Mendelova univerzita v Brně
Provozně ekonomická fakulta

Interaktivní hra na propagaci univerzity

Diplomová práce

Vedoucí práce:
Ing. Pavel Turčínek, Ph.D.

Bc. Karel Barák

Brno 2015

Tímto děkuji svému vedoucímu práce Ing. Pavlu Turčínkovi, Ph.D. za stálou otevřenost a ochotu pomoci v jakékoliv situaci. Právě díky této opoře bylo jednodušší celou práci zvládnout a úspěšně dokončit. Také děkuji svoji rodině a přátelům za podporu v průběhu celého studia.

Čestné prohlášení

Prohlašuji, že jsem tuto práci: **Interaktivní hra na propagaci univerzity** vypracoval samostatně a veškeré použité prameny a informace jsou uvedeny v seznamu použité literatury. Souhlasím, aby moje práce byla zveřejněna v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů, a v souladu s platnou *Směrnicí o zveřejňování vysokoškolských závěrečných prací.*
Jsem si vědom, že se na moji práci vztahuje zákon č. 121/2000 Sb., autorský zákon, a že Mendelova univerzita v Brně má právo na uzavření licenční smlouvy a užití této práce jako školního díla podle § 60 odst. 1 Autorského zákona.
Dále se zavazuji, že před sepsáním licenční smlouvy o využití díla jinou osobou (subjektem) si vyžádám písemné stanovisko univerzity o tom, že předmětná licenční smlouva není v rozporu s oprávněnými zájmy univerzity, a zavazuji se uhradit případný příspěvek na úhradu nákladů spojených se vznikem díla, a to až do jejich skutečné výše.

V Brně dne 12. května 2015 .....

Abstract

Barák, K. Interactive game for university propagation. Master thesis. Brno: Mendel University, 2015.

Master thesis describes design and implementation of interactive epistemic web game. Then offers overview of developing possibilities in area of web applications and helps to orientate oneself in huge amount of developing tools, which are available these days. Main part is design of data model for information storing, application design, implementation and testing. Thesis also deeply analyses real-time web based applications issue and submits solution.

Key words

AngularJS, Crossbar.io, WebSocket, Nette Framework, MySQL, web application, master thesis.

Abstrakt

Barák, K. Interaktivní hra na propagaci univerzity. Diplomová práce. Brno: Mendelova univerzita v Brně, 2015.

Diplomová práce popisuje návrh a implementaci interaktivní webové vědomostní hry. Dále nabízí přehled možnosti v oblasti vývoje webových aplikací a pomáhá zorientovat se ve velkém množství vývojových nástrojů, které jsou nyní k dispozici. Hlavní částí práce je návrh datového modelu pro uchování informací, návrh aplikační logiky, implementace a testování. Práce také důkladně rozebírá problematiku komunikace v reálném čase v prostředí webového prohlížeče a předkládá řešení.

Klíčová slova

AngularJS, Crossbar.io, WebSocket, Nette Framework, MySQL, webová aplikace, diplomová práce.

Obsah

1 Úvod

První webové stránky vznikaly na počátku 90. let minulého století. Původně sloužily pouze k zobrazování jednoduchých textových dokumentů a byly převážně statické.

Postupem času, společně s probíhajícím vývojem, se webové stránky stávají stále více dynamické a nabízí více možností. Na začátku tohoto století nastává jistý zlom. Webové stránky začínají být vnímány jako komplexní a interaktivní přenosový mechanismus.

V reakci na tuto možnost začínají vznikat rozsáhlé a složité aplikace, které jsou schopny fungovat pouze ve webovém prohlížeči. Jedná se o širokou škálu aplikací, avšak pro tuto práci je zaměřena pozornost hlavně na hry. Důvodem je totiž stále větší přesun zábavy do prostředí Internetu. Lidé se rádi baví a právě v případě webového prostředí mohou zábavu prožívat. Většinou k tomu není zapotřebí nic speciálního, vystačí si s připojením k Internetu a s webovým prohlížečem. Instalace žádných dalších speciálních nástrojů nebývá podmínkou. Velké množství her je tak vytvářeno právě pro tuto platformu. Jedná se různé nové hry, ale i hry starší, které si zachovaly svoji popularitu a byly svými nadšenci znovu přeprogramovány k obnovení hratelnosti.

V této oblasti tak jsou vyvíjeny stále nové technologie, které vývojářům umožňují více propracovaných možnosti.

Vývoj stále probíhá, avšak jaký je aktuální stav? Jak se lze jednoduše zorientovat v nepřeberném množství možností a hlavně, jak vybrat nejvhodnější přístup k tvorbě webové aplikace (hry)? Tato práce přináší odpovědi na většinu těchto otázek a ukazuje, jak lze k vývoji přistupovat.

2 Cíl práce

Cílem práce je navrhnout a vytvořit interaktivní vědomostní hru, která bude sloužit k propagaci Mendelovy univerzity a Brna. Tato hra bude určena k získávání nových a zajímavých informací zábavnou formou.

Hra bude vytvořena jako webová aplikace, univerzální a nezávislá na druhu softwaru či hardwaru, na kterém bude spuštěna. Celý proces tvorby bude rozdělen na dílčí kroky, mezi které lze zařadit důkladné prostudování problematiky vývoje interaktivních webových aplikací se zaměřením na MVC nástroje, návrh datového modelu pro ukládání všech důležitých informací, návrh vlastní aplikace, její následná implementace, testování a celkové zhodnocení s nastíněním možností, jak pokračovat v budoucnu.

Hlavní myšlenka hry spočívá v interakci více hráčů v reálném čase. Nejdůležitějším úkolem je tedy nalézt řešení, které tuto interakci umožní a to způsobem, který je dostatečně univerzální a poskytuje otevřené možnosti dalšího rozšíření.

3 Přehled literatury a pramenů

3.1 HTTP

Hyper Text Transfer Protokol je aplikační protokol pro WWW. Tento protokol definuje, jak jsou zprávy formátovány a přenášeny a jaké akce provádí web servery a prohlížeče při určitých příkazech.

HTTP je nazýván bezestavovým protokolem, protože každý příkaz je vykonáván nezávisle, bez znalostí o příkazu, který byl vykonáván před ním. Toto je jeden z hlavních důvodů, proč je obtížné implementovat webové stránky, které reagují inteligentně na uživatelský vstup. Tento nedostatek je řešen dalšími technologiemi jako například ActiveX, JavaScript a Cookies. (HTTP, 2015)

HTTP funguje na principu požadavek/odpověď. Klient zašle požadavek na server ve formě nějaké metody, URI, verze protokolu, následované informacemi o klientovi. Server odpovídá stavovým řádkem, obsahujícím verzi protokolu a zprávu o úspěchu či chybě, následovaný informacemi o serveru. HTTP silně závisí na modelu klient/server, kdy klient vždy začíná komunikaci s požadavkem na data.

HTTP dále používá stavové kódy. Tyto zprávy jsou zasílány jako odpovědi serveru a slouží k identifikaci situace, která na serveru nastala. Např. stavový kód „404 File Not Found“ znamená, že požadovaný zdroj nebyl nalezen. (RFC 2616, 1999)

3.2 AJAX

AJAX byl velkým krokem ve vývoji interaktivních webových aplikací. Poprvé se veřejně objevil v roce 2005. Tímto krokem nebylo nadále nutné znovu načítat webovou stránku při každém zásahu uživatele do aplikace. Bylo možné asynchronně volat procedury na serveru a překreslit pouze určitou část stránky na základě vrácených hodnot. AJAX však nebyl schopný pokrýt aktualizace, které mohou přicházet ze serveru. Možnou technikou bylo stále se dotazovat na server, jestli neproběhla nějaká změna, avšak tato technika je výkonově náročná a těžko použitelná. (Adaptive Path, 2005)

3.3 HTTP Long Polling

Právě díky silné závislosti HTTP na modelu klient/server nebylo možné získat od serveru data či odpověď nezávisle. K překonání tohoto nedostatku byla implementována technika HTTP Long Polling. Server v tomto případě nechává požadavek otevřený tak dlouho, dokud nejsou nová data k dispozici. Jakmile jsou tato data dostupná, server odesílá odpověď. Klient data přijme a okamžitě zasílá další požadavek na server, který zůstává otevřený do příchodu nových dat.

U techniky Long Polling je však potřeba brát v úvahu několik zásadních věcí. Jak s rostoucím použitím vhodně spravovat serverovou část aplikace? Jak obnovit

spojení po jeho přerušení? Lze nějakým způsobem řídit frontu zpráv a získat ztracené zprávy? Při tvorbě aplikace využívající HTTP Long Polling je nutné vytvořit vlastní systém řízení komunikace, který bude zodpovědný za aktualizování, správu a rozšiřování serverové infrastruktury. (PubNub, 2014)

3.4 WebSocket

Websocket protokol umožňuje obousměrnou komunikaci mezi klientem a serverem, která probíhá přes jedno TCP spojení. Tento protokol byl navržen tak, aby mohl být implementován ve webových prohlížečích i na webových serverech. Svoji strukturou přináší mnoho výhod a vnáší tak plno nových možností do vývoje webových aplikací.

Websocket spojení jsou zahajována přes HTTP, čímž je zaručena zpětná kompatibilita a po úvodní kontrole spojení již může klient se serverem komunikovat v obou směrech. (RFC 6455, 2011)

3.5 WAMP

Zkratka WAMP je často využívána ve spojitosti se skupinou serverových technologií, konkrétně se jedná o Windows, Apache, MySQL a PHP. V tomto případě však uvažujeme WAMP jako protokol (Web Application Messaging Protocol). Jedná se o protokol, který byl vytvořen k uchopení a využití všech možností, které Websockety nabízí. Ve své podstatě lze WAMP zařadit jako otevřený standard a subprotokol pro Websockety. Umožňuje použití různých technologií, procesů a zařízení tak, aby komunikovaly v reálném čase. S použití WAMP protokolu lze vytvářet distribuované systémy, kde jsou jednotlivé uzly volně spojeny, a komunikace probíhá v reálném čase.

Samotný Websocket protokol funguje pouze s nízkou úrovní komunikace a umožňuje spíše hrubý způsob zasílání zpráv. To je přesně důvod, proč přichází WAMP protokol. WAMP využívá formát JSON[1] k serializaci zpráv. JSON serializace je preferovaný způsob komunikace pro WAMP, avšak protokol může fungovat i se serializací MsgPack.

WAMP protokol byl navržen s myšlenkou, aby nebyl závislý na určitém programovacím jazyku, ale aby mohl být využit transparentně v kombinaci s jakýmkoliv jiným jazykem, stačí, aby v něm byla pro WAMP zabudovaná podpora. Toto řešení přináší velkou výhodu v budování aplikací napříč technologiemi. (WAMP, 2012–2014)

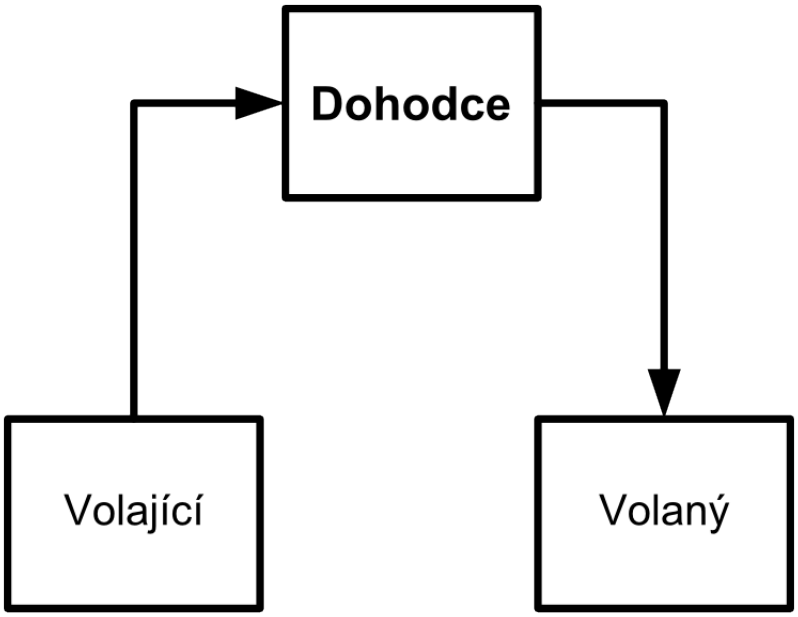
3.5.1 RPC

RPC (Remote Procedure Call) je jeden z nástrojů, které WAMP nabízí. RPC umožňuje vzdálené volání funkcí z jiných zdrojových kódů. Jedná se o asynchronní, rychlé

[1] JavaScript Objektová Notace – Formát pro výměnu dat. Jednoduchý na rozebrání i generování. (JSON, 2006)

a čisté řešení, které je velice jednoduché. Parametry a návratové hodnoty funkcí mohou být pole, mapování, čísla či řetězce. RPC funguje mezi rozdílnými programovacími jazyky a taktéž může fungovat mezi lokálními procesy či napříč Internetem. Každý klient potřebuje určitou knihovnu, aby mohl využívat WAMP ve svém kódu. Navíc je zapotřebí centrální server mezi všemi zúčastněnými klienty. V současné chvíli existuje velké množství knihoven a serverových řešení, které jsou zdarma a připraveny k použití. (WAMP, 2012 – 2014)

Obrázek 1 ukazuje princip RPC. Volající chce zavolat funkci, která se nachází někde jinde, odešle požadavek na Dohodce, ten má informace o tom, kde se funkce nachází a tento požadavek přepošle Volanému.

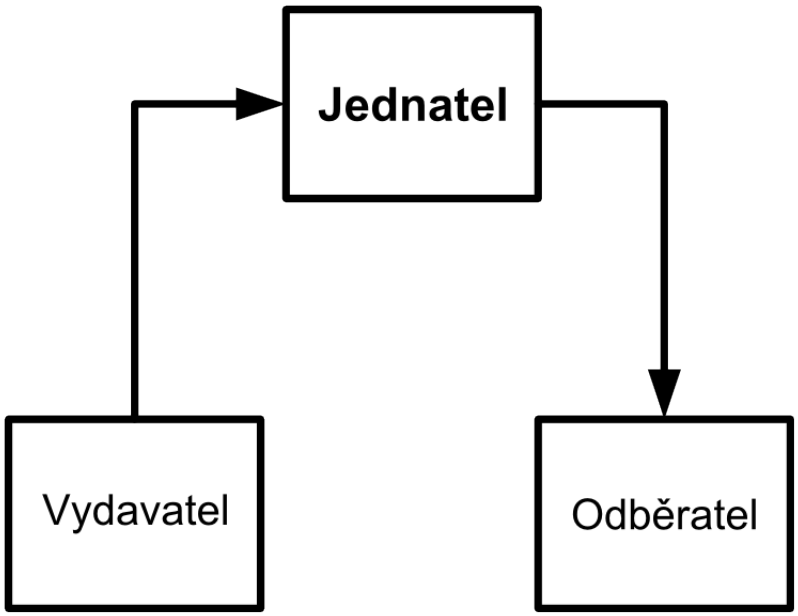


Obrázek 1: WAMP princip RPC

3.5.2 PUB/SUB

PUB/SUB (Publish–Subscribe) je vzor zasílání zpráv v softwarové architektuře, ve kterém vystupují odesílatelé zpráv (nazývaní Vydavatelé) a příjemci zpráv (nazývaní Odběratelé). Komunikace následně probíhá způsobem, kdy příjemci zpráv vyjádří zájem o určité téma. Na druhé straně Vydavatelůé zpráv mohou zveřejňovat zprávy pro určitá témata. Komunikace tak probíhá bez bližší vzájemné znalosti mezi Odběrateli a Vydavateli a hlavně není přímá, ale zprostředkovaná přes témata. Všichni, kdo vyjádřili zájem o určité téma, tak obdrží zprávu, když některý z Vydavatelů zveřejní zprávu pro toto téma. Jedná se o velice podobný přístup jako u RPC, avšak s důležitým rozšířením. V tomto případě lze poslat zprávu žádnému nebo N klientům. V případě RPC se jedná pouze o jednoho klienta. Na druhou stranu zde není žádná možnost obdržet návratovou hodnotu. (WAMP, 2012 – 2014)

Obrázek 2 znázorňuje princip PUB/SUB. Vydavatel s Odběratelem o sobě nevědí. Vše zprostředkovává Jednatel, který udržuje informace o Odběratelích a kdykoliv je nová informace vydána, zařídí správné doručení informace.



Obrázek 2: WAMP princip PUB/SUB

3.5.3 WAMP implementace

Pro to, aby mohl vývojář začít s používáním WAMP protokolu, může implementovat vlastní řešení dle předepsaných specifikací nebo může využít již vytvořené implementace, které jsou pravidelně kontrolovány lidmi, kteří vyvinuli WAMP protokol. Vždy aktuální list těchto implementací je udržován na stránkách WAMP protokolu.

WAMP implementace lze rozdělit do dvou skupin. Jednou skupinou jsou knihovny pro vytváření aplikačních komponent a tou druhou jsou řešení pro propojení jednotlivých komponent. Na listu jsou řešení pro mnoho různých programovacích jazyků. (WAMP Libraries, 2012 – 2014)

3.5.4 WAMP v1 vs. WAMP v2

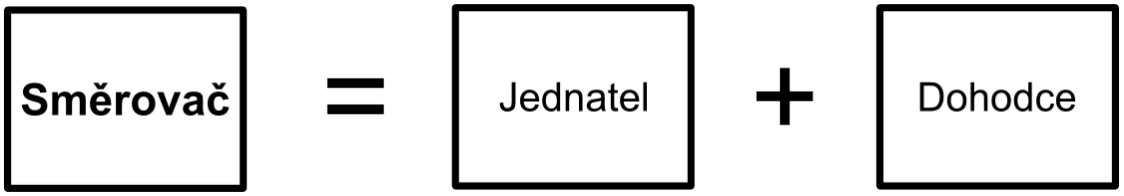
Jelikož se v případě WAMP protokolu jedná o celkem mladou technologii, která je stále vyvíjena a prochází změnami, došlo během zpracovávání této diplomové práce k vydání WAMP v2. Nová verze protokolu tak zavrhla původní verzi WAMP v1. Tato změna měla zásadní dopad na použité řešení při implementaci. Blíže bude tato skutečnost popsána v kapitole 6.6 (WAMP, 2012 – 2014)

3.5.5 Směrovač

Jak již bylo zmíněno, pro komunikaci více klientů je zapotřebí alespoň jeden centrální server, který bude obstarávat jednotné směrování napříč celou aplikací či systémem. Pro pochopení jeho funkce je nejlepší srovnání s modelem klient/server.

