy je odvozeno několik dalších, které umožňují mít konfiguraci uloženou ve formátech INI, XML, JSON nebo YAML.

4.1.4 Vlastní PHP framework

Další možností je vytvoření svého, na míru přizpůsobeného, frameworku. Má zejména tyto výhody:

- Systému detailně rozumíme.
- Jakékoliv nestandardní úpravy nebo rozšíření funkčnosti se provádějí snáze.
- Můžeme implementovat věci po svém a případně lépe, alespoň v kontextu dané aplikace.
- Nemusíme se učit žádnou novou syntaxi a technické postupy.

Stejný přístup má ovšem i řadu nevýhod:

- Musíme se sami postarat o ošetření uživatelských vstupů proti případnému zneužití.
- Musíme pochopit a správně použít návrhový vzor MVC, který bude sloužit jako základ celého frameworku.
- Musíme vytvořit vlastní systém pro vytváření prezenční vrstvy.

4.2 Technologie pro webové stránky

U webové prezentace odpadá jakákoliv přímá práce s databází, jelikož veškerá komunikace s tímto úložištěm dat probíhá skrze API server. Webová aplikace vzhledem k unikátnosti celého projektu bude potřebovat několik nestandardních funkcí.

Po analyzování možností jednotlivých frameworů a po diskuzi o jejich možnostech s kolegou Bc. Novákem, byla vybrána jako vlastní implementace

MVC frameworku. Názor mého kolegy je v tomto případě relevantní, neboť i on bude v dalších fázích pracovat na prezentaci celého projektu.

Výhodou toho řešení, jak již bylo zmíněno dříve, je hlavně absolutní kontrola nad celým frameworkem a snazší možnost přidávat nestandardní změny specifické pro systém Partyboard.

Jistou nevýhodou je nutnost zaškolit případné spolupracovníky či zaměstnance. S tímto aspektem prozatím není počítáno, a to ani z pohledu pokračujícího rozvoje celého systému.

5 Návrh a implementace systému

V dalším textu popisuji návrh a implementaci jednotlivých klíčových částí systému Partyboard, které jsou: REST API pro komunikaci mezi jednotlivými aplikacemi, MVC frawework pro tvorbu prezentační části webu a webovou aplikaci *„Board“*, která slouží pro vizualizaci zpráv.

5.1 Návrh rozhraní API

Návrh API je realizován podle architektury REST. Základní jednotkou je zdroj nebo také entita. Každá entita může být obsažena v kolekci. Žádný zdroj nemůže být pomocí kolekce vytvořen, upraven ani smazán. Jedinou možností je získání celé kolekce, a proto je tomuto stavu návrh API přizpůsoben tak, že popis i implementace zdroje obsahuje všechny základní HTTP metody pro CRUD operace a navíc metodu GET pro získání kolekce téže entity.

Popis celého API se snaží reflektovat model databáze, který je uveden v příloze A, a který vypracoval Bc. Jan Novák jako součást jeho diplomové práce [26].

Pro popis návrhu REST API jsem se rozhodl použít službu Apiary.io [27], která umožňuje popisovat API, v syntaxi nazývané API Blueprint [28]. Celá definice API je uvedena na CD v adresáři přílohy. Ta je dostupná jako open source pod licencí MIT [29]. Syntaxe je velmi podobná jazyku Markdown publikovaném Johnem Gruberem v roce 2004 [30], což byl hlavní důvod výběru tohoto nástroje pro vytvoření popisu API. Dalším důvodem je, že nástroj Apiary.io umožňuje ze zadané syntaxe vytvořit přehlednou online dokumentaci, která je dostupná všem zájemcům o toto konkrétní API. Zároveň umožňuje vytvořit testovací server *(angl. mock server)*, nabízí ukázku volání požadavku každého zdroje v několika programovacích jazycích a umožňuje přímo z prostředí dokumentace volat požadavky na vlastní implementaci serveru.

```
FORMAT: 1A
HOST: http://api.example.com/

# Example API
This is the example of API Blueprint syntax

## Example Collection [/examples]

### List All Examples [GET /examples{?searchByName}]

+ Parameters
    + searchByName (string, optional)

+ Response 200 (application/json)
    {
      "name": "example"
      "value": 1
    }
```

Zdrojový kód 13: Ukázka systaxe API Blueprint

5.1.1 Základní pravidla pro návrh API

Při návrhu API jsem stanovil několik základních pravidel či omezení, která by měla sjednotit celý návrh a rovněž přispět k jeho přehlednosti. Do budoucna usnadní aplikaci jakýchkoliv změn, které bude zapotřebí provést v návrhu API nebo samotné databázi.

- Všechny názvy entit a jejich URI jsou v anglickém jazyce a množném čísle.
- Názvy proměnných (klíčů v kontextu JSON objektu) jsou shodné s názvy příslušných sloupců v databázi a víceslovné názvy jsou oddělené znakem „_“ (podtržítko, v utf-8 kód 0x5F).
- Pokud tabulka v databázi obsahuje spojení typu 1:N (například jeden klub může mít přiřazeno více „Boardů“) potom je entita, která v sobě obsahuje v databázi cizí klíč, vrácena místo cizího klíče přímo se zanořeným objektem. Výjimkou je entita příchozích zpráv *(angl. incomming messages),* která se načítá nejčastěji (v závislosti

na konkrétním nastavení například každých 500ms), je proto důležité, aby objem přenášených informací byl co nejmenší.

