Mendelova univerzita v Brně
Provozně ekonomická fakulta

Webová aplikace pro inzerci a správu hudebních zkušeben

Bakalářská práce

Vedoucí práce:
Ing. Oldřich Faldík

Jan Čižmár

Brno 2017

Děkuji svému vedoucímu bakalářské práce Ing. Oldřichu Faldíkovi za jeho cenné odborné rady, vedení a ochotnou komunikaci v průběhu celého řešení. Děkuji také Markétě Chalupníkové za překlad aplikace do francouzského jazyka a Libuši Čižmárové za jazykovou korekturu práce. Také bych rád poděkoval všem vývojářům, kteří zdarma poskytli komponenty pro tvorbu této práce.

Čestné prohlášení

Prohlašuji, že jsem tuto práci: **Webová aplikace pro inzerci a správu hudebních zkušeben**
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 21. května 2017 .....

Abstract

Čižmár, J. Web application for advertisement and maintenance of musical rehearsal rooms. Brno, 2017

This work aims to design a web application, enabling users to efficiently maintain and advertise rehearsal rooms for musical ensambles. The application is based on javascript framework Angular 4 and PHP framework Larvel. The interface developed is a browsing engine supporting various forms of search filtering, further allowing for map-based location pinpointing and submitting new items. For the musical community world-wide, the application thus provides a simple route to securing and advertising of rehearsal studio premises.

Key words
Web application, Angular 4, Laravel, Bootstrap, PHP, MySQL, HTML5, CSS3

Abstrakt

Čižmár, J. Webová aplikace pro inzerci a správu hudebních zkušeben. Brno, 2017.

Cílem práce bylo vytvořit webovou aplikaci, která uživatelům umožní efektivně spravovat a inzerovat hudební zkušebny. Pro implementaci aplikace byl použit zejména jasvascriptový framework Angular 4 a PHP framework Laravel. Vytvořené řešení umožňuje uživatelům vkládat a zobrazovat hudební zkušebny včetně filtrování nebo zobrazení na mapě. Díky této aplikaci mohou uživatelé z celého světa snáze nacházet a inzerovat prostory k hudebním zkouškám.

Klíčová slova
Webová aplikace, Angular 4, Laravel, Bootstrap, PHP, MySQL, HTML5, CSS3

Obsah

1 Úvod a cíl práce

1.1 Úvod

V době, kdy jsou hudební nástroje dostupné a internet je přeplněný diskusními fóry, videy a články o tom, jak hrát na hudební nástroj, mnoho lidí nachází ve hře na hudební nástroj uplatnění svého volného času. Velmi často se tito lidé spojují do kapel. Kapely se ale mnohdy setkávají s problémy v tom, kde se budou scházet a zkoušet na případné koncerty. Většina moderních kapel má ve své sestavě bicí nástroje, které jsou velmi hlasité, a na tuto úroveň hlasitosti se pak musí přiblížit i hlasitost ostatních nástrojů, čímž vzniká hluk, který nelze produkovat například v pokoji panelového domu či v řadovém domku. S tímto problémem se potýkají zejména kapely ve větších městech, kde je hustota domů tak velká, že se v drtivé většině nenajde žádný člen kapely, v jehož bytě by se dala hudba produkovat.

Na scénu tak přichází sdílené a nesdílené hudební zkušebny. Často se stává, že člen kapely disponuje prostorem dostatečně odlehlým nebo odhlučněným, ve kterém se dá produkovat hlasitá hudba, ale protože na jednu kapelu je provoz zkušebny příliš nákladný, rozhodne se zkušebnu pronajímat dalším kapelám. S tím ale přichází problém, jak dát ostatním kapelám ve městě vědět, že je jeho zkušebna k dispozici.

Za tímto účelem jsem před pěti lety zprovoznil webový portál pro inzerci hudebních zkušeben, který je však zastaralý, a proto stále méně oblíbený. Mimo tento systém se v současné době dají využívat ještě některá webová fóra, inzertní servery, případně sociální sítě.

1.2 Cíl práce

Původní systém má mnoho bezpečnostních děr, svojí funkcionalitou nepřevyšuje možnosti inzertních serverů a chybí mu mnoho funkcí, které postrádají jak inzerenti zkušeben, tak kapely, které zkušebnu hledají. Jde například o možnost zobrazení zkušeben v mapě, filtrování zkušeben na základě různých parametrů, vkládání rozvrhů či evidence vybavení. Cílem práce je tedy navrhnout a implementovat aplikaci, která umožní uživatelům uspokojivě nacházet, inzerovat a spravovat hudební zkušebny v moderním, rychlém a přehledném prostředí. Nejprve však bude nutné analyzovat současné možnosti inzerce a správy hudebních zkušeben a až poté novou aplikaci navrhnout a implementovat.

2 Současný stav

V současné době je portál jednoduchým inzertním serverem, který se od ostatních liší jen tím, že se zaměřuje pouze na hudební zkušebny, což může návštěvníky lákat díky tomu, že při zadaní hesla „hudební zkušebny" do vyhledávače se objevuje na prvních pozicích. Současný vzhled portálu je na obrázku 1. Systém neumožňuje nic jiného než přidat inzerát s těmito parametry:

- Název
- Text
- Cena
- Kraj
- Obec
- Telefon
- E-mail

K inzerátu se navíc dají připojit obrázky. Po zadání těchto parametrů zadá inzerent heslo, pomocí kterého potom může inzerát smazat nebo editovat.

Návštěvníci webu mohou filtrovat inzeráty na základě parametrů:

- Kraj
- Obec
- Cena

Po kliknutí na konkrétní inzerát se objeví detail (obrázek 2), v rámci kterého se dá zkušebna sdílet na některých sociálních sítích a na inzerát odpovědět.

To je v podstatě vše, co portál může uživatelům nabídnout. Je tedy jasné, že aby byl konkurenceschopný, je třeba ho rozšířit o mnoho možností.

2.1 Využité prostředky

Momentálně systém využívá pouze programovací jazyk PHP a zdarma staženou šablonu, která byla následně upravována. Web využívá pouze zastaralé CSS styly. Protože systém neimplementuje žádný framework a je napsaný v zastaralém PHP, je jednoduší ho začít vytvářet znovu a nevycházet z původního systému ani ve smyslu využívání částí kódu.

2.2 Funkční požadavky na nový systém

Nová webová aplikace by měla vyhovovat následujícím požadavkům:

- Vytváření profilů zkušeben

Obrázek 1: Vzhled původního systému

Obrázek 2: Současný vzhled detailu

- Evidence vybavení zkušeben
- Zobrazení zkušeben v mapě
- Evidence rozvrhů zkušeben
- Možnost filtrovat zkušebny na základě údajů o nich (cena, poloha, vybavení atd.)

2.3 Nefunkční požadavky

Aplikace musí vyhovovat těmto nefunkčním požadavkům:

- Možnost vytváření uživatelských účtů
- Mnohojazyčnost celé aplikace
- Zabezpečení systému proti různým útokům
- Responzivita aplikace
- Zajištění rychlé práce s webovou aplikací bez zbytečných časových prodlev
- Implementování aplikace, aby byla uživatelsky přívětivá

2.4 Existující servery umožňující inzerovat hudební zkušebnu

Jak už jsem uváděl, žádný systém na českém ani celosvětovém trhu není tak obsáhlý jako ten, který v rámci práce budu vytvářet. Většina z nich nedisponuje žádnou rozšiřující funkcí speciálně pro hudební zkušebny, a kvůli tomu se stávají nedostačujícími. Mezi české servery, na nichž se inzeruje nejvíce zkušeben, patří zejména tyto portály:

- http://www.musicstage.cz/hudebni-zkusebny/
- http://hudebniforum.cz/zkusebny/
- http://hudebnibazar.cz/zkusebny/240000/
- http://kultura.hyperinzerce.cz/zkusebny/
- http://www.inzer.cz/hudba/zkusebny/

musicstage.cz (Atlantida s.r.o., 2014)

Všechny zmíněné portály až na musicstage.cz jsou pouze inzertními servery, kam inzerent může vložit svoje kontaktní údaje, nadpis, popis, cenu a fotografie. Takový inzerát postupně „zapadá prachem" ostatních inzerátů a provozovatelé zkušeben tak inzeráty musí stále obnovovat. Jediný portál, který se nějakým způsobem od ostatních odlišuje, je musicstage.cz. Tento portál má vyhledávací filtry zaměřené

přímo na hudební zkušebny a eviduje o nich podrobnější informace. Velký problém je ale v tom, že je uživatelsky velmi nepřívětivý. Asi proto je u něj registrováno pouhých 10 zkušeben. Abych svoji zkušebnu vůbec zaregistroval správně, musel jsem upravit HTML kód formulářů stránky.