Volající potřebuje mít znalost o tom, kde se Volaný nachází a jak ho lze dosáhnout. V tomto případě se objevuje velice silná vazba mezi Volajícím a Volaným, což není ideální, protože aplikace se mohou velice rychle stát komplexními a těžko udržitelné. Silná vazba byla velice rychle rozpoznána a vedla k přechodu na vzor PUB/SUB. K předání zpráv mezi Vydavatelem a Odběratelem většinou slouží prostředník nazývaný Jednatel. Jednatel udržuje znalost o tom, kdo je aktuálně zaregistrován jako příjemce zpráv k určitému tématu. Ve chvíli, kdy některý Vydavatel zpráv zveřejní zprávu k určitému tématu, Jednatel určí množinu všech Odběratelů a těm zprávu zašle. Právě proces určení množiny příjemců a následné zaslání zprávy lze nazvat směrováním.

V případě RPC je vztah mezi Volajícím a Volaným opět rozložen a zprostředkující prostředník se nazývá Dohodce. Dohodce je zodpovědný za směrování Volajícího k Volanému. Ani jeden z účastníků RPC netuší, kde se ten druhý nachází a ani to není jejich starost, tato znalost striktně závisí na Dohodci. Když chce Volající volat určitou proceduru, zná pouze abstraktní jméno procedury ve formě URI a parametry, které chce proceduře předat. Úkolem Dohodce je podívat se do svého seznamu, zjistit kde se nachází účastník s touto zaregistrovanou procedurou a tu zavolat.



Obrázek 3: WAMP směrovač

Ve chvíli, kdy se spojí Jednatel a Dohodce z předchozích příkladů, vzniká to, čemu se ve WAMP terminologii říká Směrovač (Router). Jedná se o velice flexibilní řešení. (WAMP, 2012 – 2014)

3.6 MVC frameworky

V současné době existuje velké množství různých frameworků, které slouží především k usnadnění práce při vývoji webových aplikací. Framework umožní vývojáři soustředit se především na skutečný problém, přičemž pomáhá se znovu použitelností kódu, se zabezpečením aplikace a zvyšuje celkovou efektivitu práce.

Framework je sada tříd, knihoven či nástrojů, které mají za cíl usnadnění a zpřehlednění programátorovy práce. Většina frameworků využívá MVC architekturu,

která slouží k přehlednému strukturování psaného kódu. MVC je softwarová architektura, která odděluje model (Model), uživatelské rozhraní (View) a řídící logiku (Controller). Model je nejnižší vrstva architektury a slouží k obsluze dat. Uživatelské rozhraní slouží k zobrazování dat uživateli. Řídící logika je část, která zprostředkovává interakci mezi modelem a uživatelským rozhraním. (MVC Architecture, 2015)

3.7 PHP frameworky

Programovací jazyk PHP slouží v této práci k zprostředkování serverové části aplikace. V roce 2014 se podle statistik uvádí, že PHP je použito na 82,1 % všech serverových částí webových stránek, které jsou spuštěny a jejichž programovací jazyk je známý. (W3Techs, 2015)

Pro samotnou práci lze využít čistě PHP jazyk, avšak existuje velký počet frameworků, které dokáží značně zjednodušit práci. Pojďme si v následujících podkapitolách některé z nich představit.

3.7.1 Laravel

Laravel je šířen jako volně dostupný open source webový aplikační Framework, který byl navržen pro vývoj MVC webových aplikací (distribuovaný pod MIT licencí[2]). Laravel se snaží především o jednoduchou expresivní syntaxi se zaměřením na zjednodušení běžných úloh, jako je autentizace, autorizace, směrování, relace, ukládání do rychlé paměti. Dále má dobře čitelný kód a přehlednou dokumentaci, což vývojářům značně zrychluje programování. Funguje dobře se všemi druhy relačních databází. Dle srovnání frameworků se jedná o vítěze roku 2014. (Code Geekz, 2014, Laravel, 2015)

3.7.2 Symfony 2

Symfony Framework je opět postaven na MVC architektuře (distribuovaný pod MIT licencí). Skládá se z 29 znovupoužitelných PHP komponent, které může programátor využít při vývoji. Mezi komponenty patří především form (komponenta pro práci s formuláři), security (komponenta pro autentizaci, autorizaci), filesystem (komponenta poskytující nástroje pro správu souborů).

Komunitu Symfony tvoří velký počet vývojářů a tak se jedná o rychle se rozvíjející projekt. Symfony není nutné využívat pouze jako celek, ale např. jako doplněk k nějakému jinému oblíbenému frameworku (lze použít pouze potřebné komponenty). Na Symfony jsou založeny velice známé projekty jako fórum phpBB či redakční systém Drupal. (Code Geekz, 2014, Symfony, 2015)

[2]Licence označující volný software, která vznikla na Massachusetts Institute of Technology (MIT Licence, 2011)

3.7.3 Cake PHP

Cake PHP je v poslední době velice sledovaným frameworkem. V roce 2014 byla pro veřejnost vydána stabilní verze 3.0. Jedná se opět o MVC architekturu a celkově nabízí rychlý a jednoduchý vývoj (distribuovaný pod MIT licencí). Ve své základní struktuře je framework postaven na Ruby on Rails. Za výhodu Cake PHP je považováno generování kódu pro základní operace (CRUD) nad databází, žádná další XML nebo YAML konfigurace není nutná.

Z hlediska zabezpečení framework obsahuje nástroje pro CSRF[3] ochranu, pro vstupní validaci formulářů, ochranu proti SQL Injection[4] a prevenci proti XSS[5]. (Code Geekz, 2014, CakePHP v3.0, 2015)

3.7.4 Zend Framework

Zend Framework je opět založený na MVC architektuře (distribuovaný pod BSD licencí[6]). Zend byl velice dlouhou dobu nejznámější a nejpoužívanější PHP framework, díky svoji flexibilní modulární architektuře. Vývojář tak může využívat jen moduly, které potřebuje. Tyto moduly mezi sebou mají minimální počet závislost.

Zend Framework verze 1 dosáhl přibližně přes 15 milionů stažení. Nyní se framework nachází ve verzi 2, která z původní verze vychází a jedná se o 100% objektově orientovaný kód. K frameworku se také váže velká komunita. Zend Framework má také uzavřené partnerství s velkými společnostmi jako Microsoft nebo Google. Nicméně i přes všechny výhody ho mnoho vývojářů ve srovnání s dalšími frameworky považuje za extrémně komplikovaný, což vnímají jako zbytečné. (Code Geekz, 2014, Zend Framework, 2006–2015)

3.7.5 Nette Framework

V případě Nette Frameworku se oproti konkurentům jedná o český open source projekt (distribuovaný pod licencí GNU GPL[7]). Nette je založeno na MVC architektuře a podporuje přístup DRY (don't repeat yourself – neopakuj se) a KISS (keep it stupid simple – udržuj to stupidně jednoduché). Nabízí podporu AJAX, znovu použitelnost kódu, perfektní ladící nástroje, čistý objektový návrh využívající komponenty a událostmi řízené programování. Obsahuje v sobě vlastní vrstvu pro

[3]Typ zneužití webových stránek, kdy důvěryhodný uživatel provádí neautorizované příkazy (CSRF, 2010 – 2015)

[4]Forma útoku na webové stránky, kdy se útočník snaží vykonat neautorizované SQL příkazy (SQL Injection, 2015)

[5]Typ zneužití webových stránek, kdy útočník podstrčí kód (většinou JavaScript) který narušuje bezpečnost, či získává citlivé informace (XSS, 2015)

[6]Licence označující volný software, které vznikla na University of California at Berkeley (BSD Licence, 2004)

[7]Licence označující svobodný software. (GNU GPL, 2014)

komunikaci s databází, která využívá NotORM[8]. Šablonovací systém Latte je skvělým nástrojem, který umožňuje psát mnohem méně kódu a dědit mezi jednotlivými šablonami.

Hlavní předností Nette je pak zaměření na eliminaci bezpečnostních rizik (XSS, CRSF, URL attack, invalid UTF-8, únos relace, zcizení relace, zachycení relace). Jedná se tak o jeden z nejbezpečnějších frameworků.

Další výhodou je pak rozsáhlá česká komunita, pořádání pravidelných školení, na které se může kdokoliv přihlásit a podpora na českém fóru (anglické fórum je taktéž k dispozici). (Code Geekz, 2014, Nette Framework, 2008–2015)

3.8 JS frameworky

Java Script je další z programovacích jazyků, který se hojně používá ve webovém prostředí pro manipulaci HTML. Lze s ním naprogramovat chování webových stránek, aby byly interaktivní. Java Script je spouštěn pouze na počítači návštěvníka webové stránky. (W3Schools, 1999 – 2015, JavaScript Tutorials and Scripts, 2015)

Oproti jazyku PHP neexistuje tolik různých JS frameworků. Avšak v posledních 5 letech probíhá i v tomto směru určitý posun. Java Script slouží v této práci k vytvoření klientské části aplikace, která bude přístupná uživatelům.

3.8.1 AngularJS

Angular byl vydán v roce 2009, má MVC architekturu a je nejstarší ze všech zmiňovaných frameworků (je distribuován pod MIT licencí). Díky svému stáří má taktéž největší aktivní komunitu. Specialitou Angularu je dvou směrné datové spojení, dosazování závislostí (dependency injection), jednoduché testování a rozšiřitelnost jazyka HTML za použití direktiv. Google postupem času projekt převzal a začal sponzorovat, díky tomu je Angular open source, tak jak je známý dnes.

Součástí Angularu je šablonovací systém, který využívá syntaxe dvojitých složených závorek. Základní stavební kameny každé Angular aplikace lze rozdělit do šesti skupin. Jedná se o Controllers, Directives, Factories, Filters, Services a Views. View je zobrazení, které se ukáže uživateli. Service slouží ke komunikaci se serverem, Controller se stará o aplikační logiku na pozadí UI, atd. V případě Angularu není potřeba přistupovat k modelu za pomocí setterů a getterů. Lze přímo modifikovat jakoukoliv vlastnost a Angular sám zjistí, že proběhla změna. Častá kritika se pak týká Directives API a nepřehlednosti některých použitých konceptů (transclusion, scope hierarchy), dále pak Angular Expressions (využívání aplikační logiky v šabloně). (AngularJS, 2010 – 2015, ReadWrite: Web Apps, Web Technology Trends, 2014)

[8]PHP knihovna sloužící k zjednodušení práce s daty v databázi s vynikajícím výkonem (NotORM, 2010)

3.8.2 Backbone.js

Backbone vyšel v půlce roku 2010, také se jedná o MVC framework a velikostí komunity se nejvíce blíží Angularu (je distribuován pod MIT licencí). Backbone si rychle získal popularitu a stal se plnohodnotnou náhradou za těžkopádné MVC frameworky. Jeho hlavní výhodou je, že je nejmenší ze všech frameworků (obsahuje minimum kódu).

Backbone ve svém úplném základu nemá žádný šablonovací systém, ale mohou do něj být integrovány různé nástroje třetích stran. Základní možností je Underscore, jelikož se jedná o závislost Backbone. Nevýhodou Underscore je, že je až příliš elementární a většinou je jeho použití možné pouze v kombinaci s dalším JS kódem. Výhodami Backbone je tedy jeho minimální velikost, dále lineární křivka učení a také to, že obsahuje pouze pár základních konceptů. Mezi ně patří Models, Collections, Views a Routes. Má výbornou dokumentaci a dokonce i okomentovanou verzi celého frameworku. Vývojář se tak může s celou funkčností seznámit během hodiny. Tím, že je Backbone tak malý a jednoduchý, je možnost si jej dostatečně upravit a postavit na něm svoje vlastní specifické řešení (za použití dalších doplňkových nástrojů třetích stran). To co je hlavní přednosti Backbone je současně i jeho hlavní nevýhodou. Backbone neposkytuje žádnou základní strukturu aplikace, ale pouze poskytuje nástroje k jejímu vytvoření. Zbytek je pak kompletně na vývojáři. Například kompletně chybí řízení paměti. Je pravda, že existuje velké množství doplňkových nástrojů, ale vybrat ten správný, který se zrovna do projektu hodí, zabere určitý čas a průzkum. Oproti Angularu zde nefunguje dvou směrný data binding a tak je zapotřebí napsat hodně kódu na ošetření změn v šabloně a modelu. (Backbone, 2015, ReadWrite: Web Apps, Web Technology Trends, 2014

3.8.3 Ember.js

Ember je ze všech frameworků nejmladší. Taktéž má MVC architekturu a uveden byl až v roce 2011 (distribuován je pod MIT licencí). Jeho historie však sahá do roku 2007, kdy byl nejprve vyvíjen společností SproutIt a následně společností Apple. Velikost aktivní komunity je o něco menší než u Backbone.

Ember v současnosti využívá šablonovací systém Handlebars, který je rozšířením pro populární šablonovací systém Mustache. Handlebars nerozumí DOM, všechno co dělá, je pouze transformace řetězců. Ember se pak celkově soustředí hlavně na zjednodušení práce. Místo psaní opakujícího se kódu dokáže Ember automaticky rozpoznávat jména rout a controllerů při definici zdrojů. Oproti prvním dvěma frameworkům přichází s plnohodnotným data modulem, který velice dobře komunikuje s Ruby-on-Rails serverem nebo s jakýmkoliv REST JSON API, které vyhovuje jednoduché množině konvencí. Výkon byl hlavním záměrem při vývoji Emberu a tak framework využívá The Run Loop (slouží k dávkování a řazení, vytváří efektivnější běh), které zaručuje, že aktualizovaná data v modulu aktualizují pouze DOM elementy, kterých se změna týká. Nevýhodou Emberu je velice nepřehledná dokumentace, jelikož jeho API se mnohokrát změnilo před tím, než se stalo stabil-

ním. Některé části dokumentace tak nejsou nadále platné, ale stále zůstávají, což může být pro začínající vývojáře matoucí. Další nevýhodou je slabší šablonovací systém, který nutí zapisovat mnoho <script> tagů do kódu a u větších projektů se tak často narazí na nepřehlednost. Tato vlastnost navíc dokáže tvořit potíže při integraci s jinými frameworky či CSS styly. (Ember, 2015, ReadWrite: Web Apps, Web Technology Trends, 2014)

3.9 Responzivní design

Ve světě webdesignu je responzivní design pojem vcelku mladý. Poprvé byl použit v článku v roce 2010. Slouží k tomu, aby uživateli co nejvíce zpříjemnil práci. Cílem responzivního designu je vytvářet stránky tak, aby měl uživatel příjemný pocit z jejich prohlížení. Typická je snadná navigace, čitelnost a jeho použitelnost na různých typech přístrojů od osobních počítačů až po chytré telefony.

Nelze považovat responzivní design a design pro mobilní zařízení za stejný pojem. S responzivním designem vytváříme webové stránky, které reagují na velikost prohlížeče a mohou tak být efektivně zobrazeny na mobilních zařízeních, avšak souvislost patří čistě k designu webového prostředí.

Efektivní webové stránky pak mohou být prohlíženy na zařízeních s různým rozlišením, ať už se jedná o dříve běžné rozlišení 1024 × 768 pixelů, FullHD rozlišení 1920 × 1080 pixelů, 4K rozlišení v hodnotách 4096 × 2160 pixelů či cokoliv mezi těmito hodnotami. Webové stránky budou vypadat dobře i na tabletech a chytrých telefonech.

Hlavní prioritou je zaměřit se na obrazovou kvalitu a najít správnou rovnováhu mezi délkou načítáním stránek a kvalitou zobrazení, čehož se dá dosáhnout za pomoci vhodně zapsaného CSS. S touto skutečností budou souhlasit hlavně uživatelé mobilních telefonů, kteří mohou mít v určitých situacích k dispozici pouze omezenou rychlost přenosu dat.

Součástí responzivního designu je i sazba, která by neměla mít univerzální velikost. Font, který vypadá atraktivně na stolním počítači, může být problematický na mobilním zařízení, tedy responzivní design se týká i typografie. Nejdůležitějším aspektem responzivní typografie je délka řádku a kvůli snadné čitelnosti by měla být sazba optimalizována na šířku obrazovky. Sazba by měla být snadno čitelná ve vertikálním směru, je tedy možné nastavit mezery mezi řádky u větších bloků na 140 % oproti výchozí hodnotě a u menších obrazovek může být mezířádkový proklad ještě větší.

Při shrnutí jde tedy o vytvoření webu, který bude dobře vypadat, hladce fungovat a bude příjemně použitelný pro uživatele bez ohledu na zařízení, na kterém bude web prohlížen. (Interval, 2013)

3.10 Databáze

Součástí práce je návrh databáze. Databáze je navržena tak, aby ukládala informace multijazyčně. Existuje několik přístupů k návrhu multijazyčné databáze. V následujících podkapitolách jsou popsány všechny přístupy s jejich výhodami a nevýhodami.