- Pokud položka může obsahovat více zanořených objektů, jsou tyto vždy vráceny jako pole, a to i v případě, že je v konkrétním případě vrácena pouze jedna entita. V případě že entita neobsahuje žádnou konkrétní zanořenou položku, a je-li to přípustné, je vráceno prázdné pole. Tímto vymezením se výrazně usnadní parsování příchozích objektů v klientech využívajících toto API.

5.1.2 Metody pro práci se zdroji

V API jsou použité základní HTTP metody GET, POST, PUT a DELETE odpovídající operacím čtení, vytvoření, úprava a smazání.

Metoda GET vrací při uvedení identifikátoru zdroje objekt odpovídající tomuto identifikátoru, v případě dotazu bez uvedení tohoto identifikátoru je vrácena celá kolekce (velikost kolekce je defaultně omezená na 30).

Metoda POST umožňuje zdroj pouze vytvářet a metoda PUT jen upravovat. Obě metody očekávají v těle požadavku *(angl. request body)* objekt podle vzoru uvedeného u každého zdroje v dokumentaci. Při úpravě je navíc očekávám identifikátor zdroje jako parametr v URI. Předpokladem pro úspěch takové požadavku, že zdroj existuje a je stále platný.

Metoda DELETE smaže zdroj specifikovaný jeho identifikátorem, pokaždé ale nedojde k přímému smazání zdroje v databázi, ale pouze k jeho zneplatnění, tj. nastavení sloupce s názvem *delete* na hodnotu TRUE. Tato funkcionalita je u některých entit zařazena hlavně z důvodu možného vrácení dané operace *(angl. undo)*. A rovněž kvůli možnosti zpětně dohledat některé informace, při řešení případných sporů (například zprávy smazané kvůli nevhodnému obsahu).

5.1.3 Filtrování kolekcí

Pro práci s kolekcemi zdrojů umožňuje API jejich filtrování. Základní filtry, které je možné použít s každou kolekcí, jsou:

- *Limit* – určuje kolik entit je v odpovědi vráceno.
- *Offset* – definuje pozici v databázi prvního zdroje kolekce (v kombinaci s limitem lze takto získat všechny zdroje).
- *Order* – říká serveru, v jakém pořadí má zdroje v kolekci uspořádat. Možné hodnoty jsou *asc* pro vzestupné a *desc* pro sestupné řazení.
- *Delete* – vrátí v kolekci i zdroje, které jsou označené jako neplatné, tento filtr je dostupný pouze u zdrojů, u kterých nedochází při požadavku na smazání k fyzickému odstranění v databázi.

Další filtry jsou již specifické jen pro dané kolekce a u jiných by neměli smysl, neboť jsou nejčastěji svázány s cizím klíčem odkazujícím na jiný propojený zdroj:

- *User* – dovoluje filtrovat entity podle ID uživatele. Dostupný je u kolekci *payments info,* i*ncomming messages* a *outcomming messages*.
- Phone – omezuje výsledky u kolekce *incomming* a *outcomming messages* podle telefonního čísla.
- *Group_setting* – filtruje kolekci *settings* podle ID skupiny nastavení.
- *Partyboard* – vrací zdroje kolekce *group settings* a co*mpetition partyboards* omezené podle ID partyboardu, ke kterému se vážou.
- Ongoing – filtruje dosud trvající bany u kolekce *bar user partyboard.*

5.1.4 Hodnoty, stavové kódy a hlavičky odpovědí

Odpověď serveru odesílanou prostřednictvím HTTP protokolu je pro naše potřeby možné rozdělit do tří částí: hlavičky *(angl. headers)*, stavový kód *(angl. status code)* a tělo odpovědi *(angl. body)*.

Stavové kódy jsou rozdělené podle jejich významu celkem do pěti skupin, které popisuje tabulka č. 5. V tomto API jsou použité prozatím pouze kódy 2xx při úspěšném požadavku, 4xx při nějaké chybě a 500 při neočekávané chybě serveru.

Řád stavového kódu	Význam kódů
1xx	Informační
2xx	Úspěšný request *(angl. success)*
3xx	Přesměrování
4xx	Chyba na straně klienta
5xx	Chyba na straně serveru

Tabulka 5: Typy stavových kódů protokolu HTTP

Požadavek na vytvoření zdroje poslaný metodou POST vrací kód 201, všechny ostatní metody při úspěšném requestu vrací kód 200.

Při neúspěšném requestu je vrácen kód ze skupiny 4xx a současně s ním je nastavena hlavička *X-Error,* v níž je uveden popis chyby. V případě že je chyba způsobena dotazem do databázového serveru je navíc vrácena hlavička *X-Error-Original,* která obsahuje text s chybou generovanou samotnou databází.