Lze tedy říct, že tento portál je zveřejněn nedokončený, a vzhledem k tomu, že je v tomto stavu už několik let, nikdo jej pravděpodobně ani nedokončí.

rehearsalspacefinder.com (Rehearsal Space Finder, 2017)

Tento britský server pro inzerci všemožných zkušeben jistě stojí za zmínku. Obsahuje filtry zaměřené na hudební zkušebny a funguje správně. U zkušeben lze nastavit cenu, velikost, kapacitu osob a základní vybavení. Bohužel není mnohojazyčný a všechny ceny jsou uváděny v britských librách, nelze ho tedy aplikovat na český trh. Systém také nijak nepracuje s rozvrhy.

2.5 Shrnutí stávajících portálů

Žádný ze zmíněných portálů nedisponuje takovými vlastnostmi, aby naplnil všechny požadavky návštěvníků. Proto si portál vytvořený v rámci této práce zajisté najde své inzerenty a návštěvníky.

Vzhledem k tomu, že se na českém trhu nenachází žádný portál poskytující funkce blízké projektu, který v rámci své bakalářské práce vytvořím, je velmi pravděpodobné, že práce na tomto projektu nebude vynaložena nadarmo a že se systém shledá s oblibou mnoha inzerentů hudebních zkušeben a skupin hledajících hudební zkušebnu.

3 Návrh aplikace

Na základě výše zmíněné analýzy je zřejmé, že žádný v současnosti využívaný systém nevyhovuje uvedeným požadavkům. Proto je nutné navrhnout celou aplikaci na míru, aby umožňovala pracovat s následujícími komponentami.

Uživatelské účty

V původním systému žádné uživatelské účty neexistovaly. Pro každou zkušebnu si inzerent musel zvolit heslo a pomocí tohoto hesla později inzerát upravovat nebo mazat. To je problém v případě, kdy má inzerent několik zkušeben a pro každou si musí pamatovat heslo. Změnu kontaktních údajů pak musí pro každou zkušebnu měnit zvlášť. To jsou problémy, které vyřeší existence těchto uživatelských účtů. Po přihlášení bude mít uživatel možnost zobrazit přehledný seznam svých zkušeben. Uživatel tak o nich bude mít kompletní přehled a bude tak moci zkušebny snáz spravovat.

Profily zkušeben

V původním systému byla zkušebna reprezentována jednostránkovým pohledem, který obsahoval jen velmi málo informací. V novém systému bude systém disponovat mnohem širší škálou informaci, nad kterou bude možné následně filtrovat. Častým problémem je také to, že zkušebny v čase zanikají, ale portály je stále evidují. Proto budou inzerentům chodit e-maily požadující prodloužení platnosti zkušebny. Pokud uživatel zkušebně platnost neprodlouží, bude nejprve zneviditelněna a následně smazána.

Evidence vybavení

Členové začínajících kapel většinou nemají dostatek financí na to, aby si sami zakoupili všechno potřebné vybavení pro dostatečně hlasitou reprodukci svého hudebního nástroje, hráči na bicí nástroje zase nemohou každý týden přenášet celou svou bicí soupravu do zkušebny a po zkoušce zpět domů. Většina sdílených zkušeben má proto nějaké základní vybavení, které mohou používat všechny platící kapely. Pokud tedy návštěvník potřebuje zobrazit například pouze zkušebny, které disponují výbavou pro reprodukci zpěvu, bude mu to umožněno díky evidenci vybavení. Těžko by se vyvíjel algoritmus, který by tyto informace doloval z textového popisu zkušebny. Evidence vybavení tak přináší návštěvníkům další možnost, jak porovnávat jednotlivé zkušebny mezi sebou.

Mapa zkušeben

Nesmírně důležitou součástí procesu výběru zkušebny je zohlednění její polohy. Členové kapely si vždy rozmyslí, jestli pojedou do zkušebny 10 minut nebo hodinu.

Proto bude dobré zobrazovat zkušebny na mapě. Žádný návštěvník nepotřebuje zobrazit zkušebny na celém světě. Díky tomu je možné se nejprve dotázat na obec a následně zobrazit zkušebny v okruhu této obce. Takto zobrazené zkušebny se pak zobrazí i na mapě a uživatel bude moci snadno zkušebny porovnávat dle polohy.

Rozvrh zkušebny

Pokud je zkušebna sdílená, zkouší v ní více kapel. Pokud jsou kapely dvě, snadno se domluví, kdy bude která kapela zkoušet. Problém ale nastává, pokud je kapel více. V takovém případě správce ztrácí přehled a musí začít psát rozvrh. Většinou k tomu používají papír a tužku, pak ale rozvrhy nejsou veřejné a kapely zbytečně kontaktují správce zkušebny ohledně termínů, které již nejsou volné. Rozvrhy zkušeben tedy ušetří čas jak kapelám, tak správcům zkušeben.

Mnohojazyčnost

Vzhledem k tomu, že projektů, jako je tento, na internetu není mnoho a kapel, které hledají zkušebnu, je mezinárodně velmi mnoho, není důvod, proč portál nerozšířit na celosvětovou úroveň. Systém tedy bude na internetu v mnohojazyčné formě a uživatelé z jiných států světa tak třeba ani nepoznají, že se nejedná o portál určený jen pro jejich trh. Největším problémem bude nejspíš získat kvalitní překlady všech položek na webu. Systém bude připraven pojmout všechny světové jazyky, avšak zatím bude pouze český, anglický a francouzský.

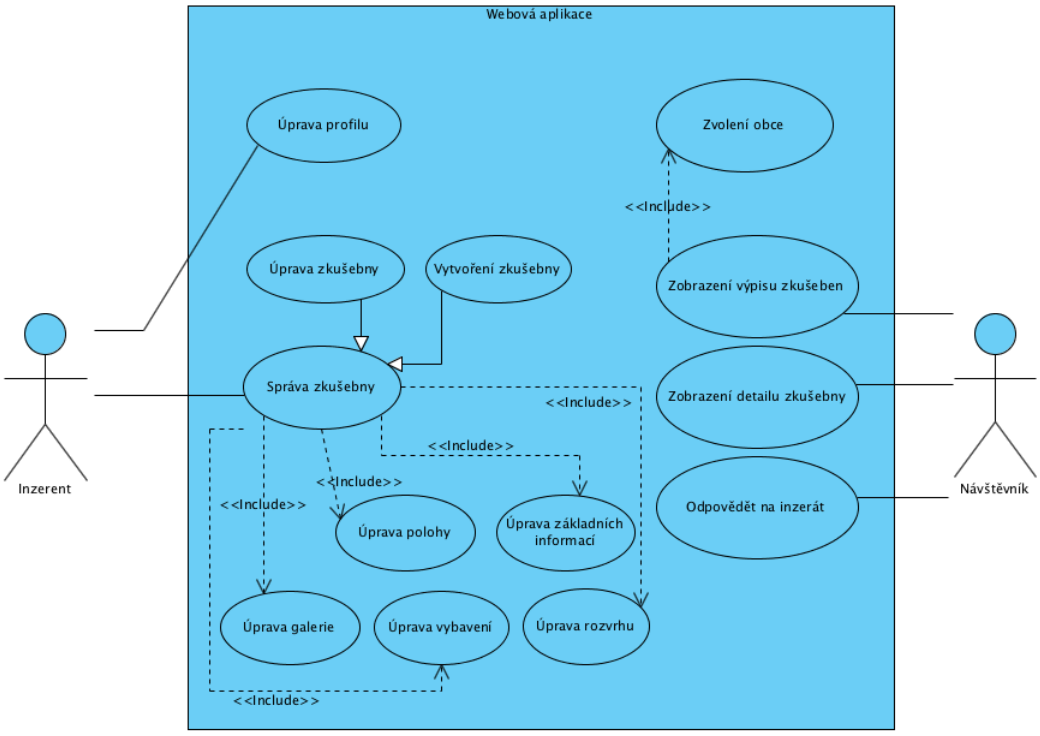
3.1 Diagram případu užití

Aby bylo zřejmé, jak budou uživatelé s aplikací pracovat, je potřeba vytvořit Use-Case diagram neboli diagram případu užití, který je na obrázku 3. V diagramu je vidět, že inzerentovi je umožněno upravovat svůj uživatelský profil (změna hesla, e-mailu a jména), vytvářet a upravovat informace o zkušebně. Tvorba a následná úprava informací je rozdělena do podsekcí odpovídajících příslušnému úkonu, protože evidovaných informací je poměrně mnoho.