3.10.1 Sloupcový přístup

V případě sloupcového přístupu se jedná o nejjednodušší možné řešení. Pro každý sloupec, který potřebujeme překládat, vytvoříme pro každý jazyk překladu jeden sloupec navíc. Při dotazech pak vybíráme sloupce pro potřebný jazyk.

Výhodou tohoto přístupu je hlavně jednoduchost, není zapotřebí spojovat tabulky mezi sebou a taktéž databáze neobsahuje žádné duplicity. Pro každý záznam je zde pouze jeden řádek a v něm se opakují sloupce překladů.

Nevýhodou je správa takové databáze, při 2–3 jazycích se jedná o vhodné řešení, avšak při velkém počtu jazyků velikost rychle narůstá. Dále není jednoduché přidat nový jazyk, pro každý takový jazyk je potřeba zásah do databáze a změna schématu. V neposlední řadě jsou ukládána volná místa, pokud nejsou zaznamenány všechny překlady. (CodeProject, 2014)

3.10.2 Řádkový přístup

Řádkový přístup je velice podobný tomu sloupcovému. Pro každý řádek, který potřebujeme přeložit, musíme vytvořit další řádek v databázi. Při dotazech pak budeme vybírat řádky podle potřebného jazyku.

Mezi výhody tohoto přístupu opět patří hlavně jednoduchost. Ani v tomto případě není zapotřebí spojovat jednotlivé tabulky.

Nevýhodou je opět správa takové databáze, například pro produkt, u kterého potřebujeme změnit cenu, bychom museli tuto operaci opakovat pro všechny řádky s překladem. Přidání nových jazyku je opět problematické, jelikož je nutné vytvořit a vložit nové záznamy pro každý jazyk. (CodeProject, 2014)

3.10.3 Přístup s jednou tabulkou

V tomto případě slouží k uložení překladů jedna tabulka. Toto řešení se jeví jako nejčistší z pohledu struktury databáze. Všechny texty jsou uloženy právě v jedné tabulce. Použití toho řešení je ideální pro velký počet jazyků.

Výhodou jsou použité normální formy tohoto řešení, společně s uchováváním všech informací na jednom místě. Navíc je jednoduché přidat nové jazyky v budoucnu bez nutnosti zásahu do schématu.

Nevýhodou je tvorba komplexních dotazů k získání potřebných informací. Tato komplexnost platí i pro vkládání, mazání a aktualizaci. Výhoda ukládání na jednom místě může být současně brána jako drobná nevýhoda, protože v případě chybějící tabulky může nastat globální problém. (CodeProject, 2014)

3.10.4 Přístup s více tabulkami

V posledním případě slouží k uložení překladů více tabulek. Pro každou tabulku, kterou chceme překládat, vytvoříme jednu tabulku navíc. Původní tabulka obsahuje všechny informace, které není nutné překládat a nová tabulka naopak obsahuje všechny informace, které potřebujeme uchovávat přeložené.

Výhodou tohoto řešení jsou opět použité normální formy . Opět je velice jednoduché přidávat nové jazyky, navíc není nutné zasahovat do schématu. Tabulky si zachovávají svoje jméno, není zapotřebí přidávat jazykové koncovky. Dotazy nad databází jsou opět velice jednoduché. Jedinou nevýhodou posledního přístupu je možnost zdvojení počtu tabulek v databázi. (CodeProject, 2014)

4 Metodika a použité metody

4.1 Nette Framework

Ze zmíněných PHP frameworků se jedná o jedny z nejpoužívanějších, avšak existuje jich mnohem větší počet. Při výběru vhodného frameworku může záležet na více faktorech, jako je např. výkon (kolik requestů dokáže obstarat v určitém čase), zabezpečení, zkušenosti, velikost aktivní komunity či integrace s dalšími nástroji.

Při výsledném výběru byly nejdůležitějším aspektem zkušenosti, které mám jako autor největší právě s Nette. Výhodou Nette je, jak už bylo zmíněno, velká míra zabezpečení aplikace, dále česká komunita, která je velice aktivní a taktéž schopnost integrace dalších nástrojů třetích stran.

Tato práce je implementována v programovacím jazyce PHP a je využito Nette frameworku, přesto se nejedná o jediné možné řešení. Dalo by se zajisté využít jiného frameworku, či dokonce jiných programovacích jazyků pro serverovou část aplikace, avšak pro tuto chvíli se vybrané řešení jeví jako ideální.

4.1.1 Composer

Composer je nástroj pro správu závislostí v PHP. Umožňuje deklarovat závislé knihovny, které v projektu používáme a ty jsou následně instalovány za nás. Tyto knihovny mohou mít samy o sobě další závislosti. Composer vyhledá, které verze kterých knihoven mají být nainstalovány a ty použije. Výhodou je jednoduchá aktualizace všech závislých knihoven, která probíhá automatizovaně.

Composer vyžaduje PHP 5.3+ a Git, aby mohl být používán. Git je distribuovaný systém správy verzí a právě z jeho databáze Composer čerpá. Navíc je multiplatformní a může být používán na Windows, Linux a OSX. Pro instalaci na Windows je potřeba stáhnout instalační balíček, zatímco pro instalaci v prostředí Linuxu stačí následující příkaz:

```
$ sudo apt-get install git
```

Pro globální instalaci na Linuxu lze použít následující dva příkazy. V případě, že příkaz umístíme do PATH, lze jej volat globálně.

```
$ curl -sS https://getcomposer.org/installer | php
$ mv composer phar /usr/local/bin/composer
```

Pro instalaci v prostředí Windows lze využít předpřipravený installer. Znovu je příjemnější přidat příkaz composer do PATH a volat jej tak jednoduše z konzole. (Composer, 2015)

Samotný Nette Framework pak stáhneme pomoci následující příkazu:

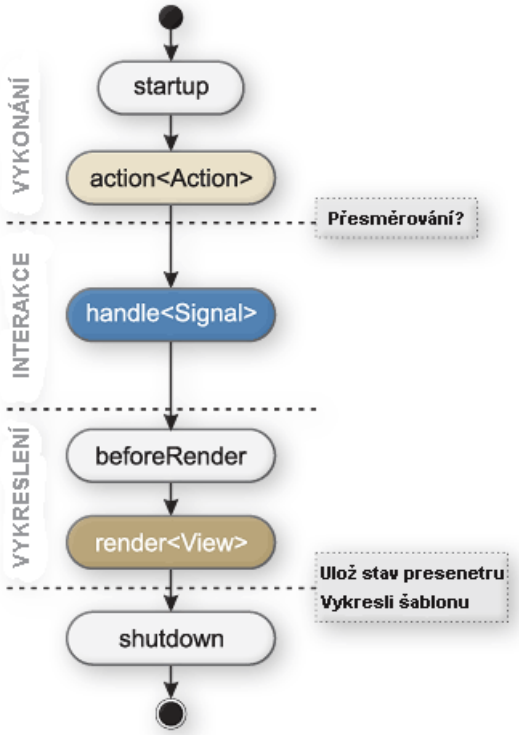
```
$ composer create-project nette/sandbox nazev-projektu
```

4.1.2 Struktura

Struktura Nette je následující.

- Model je datový a zejména funkční základ celé aplikace. Je v něm obsažena aplikační logika. Jakákoliv akce uživatele (přihlášení, změna hodnoty v databázi) představuje akci modelu. Model si spravuje svůj vnitřní stav a ven nabízí pevně dané rozhraní. Voláním funkcí tohoto rozhraní můžeme zjišťovat či měnit jeho stav. Model o existenci view nebo kontroleru neví.
- View je pohled, tedy vrstva aplikace, která má na starost zobrazování výsledku požadavku. Obvykle používá šablonovací systém (Latte) a ví, jak se má zobrazit ta která komponenta nebo výsledek získaný z modelu.
- Controller je řadič, který zpracovává požadavky uživatele a na jejich základě pak volá patřičnou aplikační logiku (tj. model) a poté požádá pohled o vykreslení dat. Obdobou kontrolerů v Nette Framework jsou presentery.

Presenter je objekt, který vezme požadavek přeložený směrovačem z http požadavku a vygeneruje odpověď. Odpovědí může být HTML stránka, XML dokument, soubor, JSON nebo přesměrování.



Obrázek 4: Životní cyklus presenteru v Nette Frameworku (Nette Framework, 2008 – 2015)

Akce presenteru způsobí volání metody render<Action>, nejedná se však o jedinou dostupnout metodu. Ihned po startu je presenteru je volána metoda startup, která inicalizuje proměnné nebo ověří uživatelská oprávnění. Metoda action<Action> je obdoba metody render<View>, protože mohou nastávat situace, kdy presenter provede určitý úkon a poté přesměruje jinam, z toho důvodu není ideální pojmenovávat metodu render, když nic nevykresluje. Metoda handle<Signal> zpracovává různé podpožadavky a je určena hlavně pro komponenty a zpracování AJAXových požadavků. Metoda beforeRender se volá před samotným vykreslením a může obsahovat například nastavení šablony či předání proměnných společných pro více view. Metoda render<View> potom slouží k předání potřebných dat do šablony.

Jednotlivé funkce, které jsou dostupné v presenteru, lze vidět na obrázku 3. Lze je rozdělit do tří sekcí. Celý obrázek pak znázorňuje životní cyklus presenteru. (Nette Framework, 2008 – 2015)

4.1.3 Dependency Injection

Dependency Injection je koncept, který slouží k odebrání zodpovědnosti třídám za získávání objektů, které potřebují ke své činnosti. Výsledný kód tak dostává z vnějšku vše co potřebuje a nemusí obsahovat žádné skryté závislosti.

Dependency Injection je možné využít více různými způsoby. Některými technikami pak jsou:

- Constructor Injection – závislosti jsou injektovány pomocí konstruktoruzjišťovat či měnit jeho stav. Model o existenci view nebo kontroleru neví.
- Property Injection – injektování změnou hodnoty veřejné vlastnosti třídy
- Method Injection – závislosti jsou injektovány prostřednictvím veřejné metody třídy (Nette Framework, 2008 – 2015)

4.1.4 Latte šablony

Nette šablony jsou prezentovány jako hlavní přednost Nette Frameworku, jelikož šetří čas a značně zpříjemní práci. Navíc zabezpečují výstup před zranitelnostmi. V šablonách se pak využívají tzv. makra neboli speciální značky (if, ifset, foreach, while, define block, atd.)

Makra lze obecně rozdělit do dvou skupin na párová a nepárová. N:makra slouží ke zkrácení zápisu párových maker (např. if… /if zapíšeme jako n:if). Nette obsahuje velké množství výchozích maker, avšak existuje zde možnost vytvářet vlastní makra.

Důležitým poznatkem je, že Latte šablony vypisované proměnné automaticky escapují a zabezpečují tak přímo proti Cross Site Scripting (XSS). Navíc disponují unikátní technologii Context-Aware Escaping, která dokáže rozeznat, ve které části dokumentu se makro nachází a podle toho zvolí správné escapování.

Součástí šablon jsou pak tzv. Latte filtry, což jsou modifikátory, které pomáhají formátovat data. Nette opět obsahuje velký počet standardních filtrů, ale programátor má opět možnost vytvářet vlastní filtry. (Nette Framework, 2008 – 2015)

4.2 AngularJS

Stejně jako u PHP frameworků se i u JS frameworků nejedná o přehled všech možností, ale pouze výběr těch nejhlavnějších. Mezi faktory výběru lze opět zařadit zkušenosti, velikost aktivní komunity či přístup k tvorbě aplikace (v každém z případů se totiž způsob velice liší).

Ember je velice výkonný, klade důraz na práci s daty a dokáže ušetřit mnoho psaní, avšak psaní šablon je nepřehledné. Backbone si zakládá na svém minimalismu, je jednoduchý na naučení, ale mnohdy je potřeba toto minimum zásadně rozpracovat na započetí práce. Angular přehledně rozšiřuje schopnosti jazyka HTML, jeho slabinou může být zmatek při kombinaci aplikační logiky s šablonami. Díky nulovým zkušenostem autora s komplexním programováním v JS (až na jQuery doplňky), je hlavním měřítkem výběru přistup k tvorbě aplikace, který se jeví nejpřehlednější v případě Angularu. Za výběr Angularu hovoří i největší aktivní komunita a velice přehledná dokumentace s velkým množstvím tutoriálů.

Stejně jako v případě PHP frameworků se ani zde nejedná o jediné možné řešení. Avšak pro tuto chvíli se Angular jeví jako nejvhodnější.

4.2.1 Node Package Manager

Node Package Manager je nástroj sloužící ke správě různých balíčků pro JavaScript vývojáře. Jak již z názvu vyplývá, je tento nástroj postaven na Node.js.

Node.js je platforma postavena nad Chrome's JavaScript runtime. Jedná se o softwarový systém navržený pro psaní webových aplikací, především webových serverů. Jedná se tak o přístup, který není zrovna tradiční. JavaScript byl dlouhou dobu považován za použitelný pouze na straně klienta. S vývojem Node.js se tento pohled postupně mění.

S NPM mohou JavaScript vývojáři publikovat svůj kód a ten může být znovupoužitelný dalšími vývojáři. NPM pak jednoduše kontroluje, jestli byly provedeny nějaké změny v kódu a tento kód udržuje aktualizovaný. Tyto znovupoužitelné kódy se nazývají balíčky nebo moduly. Balíček v sobě obsahuje jeden a více souborů a navíc package.json, ve kterém jsou zaznamenána metadata o balíčku. Hlavním úkolem balíčku je, aby kvalitně řešil určitý problém. Každý vývojář pak může použít balíčky, které zrovna potřebuje ve svém projektu. Výhodou je možnost získání a využití znalostí, které již někdo jiný má a které jsou veřejně poskytnuty. Balíčky mohou být nalezeny v rozsáhlé databázi, která je s tímto nástrojem spojena. Databázi lze prohlížet na stránkách NPM.

Instalace probíhá v následujícím pořadí. Nejprve je nutné nainstalovat Node.js, což je v prostředí Windows provedeno za pomocí příslušného instalačních balíčků. V případě Linuxu pak provedeno následujícím příkazem:

```
$ sudo apt-get install nodejs
```

Po nainstalování Node.js je zapotřebí nainstalovat samotný Node Package Manager, což lze opět provést za pomoci instalačního balíčku (Windows) nebo následujícího příkazu (Linux):

```
$ sudo apt-get install npm
```

Úspěšnou instalaci můžeme ověřit následujícími příkazy:

```
$ node -v
$ npm -v~
```

V obou případech je správnost instalace ověřena tak, že se vypíše verze příslušné instalace. (NPM, 2015)

4.2.2 Bower

Bower je jeden z balíčků, který lze nalézt v databázi NPM. Bower lze považovat za nástroj pro správu závislostí pro webové stránky. Otázkou by mohlo být, jaký je rozdíl oproti NPM? Jak již bylo zmíněno, NPM umožňuje komplexní správu JavaScript balíčků, zatímco Bower se zaměřuje pouze na front-end komponenty (CSS, HTML, JavaScript).

Místo nutnosti vyhledávat jednotlivé komponenty, stahovat je odděleně a následně je přidávat do projektu, nabízí Bower možnost uchovávat všechny informace v souboru bower.json. Tento soubor následně prochází a stáhne všechny závislosti na jedno místo, kde s nimi lze dále pracovat.

Bower potřebuje ke svému fungování Node.js, NPM a Git. Jedná se o nástroj, který funguje skrz příkazový řádek (nezávisle na OS). Pro instalaci stačí použít následující příkaz:

```
$ npm install -g bower
```

Komponenty jsou instalovány do složky bowerComponents. Pro instalaci komponenty lze využít následující příkaz:

```
$ bower install <package>
```

Balíčky v instalačním příkazu mohou být zapsány registrovaným jménem (např. jquery), zktratkou z githubu, koncovým bodem z githubu nebo URL adresou. Na stránkách nástroje Bower lze využít vyhledávač balíčků (vše tak lze hledat na jednom místě). Po vyhledání a instalaci všech balíčků jsou všechny informace zapsány

v souboru bower.json, avšak nikoliv reálně dostupné. Pro stažení všech souborů je nutné využít další příkaz:

```
$ bower init
```

K další práci se všemi nyní staženými balíčky lze poté využít další nástroje. (Bower, 2015)

4.2.3 Gulp

Gulp je další z balíčků z databáze NPM. Gulp navazuje tam, kde Bower končí a jedná se o tzv. task runner. Task runner lze využít k automatizaci procesů, které vývojář opakovaně provádí. Jedná se např. o minifikaci kódu, kompilaci, zjištění chyb. Instalace se opět provádí skrz příkazový řádek za použití následujícího příkazu:

```
$ npm install -g gulp
```

Pro Gulp existuje velké množství pluginů, které lze využít. Mezi hlavními lze zmínit Lint, který slouží k zjišťování chyb v napsaném kódu, Concat sloužící ke spojování souborů a Uglify, díky kterému můžeme minifikovat napsaný kód.