Kód chyby	Chybová hlavičky X-Error Další hlavičky	Význam
400	Bad request! X-Error-Original	Chyba v requestu.
401	X-Access-Token is required! WWW-Authenticate	Neautentifikovaný požadavek.
403	Token wasn't verified!	Neautorizovaný požadavek.
404	Resource not found!	Zdroj nebyl nalezen.
405	Method Not Allowed!	Požadovaná metoda není pro tento zdroj implementována.
409	Resource already exist! X-Error-Original	Zdroj již existuje.

Tabulka 6: Chybové zprávy a stavové kódu odpovědí API serveru

Tělo odpovědi obsahuje v případě úspěšného požadavku u metod GET, POST a PUT vždy celý objekt, případně kolekci objektů a u metody DELETE je tělo prázdné.

V případě chybného požadavku je tělo odpovědi prázdné u všech metod.

5.1.5 Zabezpečení API

API systému PartyBoard je určené výhradně jeho uživatelům, a proto je zabezpečeno proti veřejnému přístupu pomocí tzv. JWT (JSON Web Tokens). Tento token získá uživatel po přihlášení do aplikace. Je nutné, aby jej zasílal při každém svém požadavku v hlavičce *X-Access-Token*, v opačném případě dojde k vrácení chyby.

Zabezpečené jsou všechny URI a jejich metody. Výjimku tvoří metoda OPTIONS a dvě URI. První určené pro registraci nových uživatelů (*POST /users*) a druhé pro přihlášení uživatele a vygenerování tokenu *(POST /auth)*. Metoda OPTIONS je povolená z důvodu odesílání tzv. předčasného requestu *(angl. preflight request)*.

Ten se posílá z důvodu zabezpečení volání zdrojů mezi různými doménami a bez využití metody CORS (Cross-origin resource sharing) nedovoluje javascriptovému kódu z důvodu bezpečnosti odesílání requestů na jiné domény.

CORS bylo vydáno jako doporučení W3C v lednu 2014 [31] a definuje způsob, kterým lze povolit vzdálené volání zdrojů mezi různými doménami se zachováním bezpečnosti. Základem je přesný výčet povolených metod a hlaviček, které server pro požadavek na daný zdroj z jiné domény akceptuje. K tomu, aby mohl klient, který chce serveru poslat požadavek, zjistit jestli tak může učinit, slouží výše zmíněný *preflight request*.

5.1.6 Ukázka entity obsahující relaci 1:N

Entita pro skupinu nastavení *(angl. group settings)* v sobě obsahuje jednotlivá nastavení *(angl. settings)*, ta v sobě dále obsahuje typ tohoto nastavení *(angl. type setting)*. Ta ještě jako poslední entitu pro datový typ *(angl. data type)*. Zde tedy dochází k trojnásobnému zanoření a jedná se o zanoření entit v celém systému, které ovšem vychází z aplikování normálních forem při návrhu databáze.

```
{
    "id_group_setting": 1,
    "id_partyboard": 249,
    "name": "Valentine",
    "description": "Valentynsky nastaveni.",
    "active": true,
    "settings": [{
      "id_setting": 1,
      "id_group_setting": 1,
      "type_setting": {
        "id_type_setting": 1,
        "data_type": {
          "id_data_type": 3,
          "name": "color",
          "regex": "[A-Fa-f0-9]{6}",
          "create_ts": "2016-04-15 10:23:54+01",
          "update_ts": "2016-06-17 10:23:54+01"
        },
        "name": "text_color",
        "create_ts": "2016-04-15 10:23:54+01",
        "update_ts": "2016-06-17 10:23:54+01"
      },
      "value": "FFFFFF",
      "create_ts": "2016-04-15 10:23:54+01",
      "update_ts": "2016-06-17 10:23:54+01"
    },
     {
      "id_setting": 1,
      ...
    }],
    "delete": false,
    "create_ts": "2016-04-15 10:23:54+01",
    "update_ts": "2016-06-17 10:23:54+01"
}
```

Zdrojový kód 14: Ukázka objektu „Group settings“ vráceného jako tělo odpovědi

5.1.7 Popis entit

Entity v celém systému lze rozdělit do 4 základních skupin na:

- Významné – tvoří hlavní logiku celého systému.
    - Příchozí zprávy (incomming messages)

    - Uživatelé (users)
    - Partyboady (partyboards)
- Běžné – jsou entity, na kterých nezávisí jádro celého systému.
    - Kluby (clubs)
    - Reklama (ads)
    - Odchozí zprávy (outcomming messages)
    - Výherní poukazy (coupons)
    - Platební údaje (payments info)
- Číselníky – klasické a tzv. rozšířené, které v sobě zahrnují další číselník
    - Sprostá slova (dirty words)
    - Města (towns)
    - Kódy bank (banks_code)
    - Přístupové role pro partyboard (roles partyboard)
    - Oprávnění pro partyboard (permissions partyboard)
    - Přístupové role pro účet (roles partyboard)
    - Oprávnění pro účet (permissions partyboard)
    - Nastavení (settings)
    - Typ nastavení (type settings)
    - Datový typ nastavení (data types)
    - Nastavení zpráv pro soutěže (setting messages)
    - Typy soutěží (type competitions)