3.2 Návrh databázového schématu

Základem aplikace, která udržuje informace v databázi, je návrh databázového schématu. Pro jeho znázornění jsem použil ERD diagram. Ten je znázorněn na obrázku 4.

Základní entitou aplikace je *User* (Uživatel), který může v systému vytvářet entity *Rehearsal room* (Hudební zkušebny). V rámci těchto entit budou evidovány všechny základní informace o hudební zkušebně, jako je název, cena, poloha atd. Mohou být vytvářeny také entity *Image* (Obrázky), v nichž bude uložena informace o jméně obrázku. Můžete si všimnout, že podle návrhu nemusí být obrázek přiřazen zkušebně (*Rehearsal room*). To proto, že při procesu vytváření profilu zkušebny

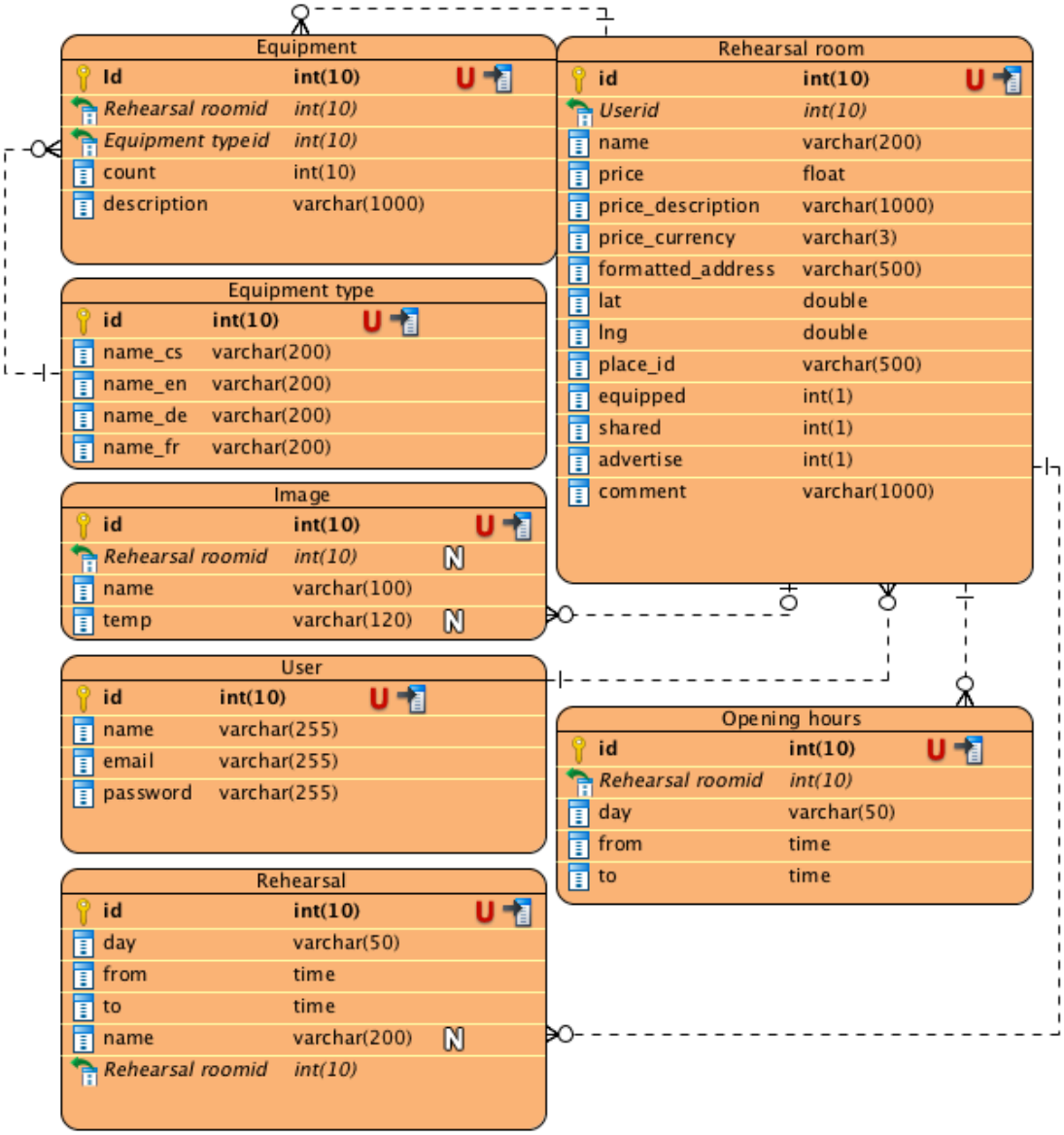


Obrázek 3: Use case diagram webové aplikace

budou obrázky ukládány ještě před tím, než se vytvoří samotný profil, aby uživatel nemusel čekat na upload obrázku a jeho ověření. Vazba tak bude vytvořena až při odeslání požadavku na vytvoření profilu zkušebny kvůli dočasnému označení obrázku v hodnotě temp.

Dále bude v aplikaci evidováno vybavení (entita Equipment). V rámci této entity bude evidován typ vybavení (Equipment type), počet kusů a popis (např. značka reproduktoru). V samotné entitě typu vybavení (Equipment type) budou ukládány pouze názvy typu v různých jazycích.

Entita Opening hours (Otevírací doba zkušebny) potom obsahuje informace o začátku a konci otevírací doby v daném dni v týdnu, podobně entita Rehearsal obsahuje den v týdnu, začátek a konec zkoušek ve zkušebně. Zde je možné namítnout, že tímto řešením nebude možné vkládat zkoušky, které se budou konat například každý druhý týden či jednorázově. Ovšem v opačném případě by byl problém s filtrováním takovýchto dat koncovým návštěvníkem webu a je velmi pravděpodobné, že zesložitění funkcionality rozvrhů by vedlo k následnému ignorování této komponenty všemi koncovými uživateli. Je zkrátka důležité zachovat aplikaci jednoduchou natolik, aby byli uživatelé ochotni informace poskytovat či využívat jejich evidence.



Obrázek 4: ERD diagram databázového schématu

4 Implementace aplikace

4.1 Výběr prostředků

Od počátku internetu po současnost se prostředky, kterými bylo možno vytvářet webové stránky, podstatně rozrostly. Ještě před několika lety bylo jedinou metodou, jak vytvořit dynamickou webovou aplikaci, využití nějakého programovacího jazyka (nejčastěji PHP) bez použití jakéhokoli zjednodušení v podobě frameworku. První PHP frameworky vznikaly až mezi lety 2000–2005 (Jones, 2016). O pár let později vznikl také dosud hojně využívaný JS framework jQuery (Resig, 2009). Skutečnou revoluci ve vytváření webových aplikací způsobily až frameworky AngularJS z roku 2009 (Lerner, 2013) a ReactJS z roku 2013. Velkou roli v tvorbě webových aplikací nyní tvoří také CSS frameworky, z nichž nejznámější je asi Bootstrap.

Abych vytvořil co nejelegantnější řešení a zároveň znovu nevynalézal kolo, budu pro tvorbu svého projektu využívat prostředky:

- PHP framework
- JS MVC framework
- CSS framework
- Databázový systém
- Další komponenty potřebné pro vytvoření moderního uživatelského rozhraní

Všemi zmíněnými prostředky by podle mého názoru měla moderní webová aplikace disponovat.

Výběr PHP frameworku

Nový systém budu implementovat v moderním PHP frameworku. V českém prostředí se nejčastěji používá framework Nette. Vzhledem k tomu, že mám s Nette už nějaké zkušenosti, rozhodl jsem v něm svůj projekt neimplementovat zejména kvůli neúplné dokumentaci, kterou disponuje. Vybíral jsem tedy zahraniční robustní framework. Na základě ankety na webu Sitepoint.com (Skvorc, 2015) jsem zvolil Laravel 5. Ve zmíněné anketě vyhrál s více než dvojnásobným počtem hlasů, než získal druhý umístěný framework Symfony2. V této anketě se na třetím místě umístil i Nette framework, kde ze 770 hlasů pochází 611 z Česka, což svědčí o celkové malé světové komunitě.

Podle M. Beana je Laravel 5 framework vycházející z více než dvaceti nezávislých PHP knihoven, které jsou spravovány balíčkovacím managerem Composer. Proto je velmi jednoduché celé jádro aplikace aktualizovat. Laravel také disponuje předpřipravenými metodami, jak aplikaci testovat. Díky tomu je potom jisté, že se správný kód spouští jen tehdy, kdy se spouštět má, a aplikace tak zbytečně nezatěžuje server. (Bean, 2015, str. 5)

Z těchto důvodů je zřejmě právě Laravel 5 optimálním frameworkem pro můj projekt.