Pluginy nainstalujeme následujícím příkazem:

```
$ npm install gulp-jshint gulp-concat gulp-uglify gulp-rename --
save-dev
```

Po nainstalování všech pluginů je zapotřebí vytvořit soubor gulpfile.js a v něm definovat jednotlivé úkoly. Gulp má celkově pouze 5 metod. Jedná se o task, run, watch, src a dest, avšak toto jsou jediné metody, které jsou zapotřebí. Gulpfile.js potom může vypadat následovně. (Travis Maynard, 2013)

```
// Include gulp
var gulp = require(´gulp´);

// Include plugins
var jshint = require(´gulp-jshint´);
var concat = require(´gulp-concat´);
var uglify = require(´gulp-uglify´);

// Lint Task
gulp.task(´lint´,function(){
return gulp.src(´js/*.js´)
.pipe(jshint())
.pipe(jshint.reporter(´default´));
});

// Concentate & Minify JS
```

```
gulp.task(´scripts´, function(){
return gulp.src(´js/*.js´)
.pipe(concat(´all.js´))
.pipe(gulp.dest(´js´))
.pipe(rename(´all.min.js´))
.pipe(uglify())
.pipe(gulp.dest(´js´))
});

// Default Task
gulp.task(´default´, [´lint´, ´scripts´]);
```

Po uložení tohoto souboru můžeme začít Gulp používat. Když se přesuneme do příkazového řádku a do složky s projektem, stači napsat:

```
$ gulp
```

Tento příkaz spustí úkol s názvem default, tedy zkontrolování kódu a spojení i minifikaci. V případě, že chceme spustit pouze některý z úkolů, použijeme:

```
$ gulp lint
```

4.2.4 Data Binding

Data-binding slouží v Angularu k automatické synchronizaci dat mezi modelem a zobrazovanou komponentou. Způsob, kterým je toto provázání vytvořeno, umožňuje přistupovat k modelu jako k jednotnému zdroji všech pravdivých informací. Zobrazení je v podstatě projekcí modelu v každém okamžiku. Kdykoliv se model změní, změna je okamžitě reflektována, což platí v obou směrech.

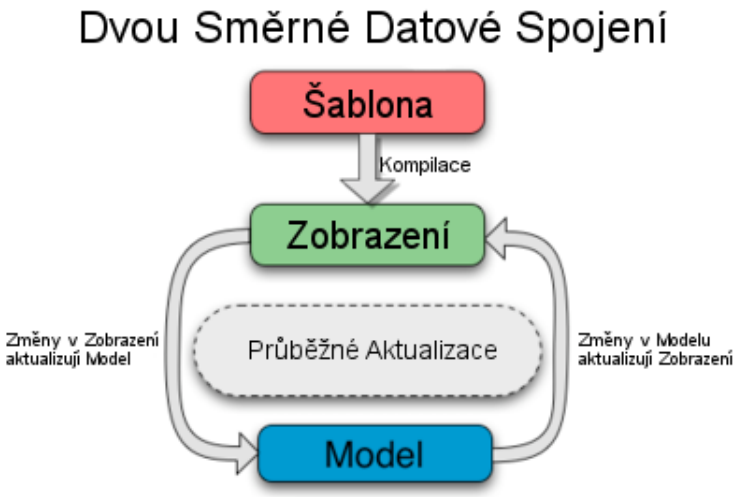
Funkčnost data-bindingu je zobrazena na následujícím obrázku.

4.2.5 Templates

V Angularu jsou šablony psané s pomoci HTML a navíc obsahují speciální elementy a atributy. Angular pak kombinuje šablonu s informacemi z modelu a controlleru a vytváří zobrazení, které se ukáže uživateli v prohlížeči.

Mezi typy elementů, které můžeme použít, patří:

- Direktiva – Rozšiřuje a vylepšuje stávající DOM element nebo reprezentuje znovupoužitelný DOM element (Document Object Model je JavaScriptová reprezentace obsahu stránky)
- Filter – Určitým způsobem formátuje data, která mají být zobrazena
- Markup – Angular obsahuje syntaxi dvojitých složených závorek, ta slouží k provázání výrazu s modelem



Obrázek 5: Obousměrný data-binding v AngularJS (AngularJS, 2010 – 2015)

```
<html ng-app="myApp">
 <!-- Body tag rozšířený o~direktivu z~MyController  -->
 <body ng-controller="MyController">
   <input ng-model="foo" value="bar">
   <button ng-click="changeFoo()">{{buttonText}}</button>
   <script src="angular.js">
 </body>
</html>
```

Příklad ukazuje HTML tag body rozšířený o některé Angular elementy. MyController je přiřazen za pomoci ng-controller. Input tag provázaný s modelem za pomoci ng-model a tlačítko, které po kliknutí vykoná funkci changeFoo() díky ng-click. Text tlačítka je navíc určen proměnnou buttonText, která se taktéž nachází v modelu.

V jednoduché aplikaci může postačit jeden HTML soubor, který obsahuje všechny potřebné výrazy. Ve složitějších aplikacích je možné vytvářet komplexní zobrazení, která obsahují více vkládaných segmentů. (AngularJS, 2010 – 2015)

4.2.6 Controllers

Controller je v podstatě funkce konstruktoru, která slouží k rozšíření scope (objekt odkazující na model aplikace). Ve chvíli, kdy je controller přiřazen k DOM elementu, vytvoří Angular novou instanci objektu controlleru. Controllery jsou určeny k nastavení počátečního stavu scope a k rozšiřování jeho chování. Naopak se nedoporučuje využívat controllery k manipulaci DOM elementů, formátování vstupů z formulářů, filtrování výstupu.

```
var myApp = angular.module('myApp',[]);

myApp.controller('MyController', ['$scope', function($scope) {
  $scope.buttonText = 'odeslat';
  $scope.foo = 'vstup';
}]);
```

Tento příklad ukazuje vytvoření nového modulu s názvem myApp a následné vytvoření controlleru MyController. Ve spojení s šablonou z předchozího příkladu se jedná o provázání funkcionality. Propojení způsobí, že tlačítko bude mít textovou hodnotu 'odeslat' a input bude obsahovat hodnotu 'vstup'. Musí souhlasit název modulu společně s hodnotou direktivy ng-app a také název controlleru s direktivou ng-controller.

Jako parametr funkce konstruktoru můžeme vidět předání řetězce scope, následovaného funkcí. Takové předání parametru zaručuje, že můžeme v controlleru použít objekt scope. (AngularJS, 2010 – 2015)

4.2.7 Scopes

Objekt scope odkazuje na model aplikace. Scope objekty jsou uspořádány v hierarchické struktuře a napodobují DOM strukturu aplikace. Jedná se v podstatě o propojení mezi controllerem a zobrazením.

V předchozím příkladu jsou přiřazeny dvě hodnoty k vlastnostem objektu scope. Jedna pro foo a druhá pro buttonText, což způsobí předání do šablony a následně do zobrazení. Direktivy i controllery mají odkaz na scope, avšak ne na sebe navzájem. Controller je tak vždy oddělen přímo od direktiv a od DOM.

Scope vytváří hierarchie a každá Angular aplikace má právě jeden základní scope, nicméně může mít více potomků. Některé direktivy totiž vytváří potomky objektu scope, ty jsou pak přiřazeny v hierarchii nadřazenému objektu. Ve chvíli, kdy Angular vyhodnocuje proměnnou buttonText, podívá se nejprve do objektu, který je přiřazen k danému elementu a zkontroluje, jestli má vlastnost buttonText. Pokud není tato vlastnost nalezena, pokračuje u nadřazeného objektu a takhle stále dokola, dokud nedosáhne základního scope. (AngularJS, 2010 – 2015)

4.2.8 Dependency Injection

Angular obsahuje subsystem, který má na starosti vytváření komponent, vyhodnocování jejich závislostí a poskytování těchto komponent na místech, kde jsou vyžadovány.

Jako komponenty mohou být vnímány služby, direktivy, filtry a animace. Všechny tyto komponenty jsou definovány konstruktorem a mohou být poskytnuty jako závislost při vytváření controlleru.

V předchozím případě při vytváření controlleru jsme mohli vidět, že jako závislost je předáván objekt scope. Existuje několik možností, jak závislost definovat.

- Anotace polem – Preferovaná možnost, v dokumentaci je možno nalézt všechny příklady s touto možností
- Anotace vlastností inject – Aby mohly minifikátory kódu přejmenovat parametry funkcí a stále byly schopny použít správné závislosti
- Implicitní anotace – Nejjednodušší možnost, kdy se předpokládá, že názvy parametrů funkce jsou názvy závislostí (AngularJS, 2010 – 2015)

4.2.9 Services

Služby jsou v Angularu objekty, které jsou přímo propojeny s DI (Dependency Injection). Služby mohou být použity ke sdílení určitého kódu napříč celou aplikací. Angular vytvoří instanci služby pouze v případě, kdy si o ni nějaká komponenta zažádá. Každá služba je navíc jako singleton, což znamená, že komponenty vždy obdrží pouze jednu instanci vygenerované služby.

Vývojáři mohou definovat svoje vlastní služby registrováním jména a továrničky, která službu vytváří. V následujícím příkladu je zobrazeno, jak se nová služba zapisuje. I služby mohou mít svoje závislosti, ty se předávají, jak již bylo zmíněno. (AngularJS, 2010 – 2015)

```
var myApp = angular.module('myApp', []);
myApp.factory('serviceId', function() {
  var shinyNewServiceInstance;
  // factory function body that constructs shinyNewServiceInstance
  return shinyNewServiceInstance;
});
```

4.3 Ratchet

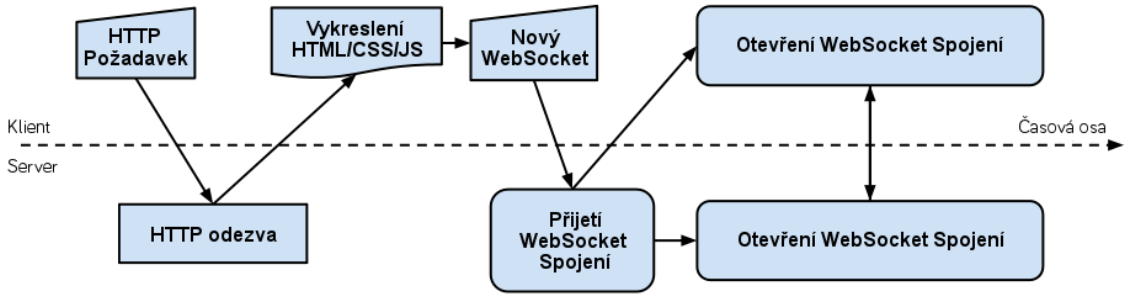
Ratchet je PHP knihovna, která zprostředkovává implementaci websocketů. Tato knihovna poskytuje nástroje, umožňující vytvářet serverové řešení aplikace, která funguje v reálném čase a je obousměrná mezi klientem a serverem.

Jakmile je vytvořeno spojení mezi klientem a serverem, toto spojení zůstává otevřené a obě strany mohou přijímat a odesílat zprávy, což znázorňuje obrázek 5.

Instalace probíhá jednoduše pomocí Composeru. K instalaci stačí použít následující příkaz.

```
$ composer require cboden/ratchet
```

Celá knihovna se skládá z několika komponent. Základní komponentou je IoServer. Tato komponenta slouží jako základ aplikace, který zpracovává nová připojení, přijímá a odesílá zprávy těmto připojením a připojení uzavírá. Další komponentou je WsServer, ten slouží ke komunikaci s prohlížečem a je kompatibilní se všemi protokoly, které se v základu k websocketům vztahují. Poslední komponentou je pak WampServer, který implementuje WAMP protokol.



Obrázek 6: Ratchet schéma (Ratchet, 2014)

Interface zprostředkující komunikaci se skládá ze 4 základních funkcí a jedná se o následující.

- OnOpen funkce je zavolána, když se připojí nový uživatel.
- OnMessage je volána v případě zprávy od uživatele
- OnClose je zavolána při uzavření připojení.
- OnError je zavolána v případě, že nastane chyba.

Kód pak vypadá následujícím způsobem.

```
namespace MyApp;
use Ratchet\MessageComponentInterface;
use Ratchet\ConnectionInterface;

class Application implements MessageComponentInterface {
    public function onOpen(ConnectionInterface $conn) {}
    public function onMessage(ConnectionInterface $from, $msg) {}
    public function onClose(ConnectionInterface $conn) {}
   public function onError(ConnectionInterface $conn, \Exception $e) {}
}
```

Ratchet je poskytován jako open source software. (Ratchet, 2014)

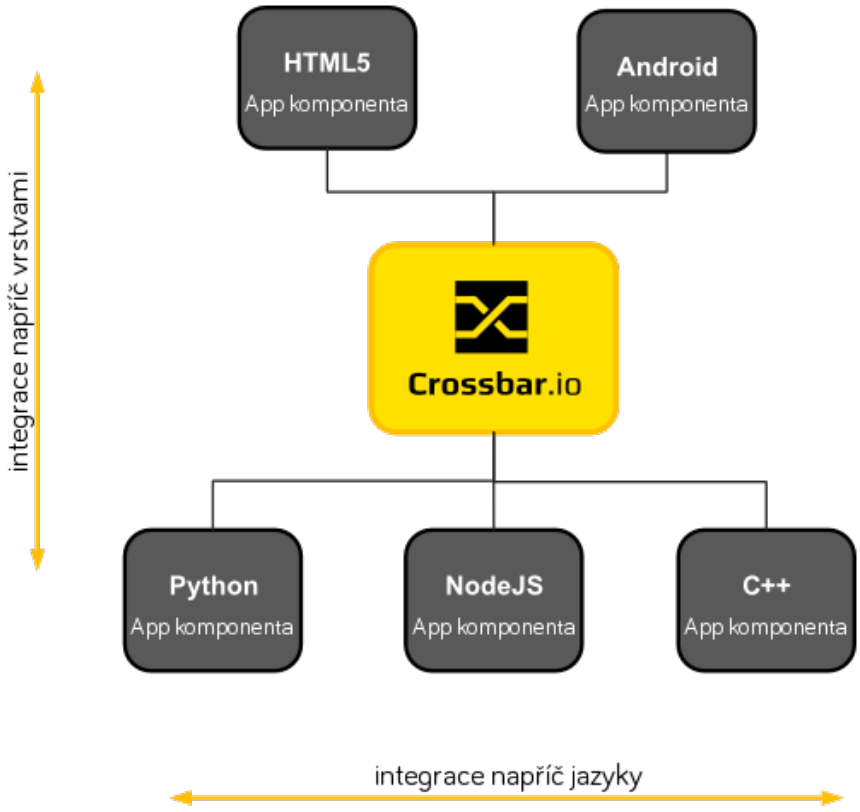
4.4 Crossbar.io

Crossbar.io je open source jednotný aplikační router, který implementuje WAMP protokol. Jedná se o již zmíněný subprotokol, který umožňuje volně propojovat různé komponenty či služby a ty spolu mohou komunikovat v reálném čase.

Crossbar.io směruje a přenáší zprávy mezi těmito komponentami, které jsou napsány ve WAMP klientských knihovnách (aktuálně jich je k dispozici 8). Každá komponenta může být napsaná v jakékoliv této knihovně a lze je jakkoliv kombinovat

dohromady. Crossbar.io tedy poskytuje možnost vytvářet aplikace, které fungují přes různé platformy, což je znázorněno na obrázku 6.

Crossbar.io je vyvíjen společností Tavendo, která má na starost i všechny záležitosti ohledně WAMP protokolu. Jedná se tedy o komplexní řešení, které je vyvíjeno na jednom místě. Kromě těchto dvou projektů má společnost Tavendo ještě třetí projekt, který se jmenuje Autobahn. Autobahn je implementace klientských knihoven v různých programovacích jazycích.

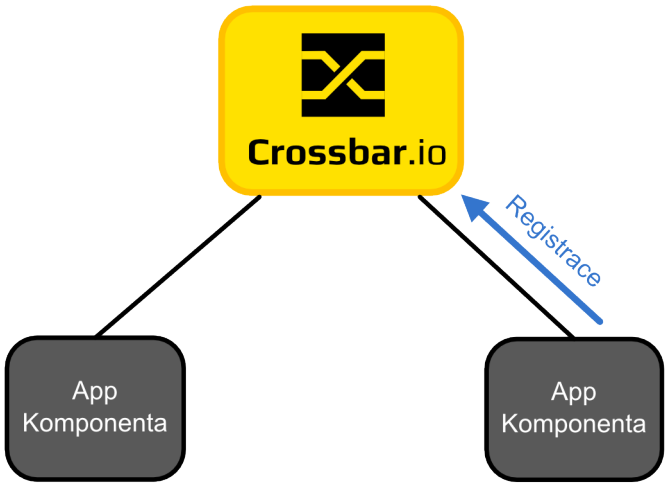


Obrázek 7: Příklad aplikační architektury s Crossbar

Crossbar.io je napsán v programovacím jazyce Python a jedná se o mutli uzlovou a multi procesní architekturu. Uzel je instance Crossbar.io, která běží na jednom stroji. Tyto jednotlivé instance mohou být propojeny ve shluk. Jednotlivé instance tak většinou poběží na různých zařízeních. Každý uzel je spouštěn ze samostatné složky, která obsahuje soubor s konfigurací uzlu, logy a případně bezpečnostní certifikáty.

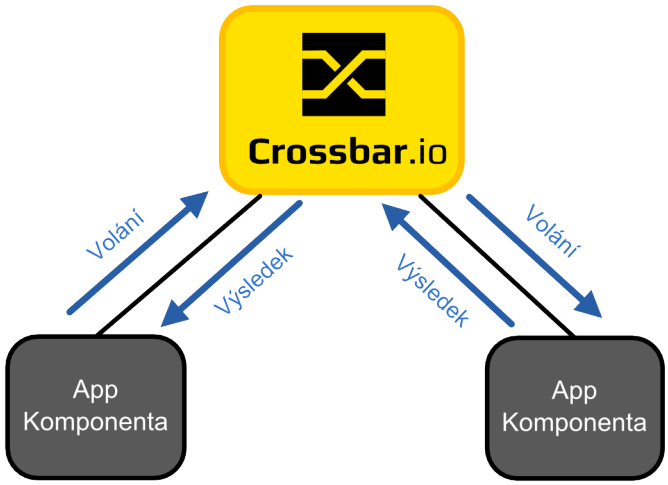
Pro bližší představu o funkčnosti jednotlivých nástrojů WAMP protokolu slouží následující obrázky. Obrázek 7 znázorňuje připojení dvou aplikačních komponent ke Crossbar.io. Jedna komponenta pak zaregistruje svoji proceduru. Ta je nyní dostupná ostatním komponentám.