    - Typ příchozí zprávy (message keywords)
- Propojovací – vzájemně propojují dvě nebo více jiných entit.
    - Skupiny nastavení (group settings)
    - Role uživatelů k partyboardům (roles users partyboards)
    - Ban uživatele v partyboardu (ban user partyboard)

5.2 Implementace API

Pro implementaci API byl zvolený jazyk Coffeescript [32] provozovaný na platformě node.js. Coffeescript je tzv. transpilovaný jazyk. Při kompilaci dochází k převedení jeho syntaxe na syntaxi Javasciptu. Tento jazyk oproti klasickému Javascriptu vylepšuje práci se třídami, moduly i proměnnými. Jedná se o jazyk postavený na odsazování, a proto bylo možné zcela vypustit středníky uváděné na konci každého příkazu a většinu závorek.

```
GetQuery = (reqQueries, query) ->
  for key,val of reqQueries
    if key is query then return val
  return null
```

Zdrojový kód 15: Ukázka funkce v Coffeescriptu

Stejná funkce napsaná v normálním Javascriptu vypadá následovně:

```
var getQuery = function(reqQueries, query) {
  var key, val;
  for (key in reqQueries) {
    val = reqQueries[key];
    if (key === query) { return val; }
  }
  return null;
};
```

Zdrojový kód 16: Ukázka funkce v Javascriptu

Jako základní prostředek pro definici URI přípojných bodů, zpracování požadavků a vytváření odpovědí byl použit framework express. Tento

framework, stejně jako většina modulů v node.js, se do projektu přidává pomocí správce balíčků npm (node.js package manager).

Express umí zpracovávat všechny HTTP metody. Je vhodný i pro REST API, které je na těchto metodách postavené. Express pracuje na principu tzv. middleware. Jedná se o metody, které daný požadavek nebo odpověď modifikují a předají k dalšímu zpracování nebo odešlou zpět klientovi. Pomocí takového middleware bylo v API povoleno CORS volání požadavků pro všechny příchozí requesty.

```
masterRouter.use (req, res, next) ->
  res.header 'Access-Control-Allow-Origin', '*'
  res.header 'Access-Control-Expose-Headers',
    "X-Error, X-Error-Type, X-Access-Token, X-Error-Original"
  res.header 'Access-Control-Allow-Headers',
      "Origin, X-Requested-With, Content-Type,
       Accept, X-Access-Token"
  res.header 'Access-Control-Allow-Methods',
      "OPTIONS, GET, HEAD, POST, PUT, DELETE,TRACE, CONNECT"
  res.header 'Content-Type', "application/json"
  next()
```

Zdrojový kód 17: Ukázka middleware povolujícího použití CORS

Každý zdroj má definovanou základní cestu a soubor, ve kterém se nachází. Uuvnitř tohoto souboru jsou poté definovány jednotlivé metody, které lze pro daný zdroj volat.

Pro příjem SMS zpráv byla vytvořena speciální URI adresa, která dovoluje službě Mobilem.cz přímou komunikaci s API. Tato služba poskytuje celému systému svou GSM bránu, jenž se stará o příjem jednotlivých SMS zpráv. Každou přijatou SMS zprávu odešle pomocí HTTP požadavku metodou GET. Použití této služby je pouze pro testovací účely. V ostrém provozu se počítá s nasazením komerční GSM brány, která by umožnila odesílání SMS zpráv zpoplatněných tzv. vyšším tarifem. Nicméně zřízení a provoz těchto služeb je

zpoplatněn. V rámci testování dobře poslouží i alternativa poskytovaná Mobilem.cz zdarma.

5.3 Návrh a popis webového frameworku

Pro implementaci a návrh vlastního MVC frameworku byl použit jazyk PHP. Všechny části včetně vlastního zpracování požadavků jsou vytvořeny speciálně pro tento framework. Většina součástí vychází z vlastních požadavků a návrhového vzoru MVC. Některé myšlenky jsou ovšem inspirovány různými jinými frameworky, například Nette.

Vývoji webových stránek v jazyce PHP se věnuji již několik let, a proto je tento framework kondenzací mých dosavadních zkušeností a znalostí. Některé myšlenky, případně malé části zdrojových kódu, jsou založeny na předchozích projektech. Přestože je framework vytvářený na míru potřebám systému Partyboard, jsou pro většinu částí použity obecné postupy, takže výsledný framework je možné použít i pro jiné systémy. Adresářová struktura celého frameworku je uvedena v příloze C.

Konfigurace celého frameworku je soustředěna v adresáři *config/*, který obsahuje zejména soubory pro nastavení jednotlivých modulů, filtrů a konstant.