Výběr JS frameworku

U ostatních prostředků bude výběr jednodušší, protože jich není ani zdaleka tolik. U JS MVC frameworků mám na výběr pouze ze dvou, a to React.js a Angular. Ostatní se buď používají poměrně zřídka, nebo jsou zastaralé. S Angularem mám už nějaké zkušenosti a troufám si tvrdit, že se v budoucnu bude používat čím dál více, proto si zvolím jej. S Angularem to ale bohužel ještě není tak jednoduché. Donedávna se totiž masově používal pouze Angilar JS, nedávno však vyšel Angular 2, který není s Angularem JS zpětně kompatibilní. V současné době navíc vznikl problém, jak Angular 2 vlastně označovat, neboť vyšel také Angular 4, který však s Angularem 2 zpětně kompatibilní je. Volím tedy nejnovější stabilní verzi, a to Angular 4. V textu budu nadále Angular 2 a 4 označovat jako Angular 2+.

Hlavní rozdíl mezi AngularemJS a Angularem 2+ je v tom, že Angular 2+ už nevyužívá čistý Javascript, ale využívá nadstavbu Typescript, ten se potom překládá do Javascrptu a mj. umožňuje typovou kontrolu a využívání objektů, jak jsme zvyklí z jazyků Java, C++ nebo PHP.

Výběr ostatních frameworků

V oblasti CSS frameworků mám zkušenosti pouze s tím nejznámějším – Bootstrapem. Pro Angular2+ navíc existuje balík ngx-bootstrap, který zprostředkovává uživatelské rozhraní pro aplikace vytvořené pomocí Angularu, a to právě pomocí Bootstrapu. Co se týče databázového systému, jsem zvyklý na MySQL. V případě webové aplikace s nepříliš rozsáhlou relační databází, jako je tato, asi není třeba se nějak s výběrem databázového systému zevrubněji zabývat. Pro rychlejší vyhledávání nad uloženými zkušebnami chci navíc využít vyhledávacího systému ElasticSearch, který umí vyhledávat na základě mnoha různých parametrů a oproti MySQL je mnohonásobně rychlejší.

4.2 Technologie zblízka

Některé z mnou vybraných technologií jsou možná pro někoho trochu méně známé, proto je v této části stručně přiblížím.

Angular 4

Základní myšlenka, to, k čemu se Angular nejvíce používá, je přenést co nejvíce práce ze serveru na počítač klienta a udělat to co v co nejpřehlednějším programátorském prostředí. AngularJS se používal převážně jako doplněk pro dynamické aplikace vytvořené na straně serveru. V Angularu 2+ už celou webovou aplikaci v podstatě zastřešuje kód stažený do počítače klienta, který se pouze serveru dotazuje o data

a tím do velké míry nahrazuje funkce serverových webových frameworků, ty ale stále můžeme využít pro generování a odesílání dat nejčastěji ve formátu JSON.

Představme si například aplikaci pro správu úkolů. V případě, že nepoužíváme javascriptový framework, musíme nejprve požádat o stránku s úkoly. Na serveru se musí vygenerovat kompletní HTML kód stránky (nebo jeho část, pokud používáme AJAX). Musíme tedy otevřít databázi, požádat o data, všechna data vykreslit a vykreslenou stránku (nebo její část) odeslat klientovi. U to pokaždé, když uživatel nějak pozmění svůj seznam úkolů. V případě využití MVC JS frameworku se načte celá webová aplikace včetně všeho, co na stránce zůstává stejné, v lepším případě jen jednou, protože se tento obsah dá uložit do cache. Následně je server požádán o data, která se na klientské straně udržují po celou dobu, kdy má uživatel stránku otevřenou. V případě, že uživatel chce ve svém úkolníčku něco změnit, odešle pouze požadavek na odstranění a server místo vrácení celé stránky (nebo její části) může vrátit například jen „ok“. Nemusí se dotazovat na data databáze, pouze do ní zapíše, a pokud vše proběhne v pořádku, stačí jen odeslat odpověď, že vše dopadlo dobře. O ostatní už se můžeme postarat na straně klienta.

Angular 4 je poměrně mladá technologie a není ještě příliš rozšířená mezi uživateli, je proto problém sehnat pro ni stabilní bezchybné komponenty jako například pro AngularJS nebo React.

Pojďme se podívat na Hello World aplikaci napsanou v Angularu 2+ (Angular.io, 2016):

```
1 import { Component } from '@angular/core';
2
3 @Component({
4   selector: 'my-app',
5   template: `<h1>Hello {{name}}</h1>`
6 })
7 export class AppComponent { name = 'Angular'; }
```

Na řádku 1 importujeme název třídy z konkrétního balíku, aby překladač věděl, odkud má čerpat při odkazu na třídu *Component*. Na řádku 3 potom voláme anotaci *@Component*, podle čehož Angular pozná, že se jedná o komponentu. V těle potom na řádku 4 označujeme selector, do kterého HTML tagu se má obsah komponenty vepsat. Na 5. řádku definujeme šablonu pro zobrazení. Slovo *name* je tam obaleno znaky {{ a }}. To znamená, že se jedná o proměnou ze třídy, které přísluší tato anotace. Nakonec na 7. řádku definujeme celou třídu *AppComponent*, uvnitř níž deklarujeme a inicializujeme proměnnou *name*, která se následně promítne do šablony. Přínosné je na Angularu to, že pokud se proměnná v čase změní, sám ji změní i v šabloně a my se nemusíme o nic starat (většinou).

Framework Laravel

Jak už bylo zmíněno, Laravel je poměrně robustní PHP framework, ve kterém lze implementovat v podstatě jakoukoli webovou aplikaci bez použití dalších součástí. Já ho ve své práci budu využívat pouze pro výměnu dat mezi databázemi (MySQL a ElasticSearch) a klientskou stranou (Angularem). Z architektury Model-View-Controller (MVC) tedy odpadne slovo view. Zobrazení bude totiž definováno pomocí Angularu na klientské straně aplikace. Důležité je tak pouze routování, controllery a práce s databází.

Routování je v Laravelu řešeno pomocí souboru *routes.php* a je implementovatelné několika různými způsoby. Nejjednodušším z nich je umístit anonymní funkci vracející přímo obsah, který se má odeslat do prohlížeče uživatele. Například takto (Bean, 2015):

```
1 Route::get('/', function () {
2     return ....
3 });
```

V Laravelu je statické vše, co může být statické, takže nepotřebuje dependency injection (iněktování závislostí). Zavoláme tedy přímo statickou funkci *get* ze třídy *Route*. To znamená, že dotaz musí přijít metodou *GET*. Analogicky se takto dají routovat všechny ostatní metody dotazů. První parametr funkce je adresa, na kterou se uživatel dotazuje, druhý parametr je funkce, která se zavolá a vrátí to, co se má odeslat v http odpovědi.

Druhou možností je pak předat dotaz nějaké funkci controlleru:

```
1 Route::get('/', 'Cnt@something');
```

a controller potom vypadá přibližně takto:

```
1  namespace App\Http\Controllers;
2  use App\Http\Controllers\Controller;
3
4  class Cnt extends Controller
5  {
6      public function something()
7      {
8          return ...
9      }
10 }
```

Třída *Cnt* musí dědit od třídy *Controller* a musí obsahovat funkci, na kterou odkazujeme. Potom stačí vrátit potřebný obsah. Možností jak routovat najdeme v Laravelu nespočet, rozhodně jeho možnosti routování hluboce přesahují požadavky běžného programátora.

Další užitečnou součást Laravelu je práce s databází. Laravel totiž využívá poměrně unikátní systém Eloquent, což je systém, který data z relačních databází

převádí na objekty (ORM), s nimiž potom programátor pracuje. Na rozdíl od ORM Doctrine pracuje Eloquent s již připravnými tabulkami, které nijak nemění. Dá se říct, že se jedná o opačný přístup. Doctrine sama tabulky vytváří a mění na základě definic modelů vystupujících jako třídy v PHP.