Na obrázku 8 jsou opět znázorněny dvě aplikační komponenty. Jedna komponenta zavolá proceduru z druhé komponenty. Tento požadavek je odeslán na



Obrázek 8: Registrace procedury pro volání (Crossbar.io, 2013 – 2015)

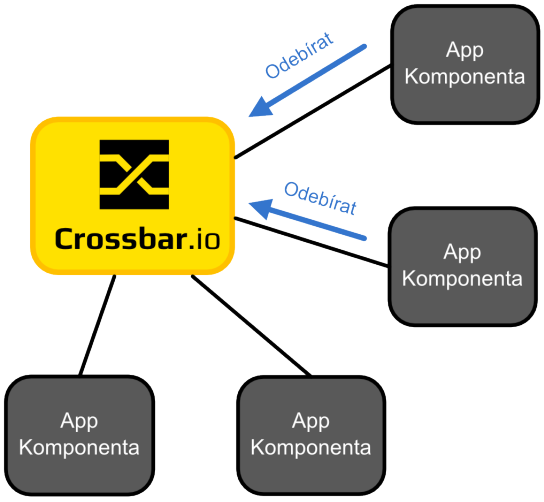
Crossbar.io, který vyhodnotí, jaké aplikační komponentě musí požadavek předat a předání provede. Výsledek procedury je opět vrácen na Crossbar.io a ten předá výsledek opět na správnou komponentu.



Obrázek 9: Volání procedury (Crossbar.io, 2013 – 2015)

Obrázek 9 znázorňuje, jak probíhá zapsání aplikačních komponent k určitému tématu. Každé téma má vlastní identifikátor, který je reprezentován řetězcem. Tento řetězec má formát URI. Aby se předešlo kolizím v identifikátorech, je využita jmenná konvence z Javy, kde jsou URI zapisovány v obráceném pořadí. Aplikační komponenta, která se chce zapsat, zašle požadavek na Crossbar.io, ten udržuje list témat a uzly, které jsou k němu připojeny.

Obrázek 10 znázorňuje, jak probíhá zveřejnění nové informace v určitém tématu. Některá aplikační komponenta uveřejní informaci k určitému tématu. Crossbar.io



Obrázek 10: Zapsání se na téma (Crossbar.io, 2013 – 2015)

příjme tuto zprávu, podívá se do svého listu, které další komponenty jsou k tomuto tématu zapsány a těm informaci rozešle.

Pro instalaci Crossbar.io je zapotřebí nainstalovat Python 2 a následně nástroj pip, který slouží pro instalaci potřebných závislostí. Samotnou instalaci pak provedeme následujícím příkazem:

```
$ pip install crossbar
```

V dokumentaci Crossbar.io jsou napsány postupy pro různé operační systémy. Po provedení kompletní instalace lze využít některou z předpřipravených šablon k nahlédnutí do struktury aplikace pro různé programovací jazyky. Pro PHP je potřeba použít následující příkaz:

```
$ crossbar init --template hello:php --appdir hello
```

Po provedení tohoto příkazu jsou staženy některé soubory šablony do umístění, které jsme v příkazu použili. PHP šablona je závislá na PHP knihovně s názvem Thruway. Pro doinstalování všech potřebných závislostí se potřebujeme přesunout do složky s projektem a použít následující příkaz:

```
$ make install
```

Předpokladem pro úspěšné doinstalování je Composer, avšak ten už máme nainstalovaný.

Nyní můžeme Crossbar.io spustit, což se provede následovně:

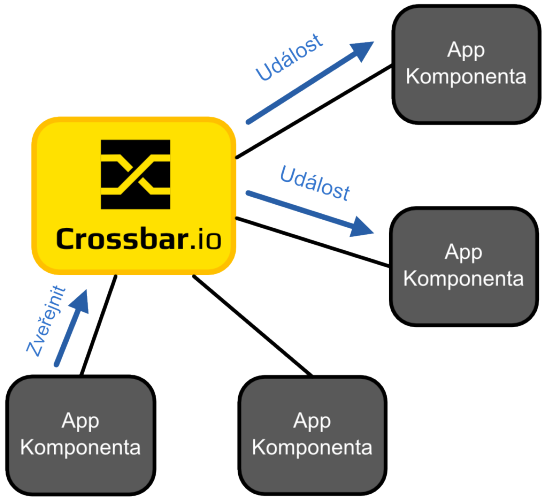
```
$ crossbar start
```

Po otevření prohlížeče a zadání příslušné adresy, na které Crossbar.io běží (najdeme v konfiguračním souboru), lze vidět, že příklad funguje (v konzoli uvidíme

výsledek procedury zavolané na straně PHP klienta). V konfiguračním souboru Crossbar.io můžeme mimo jiné nalézt následující dva řádky:

```
"executable": "php",
"arguments": ["../server.php"]
```

Tyto dva řádky určují, že implementace klienta, který je spuštěn společně s Crossbar.io, je v PHP a také ve kterém souboru můžeme kód naleznout. (Crossbar.io, 2013 – 2015)



Obrázek 11: Zveřejnění nové informace v tématu (Crossbar.io, 2013 – 2015)

4.4.1 Thruway

Thruway je implementace WAMP protokolu v PHP, která je na listu oficiálních a podporovaných projektů. Příklad PHP klienta vypadá následovně:

```
<?php
require __DIR__ . '/vendor/autoload.php';

use Thruway\ClientSession;
use Thruway\Peer\Client;
use Thruway\Transport\PawlTransportProvider;

$client = new Client("realm1");
$client->addTransportProvider(new PawlTransportProvider(
    "ws://127.0.0.1:9090/")
    );

$client->on('open', function (ClientSession $session) {
```

```
    // 1) subscribe to a~topic
    $onevent = function ($args) {
        echo "Event {$args[0]}\n";
    };
    $session->subscribe('com.myapp.hello', $onevent);

    // 2) publish an event
    $session->publish('com.myapp.hello',
        ['Hello, world from PHP!!!'], [],
        ["acknowledge" => true])->then(
        function () {
            echo "Publish Acknowledged!\n";
        },
        function ($error) {
            // publish failed
            echo "Publish Error {$error}\n";
        }
    );

    // 3) register a~procedure for remoting
    $add2 = function ($args) {
        return $args[0] + $args[1];
    };
    $session->register('com.myapp.add2', $add2);

    // 4) call a~remote procedure
    $session->call('com.myapp.add2', [2, 3])->then(
        function ($res) {
            echo "Result: {$res}\n";
        },
        function ($error) {
            echo "Call Error: {$error}\n";
        }
    );
});

$client->start();
```

V příkladu vidíme vytvoření nového PHP klienta, jeho připojení k již běžícímu routeru a následné použití všech 4 druhů možných nástrojů WAMP protokolu. Je zde provedeno zapsání na téma s názvem com.myapp.hello, dále je provedeno publikování informací v tomto tématu, je registrována procedura na sčítání dvou čísel a následně

je tato procedura zavolána s čísly 2 a 3. (Crossbar.io, 2013 – 2015)

4.5 Autobahn

Autobahn je jeden z projektů od společnosti Tavendo a poskytuje implementace různých WAMP klientů v různých programovacích jazycích. Práce se bude konkrétně zajímat o AutobahnJS. Jedná se o implementaci v JavaScriptu. Pro použití tohoto klienta je zapotřebí knihovnu nainstalovat, což provedeme následovně:

```
$ bower install autobahn
```

Po inicializaci a stažení komponenty skrz Bower lze začít s použitím. Příklad kódu pro JavaScript vypadá následovně:

```
var connection = new autobahn.Connection({
         url: 'ws://127.0.0.1:9090/',
         realm: 'realm1'
      });

connection.onopen = function (session) {

   // 1) subscribe to a~topic
   function onevent(args) {
      console.log("Event:", args[0]);
   }
   session.subscribe('com.myapp.hello', onevent);

   // 2) publish an event
   session.publish('com.myapp.hello', ['Hello, world!']);

   // 3) register a~procedure for remoting
   function add2(args) {
      return args[0] + args[1];
   }
   session.register('com.myapp.add2', add2);

   // 4) call a~remote procedure
   session.call('com.myapp.add2', [2, 3]).then(
      function (res) {
         console.log("Result:", res);
      }
   );
};
```

```
connection.open();
```

Jedná se o totožný kód jako v případě PHP klienta. Opět jsou použity všechny 4 nástroje WAMP protokolu. (Autobahn, 2014)

4.5.1 AngularWAMP

Aby bylo vše ještě o něco jednodušší, přichází na řadu AngularWAMP, který slouží jako obal pro AutobahnJS pro integraci do AngularJS aplikace.

Instalaci provedeme následovně:

```
$ bower install angular-wamp
```

Než můžeme WAMP protokol začít používat v Angularu, je potřeba využít již zmíněné Dependency Injection. Integraci tedy provedeme tímto způsobem:

```
var app = angular.module("myApp", ["vxWamp"]);
```

Ke konfiguraci připojení využijeme možnost předání objektu do funkce $wamp-Provider.init()$.

```
app.config(function ($wampProvider) {
     $wampProvider.init({
        url: 'ws://127.0.0.1:9090/',
        realm: 'realm1'
     });
 })
```

V tuto chvíli je již služba wamp dostupná v celé aplikaci a může být používána všemi ostatními komponentami. (Angular WAMP, 2015)

4.6 Bootstrap

Bootstrap je front-end framework, který byl vytvořen společností Twitter, aby ulehčil vývoj webových aplikací a stránek. Obsahuje v sobě CSS a HTML pro typografii, ikony, formuláře, tabulky, tlačítka, mřížky rozložení a navigace. Jedná se v současné době o nejpopulárnější framework, který ulehčuje vývoj responzivních aplikací. Pokud by se vývojář rozhodl pro vytvoření responzivní webové aplikace, potřebuje využít CSS, ve kterém je možné definovat tzv. Media Queries. Media Queries slouží k definování podmínek (rozsahů rozlišení), po jejichž splnění je aplikován určitý styl. Ze zmíněného vyplývá, že pro vytvoření všech variant webové stránky bude zapotřebí velké množství CSS.

Bootstrap umožňuje psát pouze jeden HTML a CSS kód a s použitím příslušných CSS stylů se postará o přizpůsobení pro různé druhy zařízení. Jedná se tedy o příjemnou pomoc, která značně zjednodušuje vývoj. (Bootstrap, 2015)

5 Popis hry

Hra se skládá z hracího pole, které je reprezentováno mapou. Mapa musí být rozdělena na určitý počet částí neboli území. Nadále je plně na správci hry, jaký druh mapy použije a na jaký počet částí bude rozdělena. Hra je prozatím určena pro dva hráče, později případně pro více hráčů. Jedna hra zabere přibližně 5–10 minut, podle toho jak rychle hráči odpovídají na otázky. Výsledky jednotlivých her jsou uchovávány a na jejich základě je možné vytvářet statistiky. Jednotlivé hry na sebe žádným způsobem nenavazují, všichni zúčastnění hráči tedy vždy začínají se stejnými podmínkami.

Kromě mapy se na obrazovce hry nachází oddíl, ve kterém jsou zobrazovány otázky a odpovědi či pouze otázka a políčko na dopsání odpovědi.

Dalším prvkem na obrazovce hry je sekce, ve které se zobrazují průběžné informace hráčům, tedy co se od nich v které chvíli očekává. Hráči získávají v jednotlivých kolech území na mapě. Vítězem je hráč, který během všech hracích kol nasbírá nejvíce území na mapě.

5.1 Průběh hry

Hra začíná ve chvíli, kdy se nějaký uživatel připojí na server a v nabídce zvolí vytvoření nové hry. První hráč je následně přesměrován na stránku, na které hra probíhá. Zde musí vyčkat, dokud se nepřipojí druhý hráč (později další hráči).

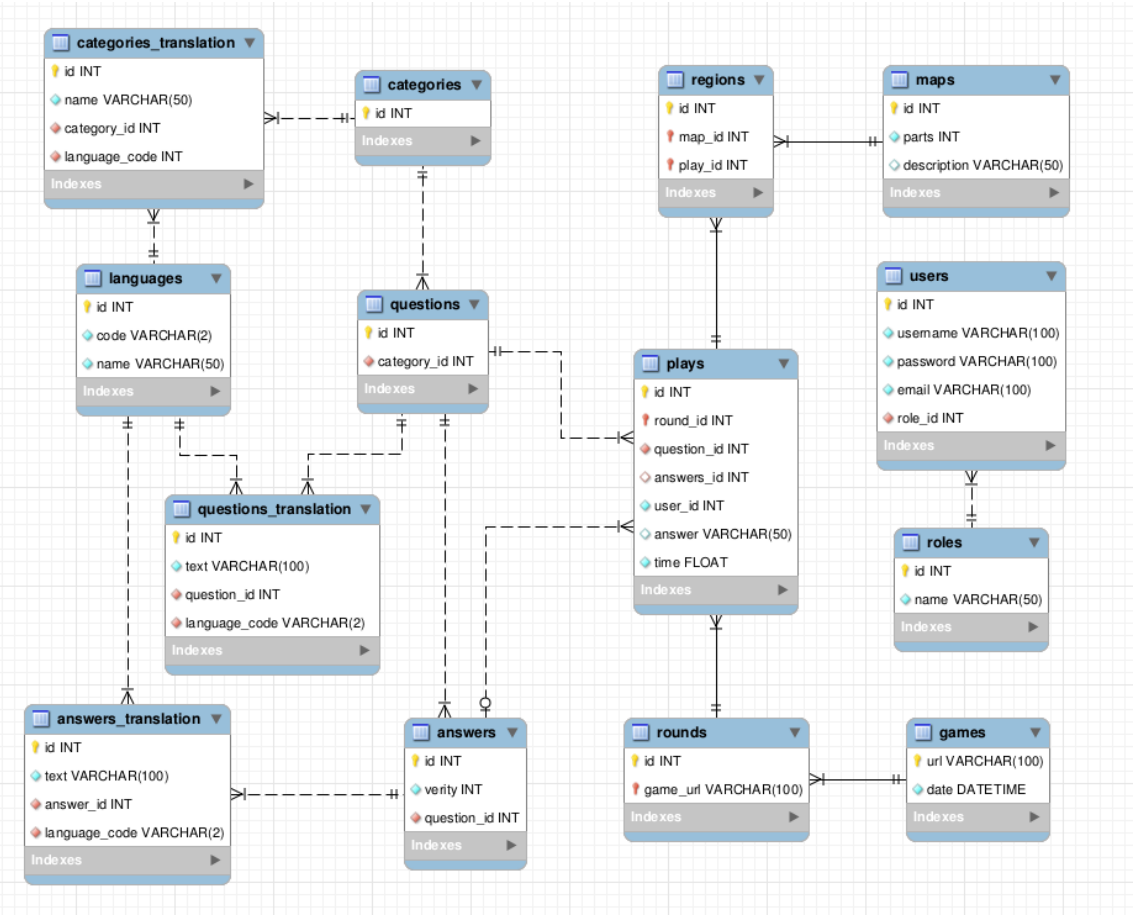
Jakmile jsou všichni hráči připojeni, hra začíná. Hráčům je zobrazena otázka, na kterou mohou tipovat odpověď (nejčastěji letopočet nějaké události) nebo vybrat jednu ze čtyř nabízených možností. Hráči postupně odpovídají, zaznamenává se doba, za kterou stihli odpovědět od položení otázky. Následně je vyhodnoceno, který uživatel byl správné odpovědi nejblíže či který ji uhodl správně. Měřítko správnosti je nejdůležitější. Při stejné odpovědi dvou uživatelů je vybrán ten, kterému zabrala odpověď méně času.

Hráč, který byl nejlepší, má možnost vybrat n+1 území na mapě, další hráč má možnost vybrat n území a případní další hráči následně n−1 území. N je konstanta, která může být nastavena trvale či později upřesněna v nastavení jednotlivé hry.

6 Tvorba hry

6.1 Návrh databáze

V databázi jsou ukládány všechny důležité informace, které s hrou souvisí. Databáze obsahuje celkem 14 tabulek a je implementována v MySQL. Schéma databáze je viditelné na obrázku 11. Při implementaci byl využit přístup pro uchování překladů ve více tabulkách. Schéma databáze je následující.



Obrázek 12: Schéma databáze

Tabulka **users** slouží k uchovávání informací o uživateli. Obsahuje sloupce k zaznamenání uživatelského jména, hesla a emailu. Tabulka **roles** slouží k rozdělení přístupových rolí, obsahuje sloupce s názvem role a vazbu do tabulky **users**. V aplikaci totiž potřebujeme později rozlišit, který uživatel má jaké pravomoci, z tohoto důvodu potřebujeme tyto informace uchovávat.

Následuje tabulka **languages**, která slouží k uchování všech jazyků, ke kterým je možné vytvořit překlad. Tato tabulka obsahuje dvoumístný kód jazyku a jeho název v angličtině. Následují tři tabulky, které jsou překládány. Jedná se o **categories**, **questions** a **answers**. Ke kategoriím se váže **categories translation**,

ve které uchováváme jméno kategorie a jazyk, ke kterému se překlad váže. K otázkám patří tabulka s názvem **questions translation**, ve které opět uchováváme text otázky a jazyk, ke kterému je překlad vázaný. V případě odpovědí se jedná o totožný případ, tabulka se jmenuje **answers translation** a je v ní uchován text odpovědi a jazykový kód.

Zbývající tabulky slouží k ukládání informací o probíhajících hrách. Tabulka **maps** uchovává informace o mapě, která je použita ke hře. Potřebujeme vědět, kolik políček má mapa k dispozici, tedy kolik obsahuje herních polí. V tabulce **map regions** jsou uloženy jednotlivá pole map, uchováváme číslo pole na mapě, a kterým hráčem je v kterém herním kole toto pole obsazeno. Tabulka **games** slouží k ukládání informací o hře samotné. Tedy na které adrese hra probíhá a kdy byla odehrána. Hru je možné rozdělit na menší části neboli herní kola. Informace o herních kolech jsou uloženy v tabulce **rounds**, kde můžeme najít, o které číslo kola se jedná a ke které hře se váže. Poslední tabulkou je pak **plays**, která spojuje dohromady celý koncept hry. Spojuje uživatele, který hru hraje, společně s herním kolem, které probíhá, ukládá, jaká otázka byla položena, jaká odpověď byla uživatelem zadána a které pole na mapě bylo uživatelem zabráno.