Celý návrh se snaží dodržovat principy OOP (Objektově orientované programovaní). Jsou v něm tedy vytvořeny základní třídy použité pro zpracování a předání requestu správnému modelu resp. kontroleru, abstraktní třídy obsahující základní funkcionalitu pro komunikaci kontroleru, modelu a šablony *(angl. view),* funkce určené pro vytváření a vykreslování jednotlivých šablon a jejich poslání do webového prohlížeče.

Modul je základní jednotkou frameworku, každý modul respektuje návrhový vzor MVC. A umožňuje vytvářet nezávislé celky, které mají od ostatních odlišnou funkcionalitu, například modul pro statické stránky a administraci.

Každý modul může mít vlastní URL prefix identifikující jej v rámci frameworku. Pokud se jedná o tzv. defaultní modul, je v případě nenalezení vyhovující adresy předán request ke zpracování jemu.

```
$cfg['module']['admin'] = array(
    'name' => 'Admin module',
    'default' => FALSE,
    'single_controller' => FALSE,
    'namespace' => 'Admin',
    'url_prefix' => array('admin')
);
```

Zdrojový kód 18: Ukázka konfigurace modulu v souboru /config/modules.php

Načítání jednotlivých modulů, stejně jako jeho tříd je prováděno pomocí tzv. autoloaderu. Jedná se o třídu statických funkcí, které je v PHP možné použít k automatickému načítání potřebných závislostí – není nutné nikde explicitně uvádět cestu k těmto třídám.

Zpracování každého požadavku je prováděno ve třídě *Morgo\Core\RequestParser*, která požadavek analyzuje a podle nastavení uvedeného v souborech *config/modules.php* a *config/contatns.php* jej předá k dalšímu zpracování konkrétnímu kontroleru. Specialitou je přidání tzv. filtrů, což jsou třídy umožňující na základě své funkcionality modifikovat požadavek i odpověď. Výborně se tyto filtry hodí v případě, že chceme zabezpečit přístup na některé stránky jen oprávněným uživatelům.

Framework používá k tvorbě šablon pouze čisté PHP, tím odpadá nutnost učit se další syntaxi nějakého šablonovacího systému, ale přidává nutnost postarat se o případné ošetření uživatelský vstupů. Celá výsledná webová stránka se postupně skládá z několika částí:

- Template – zde jdou uvedeny základní definice HTML dokumentu včetně části obsahující metadata.

- Layout – tento dokument obsahuje základní rozmístění prvků a definuje výsledný vzhled stránky.
- View – poslední část, která reprezentuje obsah stránky. Těchto souborů je většinou velké množství a jsou uloženy zvlášť u každého modulu.

Každý modul, dokonce i každý kontroler, může předefinovat výchozí *Template* nebo *Layout,* a je tedy možné mít pro 2 různé moduly zcela odlišné vzhledy.

Poslední užitečnou částí framework jsou, tzv. helpery. To je třída, která jednotlivým šablonám poskytuje užitečné funkce. Je možné pomocí nich dosáhnout k velice snadné modifikaci nebo výpisu dat, atd.

5.4 Návrh webová aplikace pro zobrazování zpráv

Aplikace pro zobrazování zpráv, je vlastně poměrně jednoduchá SPA (Single Page Application), napsaná v programovacím jazyce Javascript s využitím knihovny jQuery. Tato knihovna umožňuje komplexnější práci s DOM (Document Object Model) webové stránky a usnadňuje použití technologie AJAX pro posílání requestů na API server. Většina nastavení této stránky probíhá skrze mobilní aplikaci, a později bude umožněno nastavování také přes webové rozhraní administrační části stránek Partyboard.cz.

6 Testování a pilotní nasazení

API implementace serveru byla otestována pomocí programu Postman[6] nabízeném jako aplikace pro webový prohlížeč Google Chorme. Tento program umožňuje volání definovaných URI, a to i pomocí dávkového zpracování. Tímto způsobem byly otestovány přístupy ke všem zdrojům definovaným v rámci návrhu API. Konfigurační soubory s definicí kolekcí, jejichž prostřednictvím je možné funkčnost API vyzkoušet jsou přiložené na CD.

Webová aplikace byla otestována pouze tzv. uživatelským testováním, které ukázalo, že je schopná provozu dle svého účelu.

Kolega Bc. Novák ve své práci provedl zátěžové testování jím vytvořené implementace databáze a ověřil tak její funkčnost.

6.1 Popis testovacího prostředí pilotního nasazení

Pro implementaci, provoz a vyzkoušení komunikace jednotlivých aplikací systému Partyboard bylo Katedrou informatiky (KIV) při Západočeské univerzitě v Plzni (ZČU) poskytnuto hardwarové i softwarové řešení provozované samotným pracovištěm KIV.

Server, na kterém běží naše aplikace je složený z následujícího hardwarového vybavení: osm dvou jádrových procesorů Intel® Xeon™ 3.0 GHz s operační pamětí 32 GB. Dále 3-krát 80 GB disky v RAID 5 a 2-krát 1TB disky v RAID1.

Softwarové vybavení serveru, které využíváme, zahrnuje webový serveru Apache2, PHP 5, NodeJS a databázi PostresSQL ve verzi 9.2.