Jednoduchý model lze definovat například takto (Laravel.com, 2016):

```
1  <?php
2
3  namespace App;
4
5  use Illuminate\Database\Eloquent\Model;
6
7  class Flight extends Model
8  {
9      protected $table = 'my_flights';
10 }
```

Tím dáváme Eloquentu vědět, že existuje nějaká tabulka *my_flights*, a že pokud budeme pracovat s instancemi třídy *Flight*, pracujeme ve skutečnosti s daty v této tabulce. V dalším kódu pak můžeme například požadovat všechny řádky této tabulky (Laravel.com, 2016):

```
1  $flights = Flight::all();
```

nebo například vytvářet nová data (Laravel.com, 2016):

```
1  $flight = new Flight;
2  $flight->name = $request->name;
3  $flight->save();
```

Pomocí funkce *save* se pak data uloží do databáze. Zmíněná funkcionalita samozřejmě není vyčerpávající. Eloquent dokáže pracovat mj. s relačními daty, ty pak upravovat, přidávat, mazat atd.

Bootrstrap

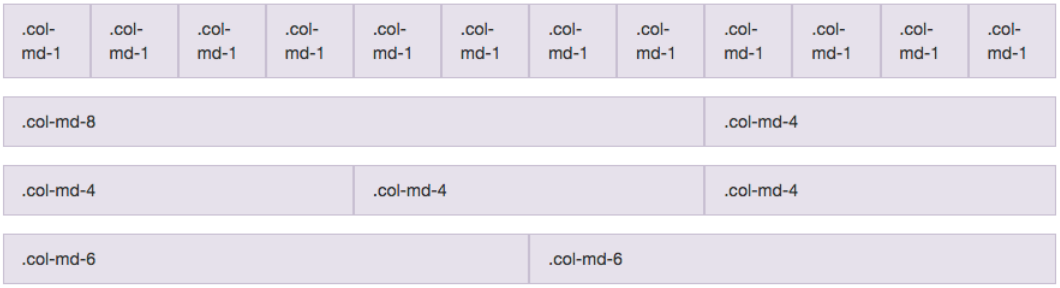
Bootstrap je css framework, který disponuje velkou spoustou komponent přes formulářové prvky po dialogy. Nejdůležitější součástí je však rozdělení stránky do mřížky. Standardně Bootstrap rozděluje stránku, popřípadě její část na 12 stejných sloupců, které se na menších zobrazovacích zařízeních skládají pod sebe. Tím je zajištěno, že se obsah zobrazí přirozeně na různých zařízeních (responzivně).

Mřížka se v Bootstrapu implementuje například takto (getbootstrap.com, 2016):

```
1 <div class="row">
2   <div class="col-md-1">.col-md-1</div>
3   <div class="col-md-1">.col-md-1</div>
4   <div class="col-md-1">.col-md-1</div>
```

```
 5    <div class="col-md-1">.col-md-1</div>
 6    <div class="col-md-1">.col-md-1</div>
 7    <div class="col-md-1">.col-md-1</div>
 8    <div class="col-md-1">.col-md-1</div>
 9    <div class="col-md-1">.col-md-1</div>
10    <div class="col-md-1">.col-md-1</div>
11    <div class="col-md-1">.col-md-1</div>
12    <div class="col-md-1">.col-md-1</div>
13    <div class="col-md-1">.col-md-1</div>
14  </div>
15  <div class="row">
16    <div class="col-md-8">.col-md-8</div>
17    <div class="col-md-4">.col-md-4</div>
18  </div>
19  <div class="row">
20    <div class="col-md-4">.col-md-4</div>
21    <div class="col-md-4">.col-md-4</div>
22    <div class="col-md-4">.col-md-4</div>
23  </div>
24  <div class="row">
25    <div class="col-md-6">.col-md-6</div>
26    <div class="col-md-6">.col-md-6</div>
27  </div>
```

Vykreslený obsah je pak znázorněn na obrázku 5.



Obrázek 5: Ukázka vykreslení mřížky v css frameworku Bootstrap

Při vlastní implementaci aplikace budu postupovat tak, aby prvky, které jsou vyžadovány jinými prvky, byly implementovány první. Nebudu tak muset generovat testovací data, které bych při implementaci potřeboval pro kontrolování práce, a při jejich ručním vytváření zároveň budu již naprogramované součásti testovat. Nejprve tedy vyřeším registrace uživatelů, potom jejich přihlašování, vytváření profilů zkušeben, zobrazení profilů zkušeben, detaily profilů zkušeben a na závěr doladím design systému a připravím systém pro produkci.

4.3 Vytvoření projektu

Před zahájením implementace je třeba vytvořit nové projekty v rámci prostředí framewroku Laravel a následně také v rámci frameworku Angular 4.

V případě frameworku Laravel se pouze přes balíčkovací systém composer nainstaluje aplikace *laravel/installer*, pomocí které se nový projekt sám vygeneruje. V rámci frameworku Angular 4 existuje podobný přístup. Pomocí balíčkovacího systému npm se nainstaluje aplikace *angular-cli*. Díky této aplikaci probíhá nejen vytvoření nového projektu, ale kompletní vývoj. Pro vývoj pomocí této aplikace stačí v příkazové řádce zadat příkaz „ng serve“ a spustí se vývojový webový server. Když potom adresu zobrazíme v prohlížeči, aplikace udržuje s prohlížečem spojení pomocí websocketové komunikace. Při změně nějakého souboru v adresářové struktuře projektu se aplikace sama přeloží a prohlížeč obnoví obsah. Příkazem „ng build” se potom výsledná aplikace sestaví, aby byla použitelná v běžném prohlížeči a bez použití zmíněného serveru (ng serve).

Internacionalizace

Jedním z cílů mé práce je implementovat aplikaci tak, aby byla internacionalizovaná. Pokud se překládání webové aplikace nevyřeší v začátcích její implementace, je to v budoucnu vždy poměrně velký problém, protože se pak musí všude vyhledávat texty k překladu a často se najdou zádrhely, proč následný překlad není jednoduše řešitelný.

V rámci Angularu 2+ existují pro internacionalizaci dvě základní cesty. Jednou z nich je využití internacionalizační systém, kterým Angular disponuje v základní instalaci. Druhou možností je pak využít balík *ngx-translate*. Já jsem si zvolil druhou cestu, protože první systém zatím není úplně ideálně vyladěn, data se navíc načítají z poměrně komplikovaného XML souboru. Naproti tomu *ngx-translate* (Combe, 2016) umožňuje exportovat překlady do formátu .po, které se poměrně snadno editují v programu Poedit, s nímž mám velmi pozitivní zkušenosti.

Po importu a registraci všech potřebných součástí se systém *ngx-translate* používá následovně (Combe, 2016)

```
1          <div>{{ 'HELLO' | translate:param }}</div>
```

v případě použití v šabloně. Pro samotné přeložení se používá roura *translate*, která text přeloží. Pokud text obsahuje nějaký parametr, lze jej translátoru také předat.

Pokud potřebujeme text přeložit jinde než v šabloně, je to možné pomocí služby *TranslateService*, jejíž instanci získáme pomocí dependency injection.

4.4 Registrace uživatelů

Pro registrování uživatele je potřeba nejprve implementovat formulář se základními údaji. Vzhledem k tomu, že aplikace by měla být pro uživatele co nejjednodušší, budu při registraci požadovat pouze email a heslo. Na formuláře se v Angularu dá použít

systém tzv. reaktivních formulářů. Jde o o členění formulářových prvků do skupin (celý formulář v je také skupina), každý prvek se pak zaregistruje příslušnému formulářovému prvku v HTML šabloně, Výhodou je, že tento systém disponuje metodami pro ověřování zadaných dat (validování) vč. asynchronních validátorů (většinou validátory, které vyžadují komunikaci se serverem).

Vytvořím si tedy formulářovou skupinu, do které zaregistruji jednotlivé prvky formuláře. Abych nemusel každý prvek vytvářet jako instanci třídy *FormControl*, mohu použít službu *FormBuilder*, jejíž instanci *this.fb* jsem získal přes dependency injection v konstruktoru.

```
1  buildForm(): void {
2      this.registrationForm = this.fb.group({
3          'email': ['', [
4              Validators.required,
5              NGValidators.isEmail(),
6          ],
7              this.registrationService.
8              isEmailUsed.bind(this.registrationService)
9          ],
10         password': this.password,
11         password2': ['', [Validators.required,
12                                     this.passwordMatch()]]
13             });
14     }
```

Metoda group třídy *FormBilder* vrací instanci třídy *FormGroup*. Jejím parametrem je objekt, jehož názvy proměnných odpovídají jménům formulářových prvků. Pro každý formulářový prvek opět definujeme pole, jehož první hodnotou je implicitní hodnota, druhou hodnotou je pole validátorů, třetí hodnotou je asynchronní validátor.