6.2 Návrh aplikace

Součástí návrhu aplikace je specifikace funkcí, které jsou v aplikaci dostupné. Na obrázku 12 lze vidět use case diagram, který tuto specifikaci funkcí upřesňuje. Lze vidět aktory, kteří k aplikaci mohou přistupovat. Jedná se o tři přístupové role, mezi které patří administrátor, člen, který se zaregistroval a host, který registraci neudělal.

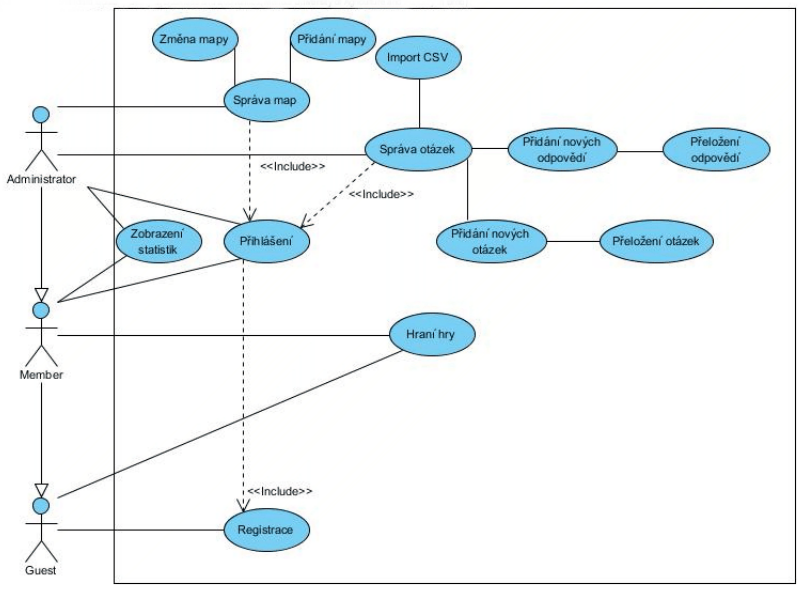
Host má možnost vyzkoušet hru a registrovat se. Registrovaný uživatel, tedy člen, má možnost také hrát hru a navíc zobrazovat statistiky, které se o jeho hrách ukládají. Administrátor má potom nejvíce možností, mezi které patří zobrazování statistik pro jednotlivé registrované členy, správa otázek, do které se řadí přidávání otázek, jejich překlad, přidávání odpovědí a jejich překlad či import otázek a odpovědi z CSV souboru. Dále správa map, na kterých jsou hry hrány, zde je možné provádět přidávání a změny map.

Na obrázku 13 je zachyceno, jak vypadají jednotlivé aplikační komponenty a jak spolu komunikují.

Obrázek 14 pak znázorňuje grafickou podobu průběhu hry za pomoci diagramu aktivit. Diagram aktivit vychází z popisu, který je uveden v kapitole 5.

6.3 Konfigurace serveru

Vývoj této diplomové práce nejprve probíhal na OS Windows, konkrétně Windows 8. Většina nástrojů, popsaných v předcházejících kapitolách lze nainstalovat na OS Windows, ale každá tato instalace zabere poměrně velké množství času a často se vyskytují potíže, protože tyto nástroje byly vyvíjeny na OS Linux.



Obrázek 13: Use case diagram aplikace

Základem pro start vývoje byla instalace XAMPP[9], po které následovala instalace Netbeans IDE ve verzi 8.0.2. Jedná se o velice příjemný nástroj, který ulehčuje vývoj ve všech směrech. Poskytuje našeptávání, zvýrazňovač syntaxe a má v sobě zabudovanou podporu pro HTML5, PHP i Nette Framework. Po doinstalování lze využívat i nástroje pro AngularJS.

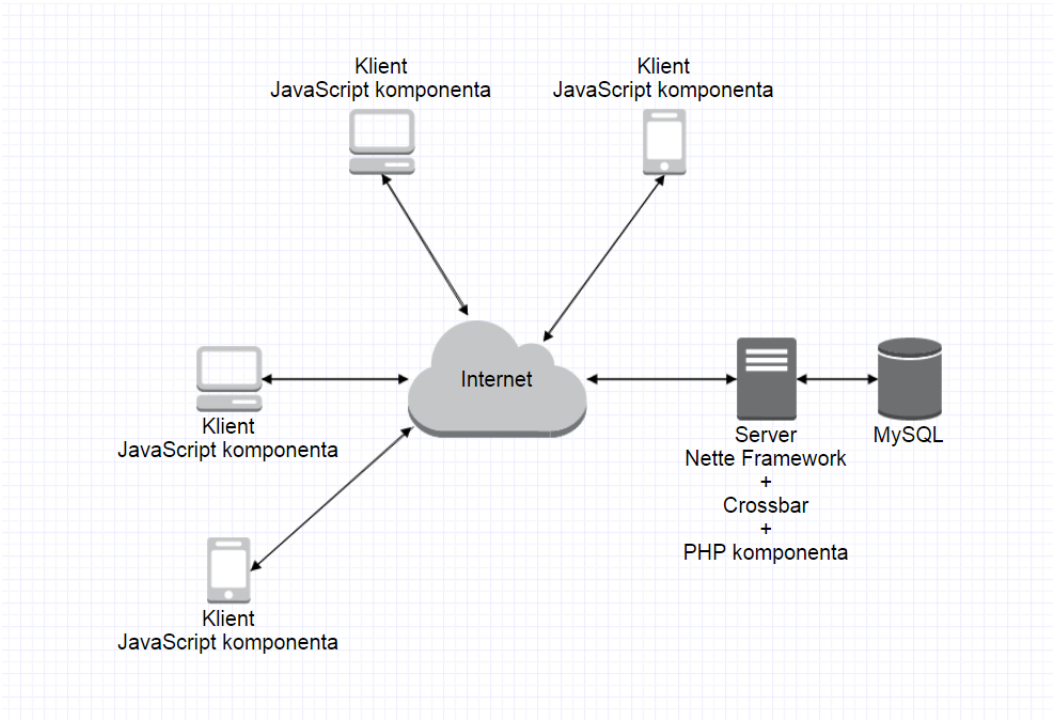
Hlavní problém nastal při zprovoznění Crossbar.io, který má některé další závislosti. OS Windows neobsahuje žádný program na správu závislostí pro aplikace, proto bylo nutné vše instalovat manuálně. Některé závislosti bylo obtížné vyhledat ve správné verzi, která byla zapotřebí (hlavně týkající se Pythonu).

Po této zkušeností přišel nápad vyzkoušet nainstalovat OS Linux a pokusit se o srovnání obou přístupů, o jejich výhody a nevýhody. Zvolenou distribucí se stalo Ubuntu ve verzi 14.04, které bylo vydáno v dubnu 2014 a jedná se o verzi LTS[10]. Instalace je skutečně jednoduchá. Po nainstalování OS následovala instalace LAMP[11] a Netbeans IDE, což poskytlo stejné vývojové prostředí jako na OS Windows. Hlavní příjemná změna přišla při instalaci všech nástrojů zmíněných v kapitole 4. K instalaci každého nástroje stačí jeden či více příkazů v terminálu a to je vše. Není zapotřebí žádné zdlouhavé hledání instalačních balíků na internetu a celý proces instalace se značně urychlí.

[9]Jedno z nejznámějších PHP vývojových prostředí (Apache, MySQL, PHP, Perl) (What is XAMPP, 2015)

[10]Zkratka pro Long Term Support, která znamená podporu na 5 let od vydání (Ubuntu, 2015,[online])

[11]Vývojové prostředí pro PHP (Linux, Apache, MySQL, PHP), alternativa pro XAMPP (LAMP, 2002 – 2015)



Obrázek 14: Aplikační komponenty

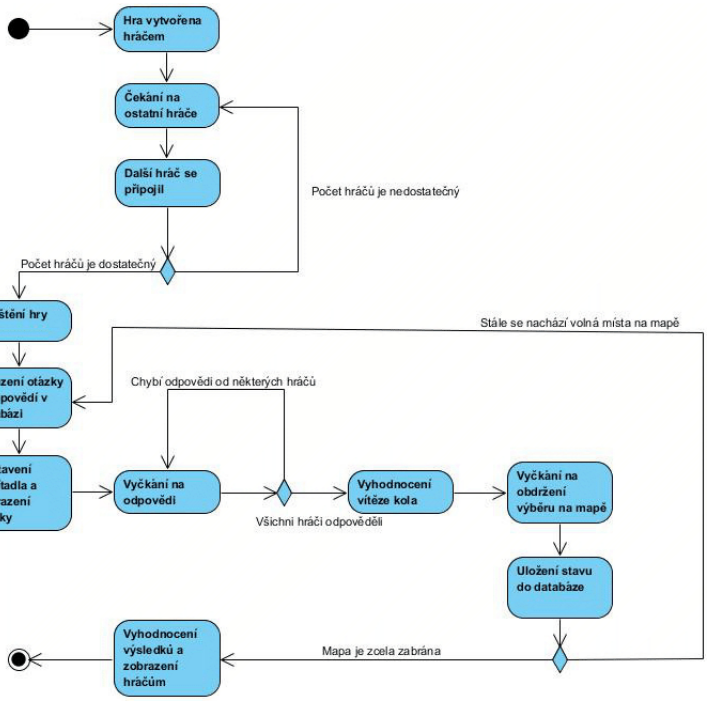
K otestování hry bylo následně zapotřebí nastavit server tak, aby byl přístupný veřejně na Internetu. Ke zpřístupnění došlo získáním veřejné IP od poskytovatele internetového připojení a dále nastavením směrovače. Ve směrovači bylo zapotřebí nasměrovat příchozích připojení na portu 80 na IP počítače, na kterém běží webový server a následně nasměrovat příchozí připojení na portu pro websockety (v tomto případě 8080) a to opět na IP počítače, na kterém běží Crossbar.io.

6.4 Implementace s využitím Nette Frameworku

Nette Framework slouží v této práci k vytvoření funkčnosti, která je popsána v use case diagramu. Nette aplikace je rozdělena na dva moduly. Jedná se o administrační modul a veřejný modul. Administrační modul poskytuje všechny funkce pro administrátora, tedy správa map a správa otázek. Veřejný modul potom slouží k vytváření nových her.

6.4.1 Překlad aplikace

K zakomponování multijazyčnosti do aplikace je využito doplňku pro Nette. Jedná se o integraci překladače ze Symfony. Multijazyčná aplikace udržuje ve všech presenterech veřejnou proměnnou, která v sobě nese informaci o tom, který jazyk si uživatel zvolil.



Obrázek 15: Diagram aktivit

```
/** @persistent */
public $locale;


/** @var \Kdyby\Translation\Translator @inject */
public $translator;
```

Tato proměnná je zakomponována do url adresy a tím se jazyk přenáší mezi jednotlivými stavy aplikace. Překlady jsou pak zaznamenávány ve strukturovaných souborech. Při spojení s multijazyčnou databází se jedná o řešení, které zajišťuje celkový překlad.

6.4.2 Model

Model je v Nette určen ke komunikaci s databází. Pro každou tabulku nebo skupinu tabulek spolu blízce souvisejících, je zapotřebí vytvořit třídu, která tyto tabulky obsluhuje. V obslužné třídě lze vytvářet metody pro práci s tabulkami a vracet potřebné výsledky.

Dobrým přístupem je vytvořit jednu abstraktní třídu, která obsahuje metody na vrácení celé tabulky, na vrácení záznamu dle ID, na vrácení záznamů dle předaných podmínek a na další případy, které se mohou opakovat u každé tabulky.

Každá obslužná třída pak implementuje abstraktní třídu a není zapotřebí opakovat kód.

```
class AbsTable extends Nette\Object
{
        /** @var Nette\Database\Context */
    public $database;

    public function __construct(Nette\Database\Context $database)
    {
        $this->database = $database;
    }
    public function getTable()
    {
        return $this->database->table(static::TABLE_NAME);
    }
}
```

6.4.3 ACL

Pro rozlišení jednotlivých přístupových rolí a jednodušší práci s právy a přístupy je vhodné použít předpřipravenou implementaci autorizátoru, kterou Nette disponuje. V autorizátoru je možné definovat jednotlivé role a vytvářet hierarchie. Následně stačí definovat, která role má přístup ke kterému presenteru či ke kterým funkcím.

6.4.4 Šablony

Důležitou součástí této práce je určité provázání funkčnosti obou částí aplikace. Tedy části, která je napsaná v Nette a části, která je napsaná v Angularu. Jelikož Nette slouží k vytváření nových her a zapisování do databáze, je zapotřebí nějakým způsobem předat tyto informace Angularu, aby s nimi mohl také pracovat. Jako nejlepší způsob se jeví zapsání JavaScript kódu do každé šablony, která je v Nette vygenerována jako nová hra.

```
<script>var player = { url: {$url}, locale: {$locale}, id: {$id}}</script>
```

Je zapotřebí přenášet informace o url, na které se uživatel nachází, dále jaký má zvolený jazyk a také jaké má id, z čehož lze rozlišit, jestli je přihlášen či nikoliv. Identifikátor url je unikátně generovaný řetězec, díky kterému je zajištěno rozlišování jednotlivých her. Tyto informace jsou pak předány v objektu s názvem player, ke kterému lze následně jednoduše přistupovat v Angularu.

6.5 Implementace PHP komponenty

Zatímco Nette Framework zajišťuje hlavní aplikační funkčnost, bylo zapotřebí vytvořit aplikační komponentu, která bude obstarávat správu her a bude komunikovat s aplikačními komponentami dalších klientů (hráčů).

Na začátku vývoje této práce vypadala knihovna Ratchet jako ideální řešení a implementace začala probíhat právě za pomoci této knihovny. Nicméně během vývoje práce došlo k vydání druhé verze WAMP protokolu. Vývojáři Ratchet knihovny se však z nedostatku času rozhodli pro nepokračování ve vývoji a pro nepřizpůsobení novým specifikacím. Ratchet knihovna fungovala na přímém spojení mezi klienty a serverem. Nebylo zapotřebí žádného směrovače. Díky tomuto rozhodnutí bylo zapotřebí vyhledat nové řešení.

Zvolena tak byla právě implementace směrovače Crossbar.io ve spojení s klientem v PHP, který je zprostředkován knihovnou Thruway (ta určitým způsobem navázala na knihovnu Ratchet a je rozšířena o potřebnou funkčnost WAMP protokolu druhé verze).

Z pohledu Crossbar.io směrovače není žádný rozdíl mezi jakýmkoliv klientem, který se k němu připojí. Z hlediska funkční logiky této aplikace se však implementace PHP klienta značně liší a slouží především ke spojování hráčů, kteří se připojují.

6.5.1 Server.php

Úkolem kódu v tomto PHP souboru je zajišťovat spojování hráčů, kteří se připojují z webových prohlížečů. V první částí kódu je specifikováno otevření spojení s WAMP směrovačem.

```
$connection = new Connection(
    [
        "realm" => 'realm1',
        "url"   => 'ws://127.0.0.1:8080/ws'
    ]
);
```

Následuje otevření spojení a přihlášení k dvěma tématům, které poskytuje sám Crossbar.io.

```
$session->subscribe('wamp.session.on_join', $onJoin);
$session->subscribe('wamp.session.on_leave', $onLeave);
```

Jedná se o témata, na která jsou zasílány informace ze směrovače v případě, kdy se někdo nový připojí či odpojí. Hlavně u odpojení se jedná o jediný způsob, jak tuto informaci získat.

Následuje registrace funkcí, které jsou využívány aplikačními komponentami uživatelů, které jsou spuštěny v prohlížeči. Druhým parametrem registrace je vždy proměnná s funkcí, která na následné zavolání reaguje.

```
$session->register('com.example.connectgame', $connectGame);
$session->register('com.example.ping', $ping);
$session->register('com.example.numberofplayers', $numberOfPlayers);
$session->register('com.example.game.map', $map);
```

```
$session->register('com.example.game.answer', $answer);
```

První funkce slouží k připojení ke hře, další dvě funkce slouží ke komunikaci se službami, které jsou vytvořeny v uživatelských aplikačních komponentách a poslední dvě funkce slouží k přijímání odpovědí od hráčů a přijímání informací o interakci s mapou.

```
$map = function ($args) use (&$games){
            if(array_key_exists($args[0]->url, $games)){
                $game = $games[$args[0]->url];
                return $game->map($args[1],$args[0]->session);
            }
       };
```

Pro udržování informací o probíhajících hrách bylo v této práci přistoupeno k vytvoření třídy, která funkční logiku poskytne. Každá hra je tak reprezentována objektem. Pro udržení více her slouží pole, které je indexováno za pomoci url identifikátoru. Jak je vidět ve funkci pro přijímání interakcí na mapě, nejprve je kontrolováno, jestli objekt se hrou existuje v poli a následně je zavolána funkce objektu s názvem map, které je předáno o jaký region mapy se jedná a od kterého hráče tato interakce přišla.

6.5.2 Třída hry

Třída hry v sobě nese všechny důležité informace, uchovává pole hráčů, uchovává pole reprezentující mapu, otázky, které jsou hráčům v jednotlivých kolech pokládány, odpovědí, které hráči v jednotlivých kolech zvolili, počet kol, relaci připojení k WAMP směrovači a další důležité parametry.