6 www.getpostman.com

7 Závěr

Cílem této práce bylo vytvořit dílčí aplikace pro systém Partyboard. Bylo navrženo a implementováno REST API, vytvořen vlastní MVC framework na míru potřebám systému a webová prezentace umožňující zobrazovat zprávy.

Funkce všech těchto částí byla ověřena pomocí uživatelské testování, a to i s aplikacemi vytvořenými Bc. Novákem.

Práce na celém projektu byla velmi zajímavá, jelikož se jedná o reálný projekt s komerčním potenciálem. Rovněž technologie REST a samotná tvorba API pro mě byla přínosná, protože v dnešní době je o podobné technologie velký zájem.

Možným rozšíření API systému je implementace metody PATCH. Ta umožňuje editovat jen vybrané části zdroje, čímž můžeme ušetřit objem přenášených dat.

Dalším postupem bude přesunutí celého systému z testovacího prostředí univerzity a spuštění pilotního nasazení v rámci prvního vybraného klubu. K provozu systému je plánováno zakoupení VPS (Virtual Private Server) a pořízení komerční GSM brány, která umožní odesílání SMS zpráv zpoplatněných tzv. vyšším tarifem (např. výherní SMS za cenu 30 Kč).

Během pilotního nasazení, které je odhadováno na 2 až 3 měsíce, dojde k naimplementování kompletní prezentační webové stránky a ostrému spuštění systému.

Seznam literatury

[1] C. Matthew MacKenzie, Ken Laskey, Francis McCabe, Peter F Brown, Rebekah Metz, [online]. Dostupné z <https://docs.oasis-open.org/soa-rm/v1.0/soa-rm.html#_Toc22282948> [cit.1. 3. 2016]

[2] The Open Group, [online]. Dostupné z <http://www.opengroup.org/soa/source-book/togaf/soadef.htm> [cit.1. 3. 2016]

[3] Nilo Mitra, Yves Lafon, Martin Gudgin a další, SOAP Version 1.2 [online]. Dostupné z <https://www.w3.org/TR/soap/> [cit.23. 4. 2016]

[4] Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman, Sanjiva Weerawarana, Web Services Description Language (WSDL) Version 2.0 [online]. Dostupné z <https://www.w3.org/TR/wsdl20/> [cit.17. 4. 2016]

[5] Martin KUBA, Web Services [online]. Dostupné z <http://dior.ics.muni.cz/~makub/soap/MartinKuba_WebServices_Datakon2006_clanek.pdf> [cit.7. 5. 2016]

[6] Roy Thomas Fielding, Architectural Styles andthe Design of Network-based Software Architectures, CHAPTER 5 Representational State Transfer (REST) [online]. Dostupné z <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm> [cit. 6. 4. 2016]

[7] T. Berners-Lee, L. Masinter, M. McCahill, Uniform Resource Locators (URL) [online]. Dostupné z <https://tools.ietf.org/html/rfc1738> [cit. 6. 4. 2016]

[8] K. Sollins, L. Masinter, Functional Requirements for Uniform Resource Names [online]. Dostupné z <https://tools.ietf.org/html/rfc1737> [cit. 6. 4. 2016]

[9] J.D. Meier, Alex Homer, David Hill, Jason Taylor, Prashant Bansode, Lonnie Wall, Rob Boucher Jr, Akshay Bogawat, Microsoft Application Architecture Guide, 2nd Edition [online]. Dostupné z <https://msdn.microsoft.com/en-us/library/ff650706.aspx> [cit. 6. 4. 2016]

[10] Roy Thomas Fielding, Architectural Styles andthe Design of Network-based Software Architectures, CHAPTER 3 Network-based Architectural Styles [online]. Dostupné z <http://www.ics.uci.edu/~fielding/pubs/dissertation/net_arch_styles.htm> [cit.30. 4. 2016]

[11] Fernando Doglio, Pro REST API Development with Node.js, Chapter 1: Rest 101 - REST Constraints, Apress. ISBN9781484209172

[12] T. Bray, [online]. Dostupné z <https://tools.ietf.org/html/rfc7159> [cit.2. 5. 2016]

[13] ECMA international, [online]. Dostupné z <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf> [cit.2. 5. 2016]

[14] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau, Extensible Markup Language [online]. Dostupné z <https://www.w3.org/TR/xml/> [cit.2. 5. 2016]

[15] Oren Ben-Kiki, Clark Evans, Ingy döt Net, YAML Ain't Markup Language [online]. Dostupné z <http://www.yaml.org/spec/1.2/spec.html> [cit.8. 5. 2016]

[16] The PHP Group, History of PHP [online]. Dostupné z <http://php.net/manual/en/history.php.php> [cit.8. 5. 2016]

[17] npm, express [online]. Dostupné z <https://www.npmjs.com/package/express> [cit.6. 5. 2016]

[18] Josh Lockhart, Andrew Smith, Rob Allen, Slim framework [online]. Dostupné z <http://www.slimframework.com/> [cit.7. 3. 2016]