Když následně v šabloně například u textového vstupního pole pro email použijeme atribut *[formControl]* nebo *formControlName*, hodnota tohoto prvku bude automaticky synchronizována s hodnou formulářové skupiny.

```
1 <input type="text" class="form-control" id="email"
2            formControlName="email" required>
```

Asynchronní validátor pro kontrolu e-mailu je implementován ve službě *RegistrationService*:

```
1      public isEmailUsed(c: AbstractControl) {
2          return new Promise((resolve) => {
3              this.http.get(AppModule.API_URL+
4                  'registration/isEmailUsed/' + c.value)
5                      .subscribe(res => {
6                  if (res.json() === true) {
```

```
7                       resolve({'isEmailUsed': true});
8                   }
9                   else {
10                      resolve(null);
11                  }
12              });
13          });
14      }
```

Funkce vrací instanci třídy *Promise*, což je jednoduchá třída pro práci s asynchronními daty. Vše, co potřebujeme v rámci asynchronního požadavku vrátit, dáme do parametru funkce *resolve()*. V mém případě čekám, až se ze serveru vrátí a zpracuje požadavek, zda je e-mail k dispozici. Až se tak stane, vrátím *true*, pokud je e-mail už používán. V opačném případě vrátím hodnotu *null*. O vše ostatní už se postará Angular. Ten za nás navíc ověří i to, jestli jsou hodnoty zadány v pořádku. Případné chyby lze vypsat například takto:

```
1  <div *ngIf="!registrationForm.controls.email?.valid &&
2            !registrationForm.controls.email?.pending &&
3            registrationForm.controls.email?.touched
4          " class="alert alert-danger">
5        <div *ngIf="registrationForm.controls.email?.
6                                        hasError('required')">
7       {{'edit_profile_email_error_required' | translate}}
8        </div>
9        <div *ngIf="registrationForm.controls.email?.
10                                       hasError('isEmail')">
11     {{'edit_profile_email_error_not_valid' | translate}}
12       </div>
13       <div *ngIf="registrationForm.controls.email?.
14                                hasError('isEmailUsed')">
15    {{'edit_profile_email_error_email_used' | translate}}
16      </div>
17 </div>
```

Jak je z kódu vidět, instance třídy *FormGroup* disponuje mnoha proměnnými a metodami pro snadnější práci se stavem a chybovými stavy formuláře. Z kódu je také patrná práce s překlady. Obdobně je vyřešen i zbytek formulářových prvků.

Po kliknutí na tlačítko registrovat se zavolá funkce *onRegistrer*:

```
1       public onRegister() {
2           this.registrationService.
3           register(this.registrationForm.value).
4           subscribe(
5               res => {
```

```
6                   if (res.status == 200) {
7                       this.auth.logUser(res.json());
8                       FlashService.addMessage(
9                       {severity:'success',
10                       summary:this.translate.
11                       instant("Registered!"),
12                       detail:this.translate.
13                      instant("you_are_now_registered")});
14                      this.router.navigate(['']);
15                  }
16              },
17              err => {
18                  this.serverErrors = [];
19                  let data = JSON.parse(err._body);
20                  console.log(data);
21                  for (let val in data) {
22                      console.log(val);
23                    this.serverErrors.push(data[val][0]);
24                  }
25              }
26          );
27      }
```

Tato funkce zavolá funkci *register* ze služby *registrationService* s hodnotami formuláře. Ta neudělá nic jiného než to, že data odešle na server a vrátí jeho odpověď. V případě pozitivní odpovědi uživatele přihlásíme a dáme mu vědět, že byl registrován a přihlášen, v případě záporné odpovědi alespoň uživateli sdělíme chyby (to by se ale nemělo stát, protože data už jsou ověřena validátory).

Implementace na straně serveru potom vypadá takto:

```
1   public function register(Request $request) {
2         $val = Validator::make($request->toArray(), [
3             'password' => 'required|min:8|max:200',
4             'email'    =>
5                   'required|email|unique:users,email',
6         ]);
7         if ($val->fails()) {
8       return \Response::json($val->messages(), 400);
9         }
10        try {
11            User::create([
12                    'email'    =>
13                      $request->input('email'),
14                    'password' =>
```

```
15                      bcrypt($request->input('password')),
16                  ]);
17
18              $token = JWTAuth::attempt([
19                              'email'     =>
20                      $request->input('email'),
21                              'password' =>
22                      $request->input('password'),
23                              ]);
24
25              return response()->json(['token' => $token,
26                          'user' => Auth::User()]);
27          } catch (\Exception $e) {
28              return "error";
29          }
30      }
```

Pomocí validátoru Laravelu ověřím, zda jsou data v pořádku, a v případě úspěchu je uložím do databáze, uživatele přihlásím a nazpět mu vrátím *JWT token* a jeho údaje, v opačném případě vrátím chybovou hlášku. Obsah *JWT tokenu* upřesním v další podkapitole.

4.5 Přihlašování uživatelů

V běžných aplikacích implementovaných v PHP je přihlašování uživatelů poměrně jednoduchou záležitostí. Serveru odešlu přihlašovací údaje a on mi buďto vrátí chybu, nebo pozitivní odpověď a do cookies uloží *session id*, kterým se následně klient při požadavcích autorizuje. V případě Angularu se tento způsob už nepoužívá. Místo *session id* server vrací tzv. *JWT token*, což je zakódovaný identifikátor uživatele a čas vypršení samotného tokenu (JWT.io, 2016). V praxi ale nastává problém s tím, co dělat, když token vyprší, a jak ověřovat, jestli se třeba uživatel neodhlásil v jiném okně nebo záložce. Pro ukládání tokenu se totiž používá pro doménu sdílené prostředí *localStorage*, což je prostředí, které se na rozdíl od cookies neodesílá na server při každém dotazu. Problém je ale v tom, že v rámci aplikace tohoto druhu mohu udělat spoustu úkonů, aniž bych poslal požadavek na server. V PHP mohu prakticky kdykoli uživatele někam přesměrovat, v Angularu to ale tak jednoduché není. Uživatel může například vejít na stránku s editací profilu, ale požadavek se odesílá až v momentě, kdy skutečně něco změním. Problém nastává, když se na serveru z jakéhokoli důvodu token zneplatní a na klientský prohlížeč se pak vrátí negativní odpověď. Abych toto riziko odvrátil, musel bych u každého požadavku nejprve ověřit, zda nedošlo k odhlášení, a až potom ho předat té komponentě nebo službě, která ho skutečně vyslala. Proto je lepší variantou hlídat čas vypršení tokenu a v pravidelných intervalech ho obnovovat, aby nevypršel. Zvolil jsem tedy dobu

platnosti tokenu 2 týdny (v rámci aplikace nepracujeme s žádnými citlivými daty) a obnovu tokenu každých 5 minut.

Pojďme se tedy podívat, jak probíhá samotná autentizace. Uživatel vyplní přihlašovací údaje a klikne na tlačítko přihlásit. V tom okamžiku se zavolá metoda *onLogin*

```
onLogin(credentials) {
        let that = this;
        this.auth.login(credentials).then(function () {
            that.router.navigate(['/'])
        }).catch((error) => {
            window.console.error(error);
        });
    }
```

V jejím těle se pak volá funkce *login* ze služby *AuthService*, která vrací již známou instanci třídy *Promise*, při pozitivním výsledku je uživatel přesměrován na kořenovou stránku. Samotná funkce *login* je implementována takto:

```

 login(credentials) {
        let that = this;
        return new Promise(function (resolve, reject) {
            that.http.post('
                AppModule.API_URL+'authenticate',
                credentials)
             .map(res => res.json())
             .subscribe(
                    function (data) {
                        that.logUser(data);
                        that.token = data.token;
                        that.error = '';
                        resolve(data);
                        that.autoRefreshToken();
                    },
                    error => {
                        that.error = error;
                        reject(error)
                    }
                );
        });
    }
```

Při pozitivním výsledku uloží token a data o uživateli do *localStorage*, smaže případnou chybu z předchozího pokusu, spustí automatickou obnovu tokenu a vrátí

pozitivní výsledek.

Asi je také potřeba objasnit, jak vlastně funguje samotná obnova tokenu. Fuknce jednoduše každých 5 minut odešle požadavek na nový token. Server mu buď odpoví novým tokenem, nebo vrátí negativní odpověď v případě, že token již vypršel nebo že je neplatný. V druhém případě je pak nutné uživatele, který se nachází na nějaké stránce, jež je přístupná pouze přihlášeným, přesměrovat a oznámit mu, že už není přihlášen. To je zabezpečeno tím, že každá komponenta, která se zobrazuje pouze přihlášeným, dědí od třídy *BaseAuthComponent*, v níž je definována fuknce *ngOnInit*, která se vždy automaticky zavolá při načtení komponenty, pokud třída implementuje rozhraní *OnInit*. V rámci této funkce je inicializován posluchač změny stavu přihlášení. Pokud se uživatel odhlásí, je automaticky přesměrován na domovskou stránku.