Kromě vlastní funkcionality přebírá třída i funkcionalitu vytvořenou skrz Nette a tak je umožněno přistupovat k funkcím modelu a pracovat jednoduše s databází. Do databáze tak lze jednoduše ukládat stavy hry v každém hracím kole.

Celá třída obsahuje necelých 300 řádků kódu. Mezi hlavní funkce patří přidávání hráčů a jejich kontrola, ukládání přijatých změn, které vznikají na mapě, rozesílání otázek, přijímání odpovědí a hlavně ukládání do databáze.

Důležitým aspektem třídy hry bylo vytvořit logiku aktivního hráče. Při odpovídání na otázku lze jednoduše kontrolovat, jestli máme odpovědi od všech hráčů. V tu chvíli lze postoupit k vybírání hracích regionů jednotlivými hráči. Avšak při vybírání regionů na mapě je zapotřebí určit, že interakce bude přijímána pouze od hráče, který je uvnitř třídy nastaven jako aktivní.

Funkce pro zaslání otázky a odpovědí vypadá následovně.

```
public function question($tips){
        $this->tips = $tips;
        $randomTip = array_rand($tips, 1);
        $randomTip = $tips[$randomTip]->id;
```

```
        $question = $this->questions_db->getTranslation($randomTip);
        $answers = $this->answers_db->getTranslations($randomTip);

        $this->answers[$this->round] = $answers;
        $this->question[$this->round] = $question;

        if(count($answers) == 4)
        {
            $rand =  rand(1 , 2);
        }
        elseif (count($answers)== 1)
        {
            $rand = 2;
        }

        $json = json_encode(['question' => $question,
                'answers' => $answers,
                'rand' => $rand]);
        $this->session->publish('com.example.game.'.$this->url.'.
                question', [$json]);
        $this->time = microtime(true);
    }
```

Funkce přebírá v parametru pole identifikátorů otázek, ze kterých může vybírat. Následně získává otázku a odpovědi z databáze. Podle počtu odpovědí je určeno číslo (nebo náhodně vybráno), které v JavaScriptové aplikační komponentě pomůže určit, jak otázku prezentovat. Všechny proměnné jsou následně převedeny do JSON formátu a publikovány příslušné hře. Jako poslední je provedeno zaznamenání času, které pomůže určit, jak dlouho trvala hráčům odpověď.

Ostatní funkce vypadají podobným způsobem, vytváří potřebnou logiku, získávají potřebné informace z databáze, převedou všechny proměnné do JSON formátu a zašlou informace správným aplikačním komponentám.

6.6 Implementace JavaScript komponenty

Každý připojený hráč je brán jako nová aplikační komponenta, která se připojuje k WAMP směrovači. Herní prostředí je implementováno do zobrazení, které vytvoří Nette Framework. Předpokladem pro zprovoznění Angular aplikace je nalinkování Angular knihovny a souborů, které obsahují aplikační kód.

V případě této práce je kód rozdělen do 4 souborů, což slouží k lepší přehlednosti při vývoji. Jedná se o soubory obsahující controllery, directivy, filtery a soubor s hlavními definicemi. V následujících podkapitolách si přiblížíme každý z nich.

6.6.1 Main.js

Hlavní soubor obsahuje vytvoření Angular modulu, ve kterém předáváme závislost ve formě Angular WAMP knihovny.

```
module = angular.module('dpApp',["vxWamp"]);
```

Následně je vytvořena a předána konfigurace, týkající se připojení k WAMP směrovači. Je zapotřebí předat url adresu běžícího WAMP směrovače a specifikovat ke které jeho doméně se připojit. V této práci je využito pouze jedné domény a url adresa může být nahrazena veřejnou IP adresou.

```
module.config(function ($wampProvider) {
     $wampProvider.init({
        url: 'ws://localhost:8080/ws',
        realm: 'realm1'
     });
 });
```

Dalším krokem je otevřít spojení s WAMP směrovačem.

```
module.run(function($wamp){
        $wamp.open();
});
```

Tyto tři krátké části kódu jsou to nejdůležitější v hlavním souboru. Nyní je připraveno a otevřeno spojení s WAMP směrovačem a je definován modul, na který budou navazovat další části kódu.

Hlavní soubor navíc obsahuje dvě továrničky, které vytvářejí služby. Jedná se o službu sloužící k zjištění odezvy od aplikační komponenty spravující hry a dále o službu zjišťující připojený počet hráčů k určité hře. Obě továrničky jsou velice podobné.

```
module.factory('PlayerService', function($timeout, $wamp) {
    function PlayerService($timeout, $wamp) {
        var self = this;

        self.onPlayerChanged = null;

        var testPlayers = function() {
            $wamp.call('com.example.numberofplayers', [player.url])
                    .then(function(res){
                        self.onPlayerChanged(res);
                        $timeout(testPlayers, 500);
                    });
                };
```

```
                testPlayers();
        }
        return new PlayerService($timeout, $wamp);
  });
```

Továrnička využívá službu timeout, která slouží k opakovanému volání nějaké funkce. Dále využívá službu wamp, která slouží k využití WAMP nástrojů. Vnitřní funkce továrničky volá funkci na zjištění počtu hráčů na aplikační komponentě, která udržuje informace o hrách a předává jako argument url adresu hry, kde se hráč nachází. Tato funkce se pak dotazuje každých 500ms.

Továrnička na zjištění odezvy pouze volá jinou funkci na aplikační komponentě, která udržuje informace o hrách a k tomu navíc počítá, jaký čas uběhne od vyslání dotazu po jeho návrat.

6.6.2 Controllers.js

V souboru s controllery je obsažena celá aplikační logika. Využito je dvou controllerů. ServiceController slouží k využití služeb, které jsme si nadefinovali v hlavním souboru. Ty předáváme jako závislosti a navíc předáváme službu scope, která slouží ke spojení s aplikačním modelem napříč controllery.

```
.controller('ServiceController',
    function($scope, PingService, PlayerService) {
      $scope.ping = null;
      PingService.onPingChanged = function(ping) {
          $scope.ping = ping;
      };
      PlayerService.onPlayerChanged = function(player) {
          $scope.player = player;
      };
});
```

Controlleru pouze předáváme vytvořené služby a následně je použijeme, přičemž předáme hodnotu nějaké proměnné, kterou pak zobrazíme uživateli.

Druhý controller se jmenuje MainController a využívá služeb scope, wamp a timeout. V tomto controlleru jsou všechny funkce týkající se přijímání nových otázek k zobrazení, odesílání odpovědí, které uživatel zvolil, udržování aktuálních informací o mapě a zabraných regionech.

```
angular.module('dpApp').controller('MainController',
  function($scope, $wamp, $timeout){
  }
```

Při nahrání stránky dojde ke spojení s WAMP směrovačem, což má za následek zavolání funkce, která reaguje na otevření spojení.

```
$scope.$on("$wamp.open", function (event, session) {
  player.session = session.session.id;
  $wamp.subscribe('com.example.game.'+player.url,
    onGameMessage);
  $wamp.subscribe('com.example.game.'+player.url+'.map',
    mapChange);
  $wamp.subscribe('com.example.game.'+player.url+'.question',
    questionMessage);
  $wamp.subscribe('com.example.game.'+player.url+'.result',
    resultMessage);
  $wamp.subscribe('com.example.game.'+player.url+'.'+player.session,
    sessionAlert);
  $wamp.call('com.example.connectgame', [player]).then();
```

Proměnná session je identifikátor relace, který jednoznačně určuje, o které připojení se jedná. Tento identifikátor je potřeba uchovat po celou dobu hry. Jednoznačně tak máme určeno, o kterého hráče se jedná, a můžeme s ním komunikovat. Následně se provede přihlášení k odběru z pěti různých témat a je zavolána funkce, která vykoná připojení ke hře (v aplikační komponentě, která udržuje informace o hrách). Tato témata slouží k přijímání různých druhů zpráv. Každé téma má url com.example.game, za kterým následuje určení url stránky, na které hra běží a následně určení, jestli se jedná o téma zprostředkovávající zprávy o mapě, otázkách či výsledcích.

Jako druhý parametr funkce wamp.subscribe je předávána proměnná, do které je přiřazena určitá funkce. Tato funkce je vykonána pokaždé, když je přijata nová zpráva z daného tématu. Téma mapy tedy zajistí synchronizaci přijatých dat s modelem Angular aplikace a zobrazí uživateli, jaký region vybral protihráč. Téma otázky zajistí zobrazení otázky uživateli. Při zobrazení nové otázky je nejprve spuštěno počítadlo, po jehož odpočítání je otázka zobrazena.

```
$scope.tipQuestionTimeout = function(){
if($scope.counter > 0){
  $scope.gameMessage = 'Zaciname za '+$scope.counter+'.';
  $scope.counter--;
  mytimeout = $timeout($scope.tipQuestionTimeout,1000);
} else {
  $scope.gameMessage = 'Odpovezte na otazku!';
  $scope.tipContainer = true;
  $scope.question = $scope.lastRound.question.text+'?';
}
}
```

Při zobrazování otázek mohou nastat dvě možnosti. V prvním případě se jedná o tipovací otázku, na kterou uživatel zapíše svůj číselný tip a je vyhodnoceno, který

uživatel měl svůj tip nejbližší. Druhou možností otázek jsou takové, na které uživatel dostane čtyři možnosti odpovědi a musí jednu vybrat. V druhém případě je implementováno náhodné promíchání odpovědí tak, aby správná odpověď nebyla vždy na stejném místě.

6.6.3 Directives.js

Soubor obsahující direktivy rozšiřuje v této práci jazyk HTML a obohacuje ho o zpracování herní mapy, která je v SVG formátu[12]. Obsaženy jsou konkrétně dvě direktivy, které na sebe navazují. SVG mapa v sobě obsahuje jednotlivé cesty a předtím než můžeme přistupovat k jednotlivým regionům mapy, je zapotřebí ji předzpracovat.

```
angular.module('dpApp').directive('svgMap',
  ['$compile', function($compile){
    return {
        restrict: 'A',
        templateUrl: '/dp/www/skola.svg',
        link: function (scope, element, attrs) {
            var regions = element[0].querySelectorAll('.state');
            scope.count = 0;
                angular.forEach(regions, function (path, key){
                scope.count++;
                var regionElement = angular.element(path);
                regionElement.attr('region','');
                regionElement.attr('map-data','mapData');
                regionElement.attr('id', scope.count);
                $compile(regionElement)(scope);
            });
        }
    };
}])
```

První direktiva má název svgMap, specifikujeme, že patří k modulu dpApp, který jsme specifikovali v hlavním souboru a předávám službu compile, která umožňuje udělat změny v mapě a poté ji zkompilovat.

Direktiva obsahuje cestu k mapě a obsahuje funkci, ve které vybíráme všechny cesty z mapy, které mají třídu state. Následně procházíme tyto cesty a vkládáme některé nové informace. Přidáváme očíslování jednotlivých regionů, abychom mohli rozlišit, o který region se jedná. Vkládáme propojení s mapData a vkládáme direktivu se jménem region. Odkaz na mapData slouží k propojení s modelem, ve kterém již máme nadefinované pole s hodnotami jednotlivých mapových regionů. Následně mapu překompilujeme.

[12]Formát pro bezztrátovou vektorovu grafiku (SVG, 1999 – 2015)

```
.directive('region', ['$compile', '$wamp', function($compile, $wamp){
    return {
        restrict: 'A',
        scope: {
            mapData: "="
        },
        link: function (scope, element, attrs) {
         scope.elementId = element.attr('id');
         scope.regionClick = function () {
           $wamp.call('com.example.game.map',
             [player, scope.elementId]).then(
                    function (res) {
                        if(res != null){
                        alert(res);
                    }
                });
           };
         element.attr("ng-click", "regionClick()");
         element.attr("ng-attr-fill",
           "{{mapData[elementId] | map_colour}}");
         element.removeAttr("region");
         $compile(element)(scope);
        }
    };
}]);
```

Direktiva region využívá opět službu compile a službu wamp. Definicí objektu scope s vlastností mapData spojujeme proměnnou z modelu a můžeme k této proměnné přistupovat právě v této direktivě.

Nyní jsme již schopni využít id, které jsme přidali v direktivě svgMap a přidáváme funkci, která komunikuje s aplikační komponentou, která udržuje informace o hrách. Tuto funkci následně definujeme jako parametr atributu ng-click, což má za následek, že při každém kliknutí na region v mapě bude tato funkce zavolána a navíc přidáváme atribut ng-attr-fill, který způsobí vykreslení barvy, kterou získáváme za pomoci filtru mapCcolour. Následně opět mapu kompilujeme a máme tak zaručeno, že všechny atributy budou na správném místě.

6.6.4 Filters.js

Soubor s filtry obsahuje filtr pojmenovaný mapColour a slouží k obarvení regionů mapy. Obsahuje v sobě funkci, která přebírá číslo, které je uloženo v poli s regiony mapy. Na základě čísla, které určuje počet hráčů a číslo jednotlivého hráče, je vrácena příslušná barva. Funkce je připravena na vracení více různých druhů barev a vypadá následovně.

```
angular.module('dpApp').filter('mapColour', [function () {
    function calculate(number){
        switch(number) {
            case 1:
                return "rgba(255, 0, 0, 1)";
                break;
            case 2:
                return "rgba(255, 255, 0, 1)";
                break;
            case 0:
                return "rgba(255, 255, 255, 1)";
                break;
            default:
                return "rgba(255, 255, 255, 1)";
        }
    }

    return function (input) {
        return calculate(input);
    };
}]);
```

7 Závěr

7.1 Diskuze

Tato práce předkládá možnost, jak přistupovat k vývoji webových aplikací. Je možné vidět, jak se důraz stále více přesouvá na uživatele a ten je stavěn do popředí. Zajímavou možností je přesunout vývoj klientské částí aplikace z prostředí PHP nástrojů do prostředí JavaScript nástrojů, ty totiž v dnešní době představují komplexní řešení, umožňující vytváření dynamických aplikací. Přímo se tedy nabízí zanechat serverovou část aplikace pouze pro komunikaci s databází či dalšími službami, vytvořit pouze aplikační rozhraní a vše ostatní nechat na starost JavaScriptu.

Vytvořená hra pro tuto chvíli splňuje všechny požadavky ze zadání a jasně ukazuje, jak je možné vyvíjet aplikace, kde spolu uživatelé mohou komunikovat v reálném čase. Hlavní výhodou použitého řešení je skutečnost, že je aplikovatelné napříč aplikačními vrstvami a také napříč různými programovacími jazyky. Každý vývojář tak může sáhnout po svém preferovaném řešení pro jakoukoliv aplikační část.

Testování aplikace ukázalo, že je funkční a nebylo zapotřebí dělat žádné zásadní změny. Jedinou drobností, kterou testování odhalilo, bylo různé zobrazování otázek hráčům. Stejná otázka se tak jednomu uživateli zobrazila jako tipovací s políčkem pro zapsání odpovědi a druhému jako otázka s nabídnutými odpověďmi. Tato funkcionalita tedy byla přesunuta z klientské aplikační komponenty do řídící aplikační komponenty.

Testování nadále ukázalo, že je zapotřebí ošetřit všechny neočekávané situace, které mohou nastat ve spojení mezi jednotlivými aplikačními komponentami. Pro tuto chvíli aplikace neřeší situaci, kdy neočekávaně vypadne spojení některému z hráčů, což je jistě situace, která může nastat. Před plným nasazením aplikace do produkčního prostředí je ještě zapotřebí tyto nedostatky ošetřit a zaměřit se na všechny další potenciální problémové situace, které mohou nastat. Tato neúplnost pro produkční prostředí je prozatím způsobena tím, že průběh práce byl náročný na studium všech nových a použitých principů. Většina těchto principů byla pro autora úplně nová a některé funkční nedostatky se postupně objevovaly až v průběhu implementace práce. Podněty k řešení se tak stále rozšiřovaly.

Všechny zjištěné nedostatky, které se objevily v průběhu práce, budou sloužit jako podklady pro další zlepšování a vývoj aplikace. Oblast aplikací komunikujících v reálném čase totiž nabízí mnoho možností pro budoucí použití.

7.2 Možnosti rozšíření

Jak již bylo zmíněno, hlavním směrem rozšíření aplikace může být logika ošetřující nenadálé situace. Mezi další směry vývoje pak lze zahrnout hratelnost pro více hráčů společně či přidání nových herních modifikací

Jako herní modifikace může být vnímána situace, kdy hráči obsadí všechny mapové regiony. V tuto chvíli hra po obsazení mapových regionů končí. Může však navazovat pokračování ve formě bojování o již obsazená pole.

8 Literatura

ADAPTIVE PATH. *Ajax: A New Approach to Web Applications*. [online]. 2005, [cit. 2015-05-10]. Dostupné z < *http* : *//www.adaptivepath.com/ideas/ajax − new − approach − web − applications/*>.

ANGULARJS. *Superheroic JavaScript MVW Framework*. [online]. 2010–2015, [cit. 2015-05-10]. Dostupné z < *https* : *//angularjs.org/*>.

ANGULAR WAMP. *An AngularJS wrapper for AutobahnJS*. [online]. 2015, [cit. 2015-05-10]. Dostupné z < *https* : *//github.com/voryx/angular − wamp*>.

AUTOBAHN. *Autobahn.ws* [online]. 2014, [cit. 2015-05-10]. Dostupné z < *http* : *//autobahn.ws/*>.

BACKBONE. *Backbone.js* [online]. 2015, [cit. 2015-05-10]. Dostupné z < *http* : *//backbonejs.org/*>.

BOOTSTRAP. *The world's most popular mobile-first and responsive front-end framework*. [online]. 2015, [cit. 2015-05-10]. Dostupné z < *http* : *//getbootstrap.com/*>.

BOWER. *A package manager for the web*. [online]. 2015, [cit. 2015-05-10]. Dostupné z < *http* : *//bower.io/*>.