[19] Node.js Foundation, Express [online]. Dostupné z <http://expressjs.com/> [cit.3. 4. 2016]

[20] Chris Pitt, Pro PHP MVC, Apress. ISBN9781430241652

[21] David Grudl, Nette [online]. Dostupné z <https://nette.org/> [cit.5. 5. 2016]

[22] Regents of the University of California, BSD license [online]. Dostupné z <https://opensource.org/licenses/BSD-3-Clause> [cit.1. 5. 2016]

[23] Richard Stallman, GNU GPL [online]. Dostupné z <http://www.gnu.org/licenses/gpl-3.0.html> [cit.1. 5. 2016]

[24] Fabien Potencier, Symfony, [online]. Dostupné z <http://symfony.com/> [cit.5. 5. 2016]

[25] Zend Technologies Ltd, Zend framework [online]. Dostupné z <http://framework.zend.com/> [cit.5. 5. 2016]

[26] Jan Novák, Databázový systém, mobilní aplikace a business plán pro aplikaci Partyboard, diplomová práce 2016

[27] Antonín Neumann, [online]. Dostupné z <http://docs.partyboard.apiary.io/> [cit.1. 4. 2016]

[28] apiary.io, Apiary.io [online]. Dostupné z <www.apiblueprint.org > [cit.15. 4. 2016]

[29] Massachusetts Institute of Technology, MIT license [online]. Dostupné z <https://opensource.org/licenses/MIT> [cit.1. 5. 2016]

[30] John Gruber, Markdown [online]. Dostupné z <https://daringfireball.net/projects/markdown/> [cit.13. 4. 2016]

[31] Anne van Kesteren, Cross-Origin Resource Sharing [online]. Dostupné z <https://www.w3.org/TR/cors/> [cit.4. 5. 2016]

[32] Jeremy Ashkena, Coffeescript [online]. Dostupné z <http://coffeescript.org/> [cit. 4. 5. 2016]