4.6 Vytvoření profilu zkušebny

Aby mohli případní inzerenti vkládat svoje zkušebny, bylo potřeba jim za tímto účelem vytvořit rozhraní a vložená data následně vložit do databáze. Pro přehlednost celého procesu jsem se rozhodl rozdělit celou registraci do sedmi kroků:

1. Volba polohy zkušebny
2. Zadání základních informací
3. Zadání ceny a informací o ní
4. Vložení obrázků
5. Volba vybavení
6. Zadání otevírací doby a rozvrhu
7. Zadání komentáře

Pro každý krok jsem vytvořil samostatnou komponentu. Pro každý formulář každého kroku jsem vytvořil také samostatnou komponentu, kterou v budoucnu použiji při editaci takto vytvořeních dat. Komponenty všech kroků vypadají velmi podobně. Každá z nich implementuje metody *ngOnInit*, *formSubmitted* a *previousClicked*.

V metodě *ngOnInit* ověřím, jestli byl správně zadán předchozí krok, a jinak uživatele přesměruji na první krok. Kroky jsou na sobě závislé. Metoda *formSubmitted* je volána při potvrzení formuláře a ukládá data do *localStorage*. Také zapíše, že uživatel bodem prošel a všechny položky správně vyplnil (formulář totiž nelze odeslat, aniž by byla data vyplněna správně) Totéž dělá i metoda *previousClicked*, která data také uloží, ale už neuloží informaci o tom, že uživatel data vyplnil správně. Šablona komponenty pak obsahuje nadpisy každého kroku a komponentu každého formuláře.

Volba polohy

Volba polohy je uživateli umožněna pomocí google mapy. Pro Angular 2+ existuje balík *SebmGoogleMap*, který celou práci s mapami velmi usnadňuje. Vše potřebné pro vykreslení samotné mapy a vyznačení bodů je pak realizováno pomocí šablony (Müller, 2016):

```
1 <sebm-google-map [latitude]="lat" [longitude]="lng">
2   <sebm-google-map-marker
3       [latitude]="lat"
4       [longitude]="lng">
5   </sebm-google-map-marker>
6 </sebm-google-map>
```

Tag *sebm-google-map* zajistí vykreslení mapy se středem v bodě $[lat, lng]$. O vykreslení bodů se stará komponenta označená HTML tagem *sebm-google-map-marker*.

Moje komponenta pro výběr polohy zkušebny obsahuje mapu a řádek pro zadání adresy. Uživatel si může vybrat, jestli zadá adresu nebo jestli místo vybere na mapě.

Komponenta pak obsahuje funkci *mapClick($event)*, která po kliknutí získá z google map api informace o vybraném bode. Ze získaných informací využívám google identifikátor místa (place ID) a adresu. Tato adresa je automaticky doplněna do adresního řádku. Naopak při zadání adresy se z google api získá informace o poloze a tato poloha je znázorněna v mapě.

Výsledný formulář před doladěním stylů je na obrázku 6.

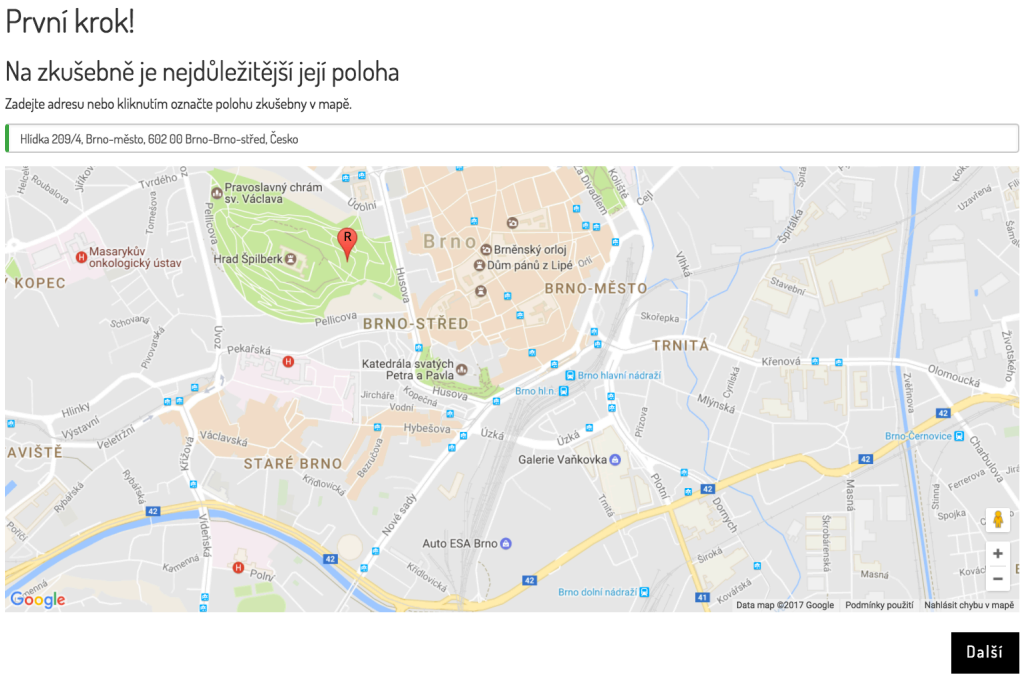
Základní informace

V rámci komponenty pro zadání základních údajů je potřeba od uživatele získat jméno zkušebny, zda je sdílená či nesdílená, zda je vybavená či nevybavená, jestli má uživatel zájem o to, aby zkušebna byla inzerována, a zda má zájem evidovat rozvrh hodin. Kromě jména jsou všechny tyto informace dvoustavové (ano/ne), proto jsem se pro jejich vykreslení rozhodl použít komponentu *InputSwitch* z balíku *PrimeNG*, což je balík uživatelského rozhraní pro Angular 2+ (PrimeTek, 2017). Toto řešení se mi zdá o něco jasnější než použít klasický checkbox.

Výsledná komponenta před doladěním stylů je na obrázku 7.

Cena

Provozovatelé hudebních zkušeben řeší ceny pronájmů mnoha způsoby. Po důkladném zvážení jsem dospěl k tomu, že nelze všechny možnosti komplexně obsloužit tak, aby se nad nimi dalo přehledně filtrovat. Proto jsem se rozhodl přistupovat k cenám pomocí obvyklé ceny, kterou kapela zaplatí za měsíc. Uživatel pak k této obvyklé ceně může ještě zadat měnu a komentář, kde může vysvětlit, jak



Obrázek 6: Komponenta pro zvolení polohy zkušebny

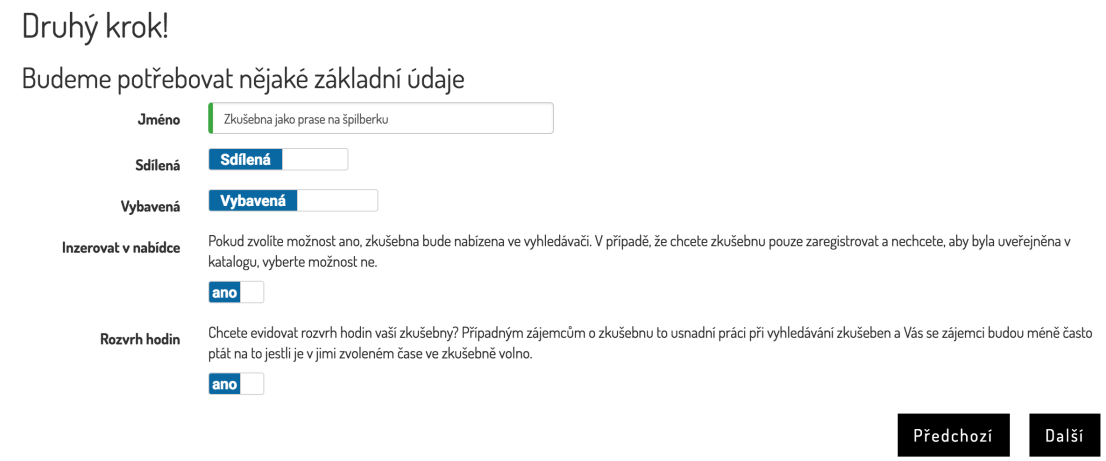
se cena skutečně určuje. Komponenta tak obsahuje tři jednoduché položky. Výsledek před dokončením překladů a doladěním stylů lze vidět na obrázku 8.