BSD LICENCE. *BSD License Definition*. [online]. 2004, [cit. 2015-05-10]. Dostupné z < *http* : *//www.linfo.org/bsdlicense.html*>.

CAKEPHP V3.0. *The rapid development php framework*. [online]. 2015, [cit. 2015--05-10]. Dostupné z < *http* : *//cakephp.org/*>.

CODE GEEKZ. *20 Best PHP Frameworks for Developers in 2014*. [online]. 2014, [cit. 2015-05-10]. Dostupné z < *http* : *//codegeekz.com/20 − best − php − frameworks − developers − august* − 2014/>.

CODEPROJECT. *Multilanguage Database Design in MySQL*. [online]. 2014, [cit. 2015-05-10]. Dostupné z < *http* : *//www.codeproject.com/Articles/*752301/*Multi− language − Database − Design − in − MySQL*>.

COMPOSER. *Dependency Manager for PHP*. [online]. 2015, [cit. 2015-05-10]. Dostupné z < *https* : *//getcomposer.org/*>.

CROSSBAR.IO. *Unified Application Router*. [online]. 2015, [cit. 2015-05-10]. Dostupné z < *http* : *//crossbar.io/*>.

CSRF. *What is Cross-Site Request Forgery (CSRF)? - Definition from Techopedia*. [online]. 2010–2015, [cit. 2015-05-10]. Dostupné z < *http* : *//www.techopedia.com/definition/*172/*cross − site − request − forgery − csrf*>.

EMBER. *A framework for creating ambitious web applications.* [online]. 2015, [cit. 2015-05-10]. Dostupné z $< http://emberjs.com/ >$.

GNU GPL. *GNU General Public License.* [online]. 2014, [cit. 2015-05-10]. Dostupné z $< http://www.gnugpl.cz/ >$.

HTTP. *HyperText Transfer Protocol.* [online]. 2015, [cit. 2015-05-10]. Dostupné z $< http://www.webopedia.com/TERM/H/HTTP.html >$.

INTERVAL. *Co byste měli vědět o responzivním designu.* [online]. 2013, [cit. 2015-05--10]. Dostupné z $< https://www.interval.cz/clanky/co-byste-meli-vedet-o-responzivnim-designu/ >$.

JAVASCRIPT TUTORIALS AND SCRIPTS. *What Is JavaScript?* [online]. 2015, [cit. 2015-05-10]. Dostupné z $< http://javascript.about.com/od/reference/p/javascript.htm >$.

JSON. *Introducing JSON.* [online]. 2006, [cit. 2015-05-10]. Dostupné z $< http://www.json.org/ >$.

LAMP. *Building a LAMP Server.* [online]. 2002–2015, [cit. 2015-05-10]. Dostupné z $< http://lamphowto.com/ >$.

LARAVEL. *The PHP Framework For Web Artisans.* [online]. 2015, [cit. 2015-05-10]. Dostupné z $< http://laravel.com/ >$.

MIT LICENCE. *What is MIT License (X11 license or MIT X license)? – Definition from WhatIs.com* [online]. 2011, [cit. 2015-05-10]. Dostupné z $< http://whatis.techtarget.com/definition/MIT-License-X11-license-or-MIT-X-license >$.

MVC ARCHITECTURE. *Basic MVC Architecture.* [online]. 2015, [cit. 2015-05-10]. Dostupné z $< http://www.tutorialspoint.com/struts_2/basic_mvc_architecture.htm >$.

NETTE FRAMEWORK. *Rychlý a pohodlný vývoj webových aplikací v PHP.* [online]. 2008–2015, [cit. 2015-05-10]. Dostupné z $< http://nette.org/ >$.

NOTORM. *PHP library for simple working with data in the database.* [online]. 2010, [cit. 2015-05-10]. Dostupné z $< http://www.notorm.com/ >$.

NPM. *The package manager for Node.js* [online]. 2015, [cit. 2015-05-10]. Dostupné z $< https://www.npmjs.com/ >$.

PUBNUB. *What is HTTP Long Polling?* [online]. 2014, [cit. 2015-05-10]. Dostupné z $< http://www.pubnub.com/blog/http-long-polling/ >$.

RATCHET. *PHP WebSockets.* [online]. 2014, [cit. 2015-05-10]. Dostupné z $< http://socketo.me/ >$.

RFC 2616. *Hypertext Transfer Protocol*. [online]. 1999, [cit. 2015-05-10]. Dostupné z $< http://tools.ietf.org/html/rfc2616 >$.

RFC 6455. *The WebSocket Protocol*. [online]. 2011, [cit. 2015-05-10]. Dostupné z $< http://tools.ietf.org/html/rfc6455 >$.

ReadWrite: Web Apps, Web Technology Trends. *Angular, Ember, And Backbone: Which JavaScript Framework Is Right For You?* [online]. 2014, [cit. 2015--05-10]. Dostupné z $< http://readwrite.com/2014/02/06/angular-backbone-ember-best-javascript-framework-for-you >$.

SQL Injection. *What is SQL injection? A Webopedia Definition*. [online]. 2015, [cit. 2015-05-10]. Dostupné z $< http://www.webopedia.com/TERM/S/SQL-injection.html >$.

SVG. *SVG Tutorial*. [online]. 1999–2015, [cit. 2015-05-10]. Dostupné z $< http://www.w3schools.com/svg/ >$.

Symfony. *High Performance PHP Framework for Web Development*. [online]. 2015, [cit. 2015-05-10]. Dostupné z $< http://symfony.com/ >$.

Travis Maynard. *Getting Started with Gulp*. [online]. 2013, [cit. 2015-05-10]. Dostupné z $< https://travismaynard.com/writing/getting-started-with-gulp >$.

Ubuntu. *Co je Ubuntu?* [online]. 2015, [cit. 2015-05-10]. Dostupné z $< http://www.ubuntu.cz/ >$.

WAMP. *Web Application Messaging Protocol*. [online]. 2012–2014, [cit. 2015-05-10]. Dostupné z $< http://wamp.ws/ >$.

WAMP Libraries. *WAMP Implementations*. [online]. 2012–2014, [cit. 2015-05-10]. Dostupné z $< http://wamp.ws/implementations/ >$.

W3Schools. *JavaScript Tutorial*. [online]. 1999–2015, [cit. 2015-05-10]. Dostupné z $< http://www.w3schools.com/js/ >$.

W3Techs. *Usage statistics and market share of PHP for websites*. [online]. 2015, [cit. 2015-05-10]. Dostupné z $< http://w3techs.com/technologies/details/pl-php/all/all >$.

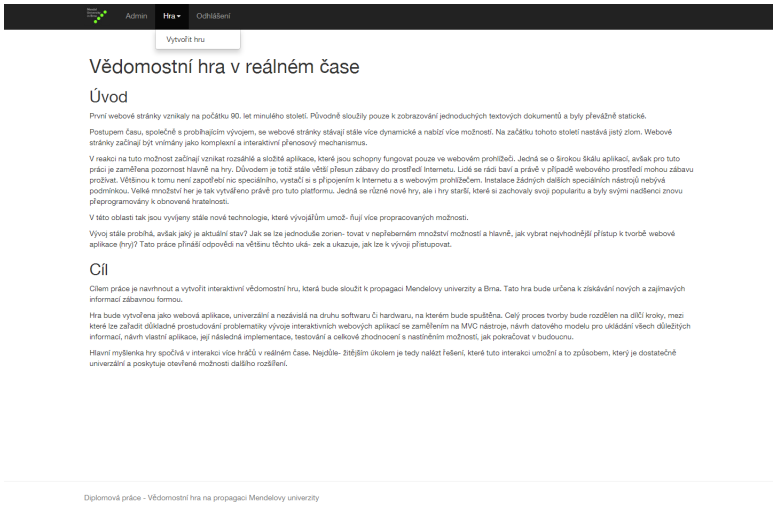
What is XAMPP. *XAMPP Installers and Downloads for Apache Friends*. [online]. 2015, [cit. 2015-05-10]. Dostupné z $< https://www.apachefriends.org/index.html >$.

XSS. *What is XSS? A Webopedia Definition*. [online]. 2015, [cit. 2015-05-10]. Dostupné z $< http://www.webopedia.com/TERM/X/XSS.html >$.

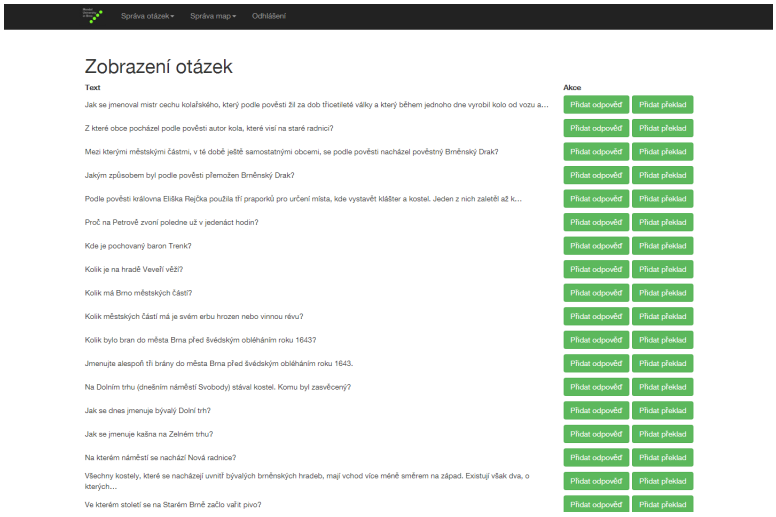
Zend Framework. *The most popular framework for modern, high-performing PHP applications*. [online]. 2006–2015, [cit. 2015-05-10]. Dostupné z $< http://framework.zend.com/ >$.

Přílohy

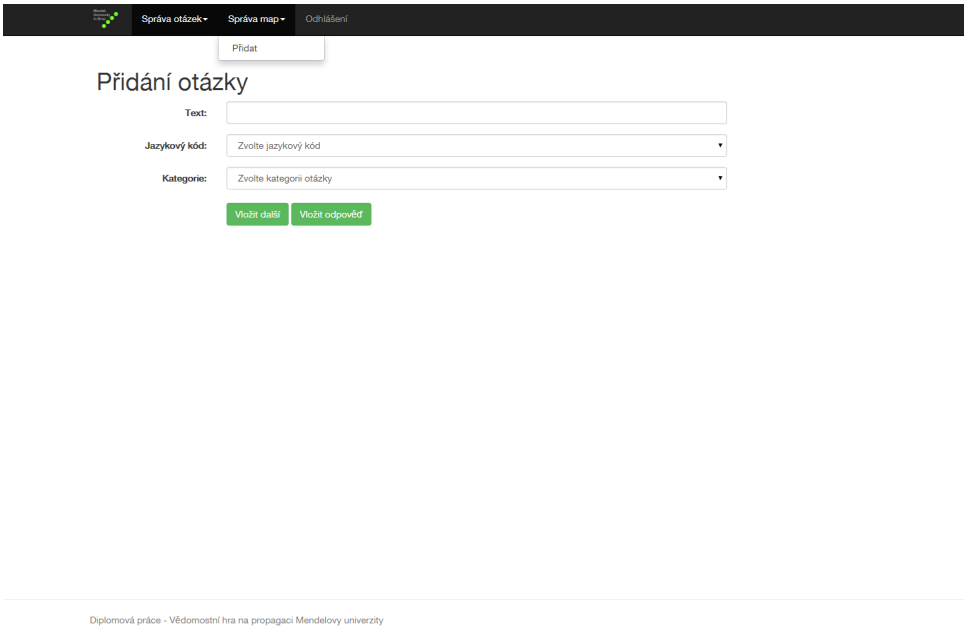
A Příklady zobrazení webové aplikace na PC



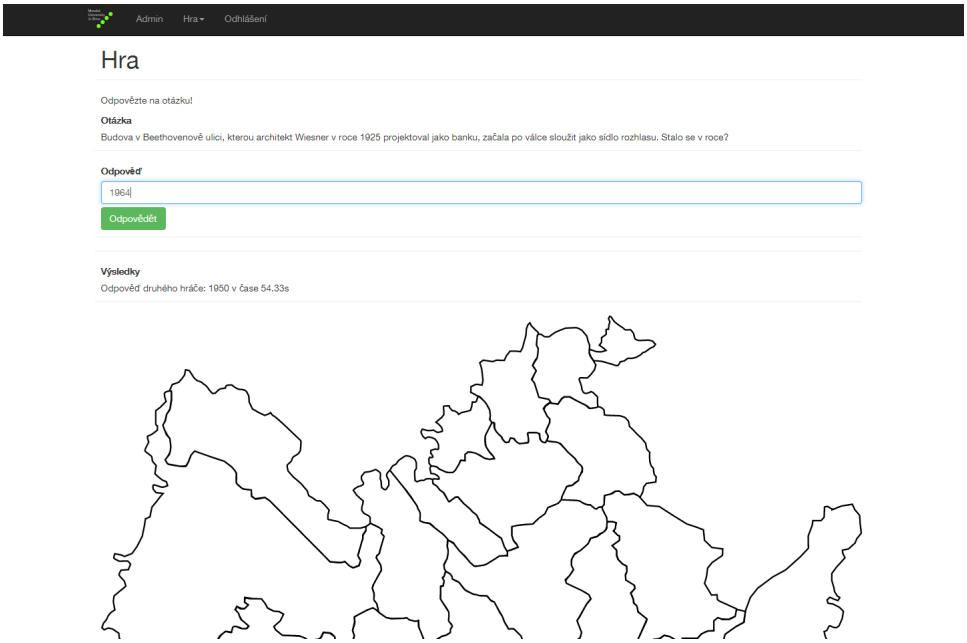
Obrázek 16: Hlavní stránka



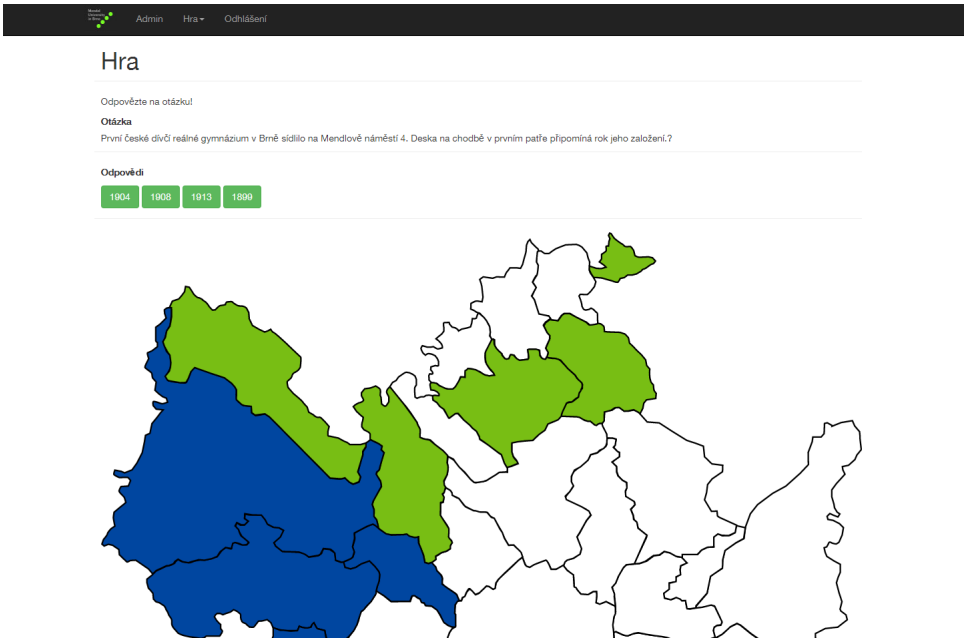
Obrázek 17: Přehled otázek a jejich správa



Obrázek 18: Administrační prostředí s přidáním otázky



Obrázek 19: Začátek hry s tipovací otázkou

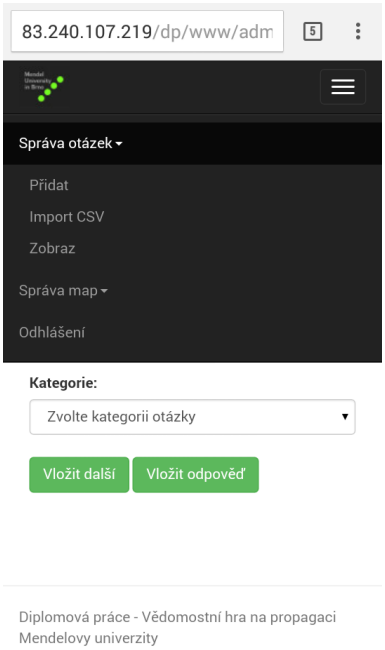


Obrázek 20: Rozehraná hra s položenou otázkou

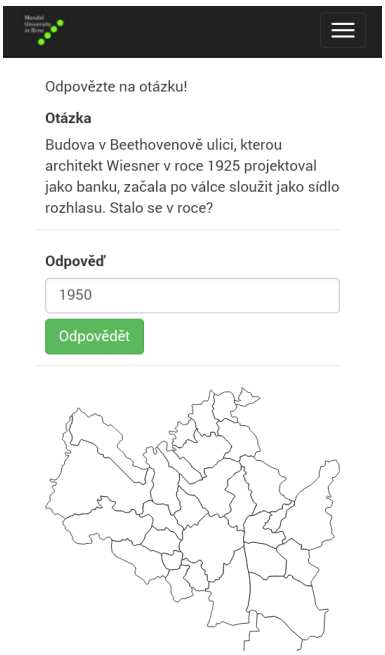
B Příklady zobrazení webové aplikace na mobilu

Obrázek 21: Ukončená hra

Obrázek 22: Registrace uživatele



Obrázek 23: Administrační prostředí s přidáním otázky



Obrázek 24: Začátek hry s tipovací otázkou

Obrázek 25: Rozehraná hra s položenou otázkou