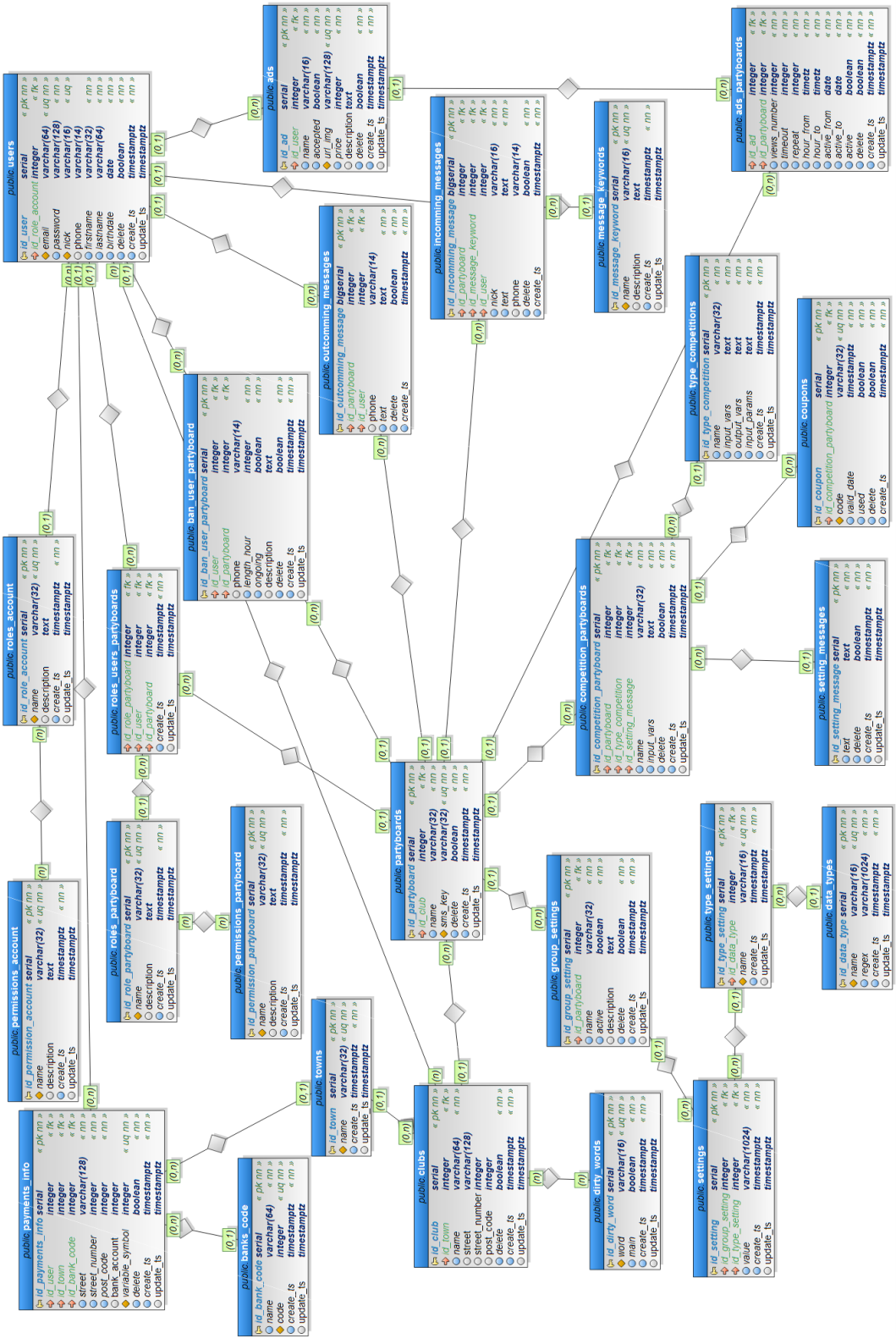
Seznam tabulek

Seznam grafů

Seznam zkratek

AJAX	Asynchronous JavaScript and XML
BSD	Berkeley Software Distribution
CORS	Cross-origin resource sharing
CRUD	Create-Read-Update-Delete
DCOM	Distributed Component Object Model
DOM	Document Object Model
DNS	Domain Name System
GNU	GNU's Not Unix!
GPL	General Public License
HTTP	Hypertext Transfer Protocol
JAX-RS	Java API for RESTful Web Services
JVM	Virtual Machine
JWT	JSON Web Tokens
JavaEE	Java Enterprise Edition
MOVE	Models-Operation-Views-Events
MVC	Model-View-Controller
OOP	Objektově orientované programovaní
ORM	Object-relational mapping
PDO	PHP Data Objects
PEAR	PHP Extension and Application Repository
REST	Representational State Transfer
SGML	Standard Generalized Markup Language
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
SPA	Single Page Application
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UI	User Interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
VPS	Virtual Private Server
WSDL	Web Services Description Language
WWW	World Wide Web
XML	Extensible Markup Language
XSD	XML Schema Definition

Příloha A: databázový model



Příloha B: ukázka WSDL souboru

```
<!--
   (c) Antonín Neumann
   Online WSDL 1.1 SOAP generator 0.2
   Julien Blitte
-->
<definitions xmlns:tns="com.example.dp.wsdl"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsd1="com.example.dp.xsd"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns="http://schemas.xmlsoap.org/wsdl/" name=""
  targetNamespace="com.example.dp.wsdl">

  <!--  definition of datatypes  -->
  <types>
    <schema xmlns="http://www.w3.org/2000/10/XMLSchema"
      targetNamespace="com.example.dp.xsd">
      <element name="User">
        <complexType>
          <all>
            <element name="value" type="string"/>
          </all>
        </complexType>
      </element>
      <element name="UserName">
        <complexType>
          <all>
            <element name="value" type="string"/>
          </all>
        </complexType>
      </element>
    </schema>
  </types>

  <!--  response messages  -->
  <message name="GetUserResponse">
    <part name="User" type="xsd:User"/>
  </message>
```

```
  <!--  request messages  -->
  <message name="GetUser">
    <part name="UserName" type="xsd:UserName"/>
  </message>

  <!--  server's services  -->
  <portType name="Users">
    <operation name="GetUser">
      <input message="tns:GetUser"/>
      <output message="tns:GetUserResponse"/>
    </operation>
  </portType>

  <!--  server encoding  -->
  <binding name="Users" type="tns:Users">
    <soap:binding style="rpc"
      transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="GetUser">
      <soap:operation

        soapAction="urn:xmethods-delayed-quotes#GetUser"/>
      <input>
        <soap:body use="encoded"
          namespace="urn:xmethods-delayed-quotes"
          encodingStyle="http://schemas.xmlsoap.org/
          soap/encoding/" />
      </input>
      <output>
        <soap:body use="encoded"
          namespace="urn:xmethods-delayed-quotes"
          encodingStyle="http://schemas.xmlsoap.org/
          soap/encoding/"/>
      </output>
    </operation>
  </binding>

  <!--  access to service provider  -->
  <service name="exemple">
    <port name="exemple_0" binding="Users">
      <soap:address location="http://example.com/users"/>
    </port>
  </service>
</definitions>
```

Příloha C: Adresářová struktura PHP frameworku

```
+ assets
     + css
     + img
     + js
     + less (soubory CSS preprocesor files)
     + template (šablony pro prezentační vrstvu)
       + template.tpl
       + layout.tpl
       + body_end_js.tpl
       + head_css_js.tpl
+ core
     + config
       + locks
         + _db.lock
       + config.php
       + constatnts.php
       + filters.php
       + modules.php
     + exception
     + lib
       + Locker.php
       + debug.php
       + dibi.phar
       + error_handler.php
       + filter_register.php
       + magic_quotes.php
       + password.php
       + string.php
       + tracy.phar
     + ABaseCotronller.php
     + ABaseFilter.php
     + ARenderController.php
     + loader.php
     + RequestParser.php
+ modules
     + acl (modul pro řízení přistupových rolí)
     + admin (administrační modul)
     + page (modul pro statické stránky)
+ index.php
```