Obrázky

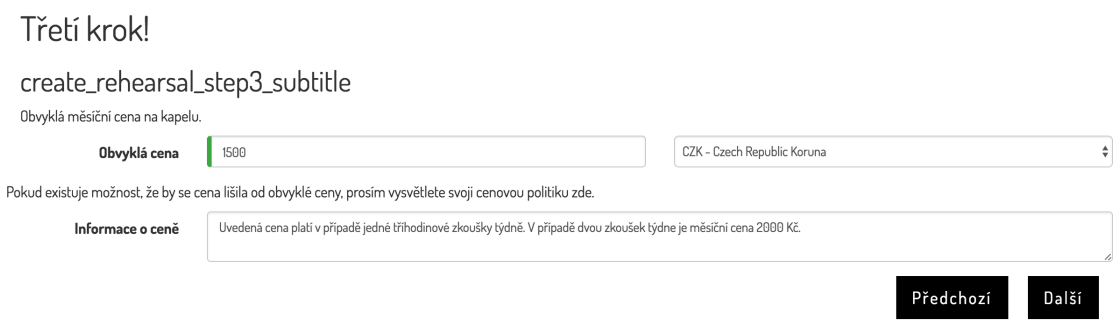
O poznání zajímavější než předchozí komponenty je komponenta pro nahrávání obrázků. Nahrané obrázky se totiž uživateli hned zobrazí, může si je tak hned prohlédnout a nemusí čekat až na finální odeslání formuláře.

Aby uživatel mohl obrázky jen přetáhnout a nemusel je vždy jen vybírat z dialogového okna, využil jsem direktivy *ng2FileDrop* (Winder, 2017). Po vložení jednoho nebo více obrázků se obrázky ihned zobrazí (prostá data z obrázku se připojí ke zdrojovému kódu stránky v prohlížeči díky možnosti zobrazit obrázky kódované v base64) a obrázky se začnou ihned po jednom nahrávat na server díky balíku *ng2-file-upload* (Shekhovtsov, 2017). Této knihovně pouze předáme adresu, na kterou se má obrázek nahrávat, a knihovna už udělá vše za nás. Knihovna také poskytuje informaci o tom, kolik procent už je nahráno, což se hodí ke grafickému vykreslení průběhu.

Po nahrání se ze serveru vrátí dočasný unikátní identifikátor obrázku, který se používá pro následné uložení obrázku do databáze nebo pro odstranění obrázku ze serveru v případě, že uživatel obrázek ihned smaže. Komponenta před dokončením překladů a stylů je vidět na obrázku 9.



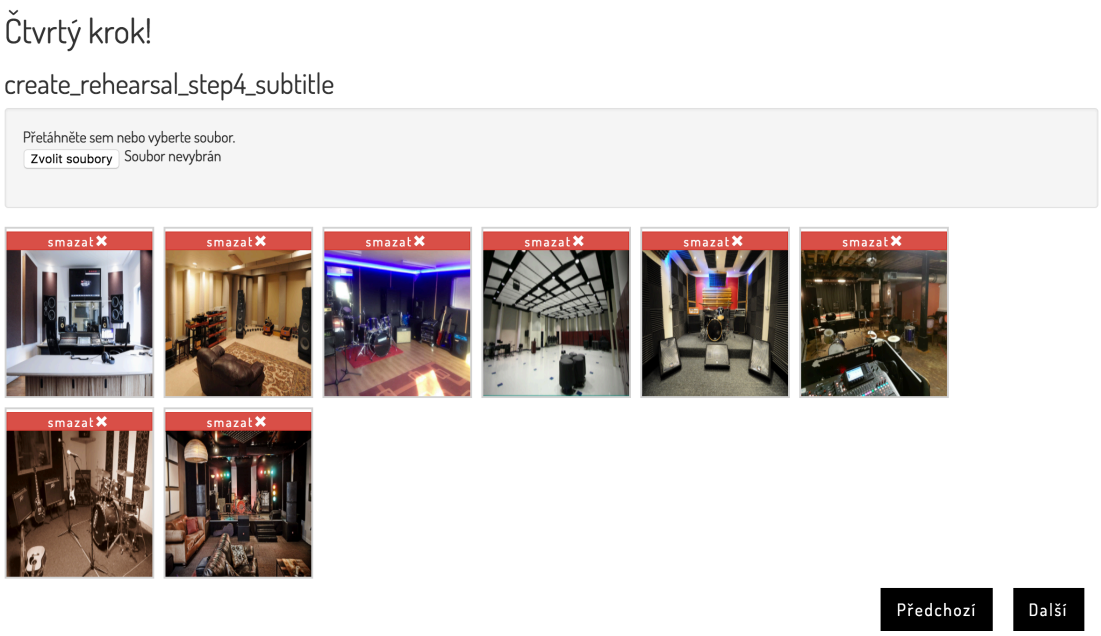
Obrázek 7: Komponenta pro zadání základních údajů



Obrázek 8: Komponenta pro zadání zadání ceny

Evidence vybavení

V dalším kroku vytvářecího průvodce uživatel zadává, jakým vybavením jeho zkušebna disponuje. Uživatel tedy může přidávat a odebírat počet položek vybavení dle libosti. To je zabezpečeno tím, že v Angularu 2+ existuje třída *FormArray*. Formulářové skupiny typu *FormGroup* lze do takového pole přidávat nebo z něj mazat. Aby se nad takovými daty dalo v budoucnu filtrovat, musí uživatel vybrat typ položky z rozbalovacího seznamu. Aby rozbalovací seznam měl vyhledávání a dal se snáz graficky upravovat, použil jsem komponentu *basvandenberg/ng-select* (Berg, 2017). Položky se do seznamu načítají asynchronně ze serveru a nacházejí se v tabulce *equipment_types*. Po vybrání typu vybavení je ještě třeba zadat počet a komentář k danému vybavení. Výstupem tohoto kroku je pak pole jednotlivých položek vybavení, které uživatel vložil. Výsledná komponenta je před doladěním stylů a překladů na obrázku 10.



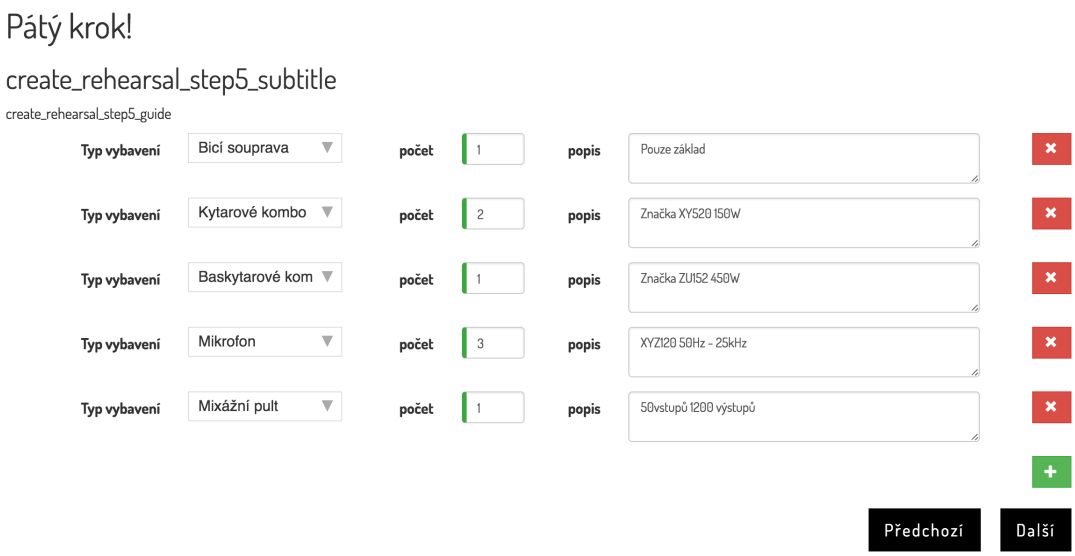
Obrázek 9: Komponenta pro nahrání obrázků

Rozvrhy

Bezesporu nejkomplikovanější součástí celého systému je evidence rozvrhů zkušebny. Uživatel musí zadat otevírací dobu zkušebny, která se v různých dnech týdnu může lišit, musí zadat, v kterých časech je zkušebna obsazená, a tohle všechno se musí přehledně vykreslit do rozvrhu (obr. 20). Nejprve je třeba zvolit, zda jsou otevírací hodiny ve všechny dny stejné, na základě tohoto pole je buďto vykreslen jeden časový interval, nebo sedm intervalů pro každý den.

Po zadání otevíracích hodin může uživatel přidávat zkoušky kapel. Po kliknutí na tlačítko „plus“ se zobrazí dialogové okno s nastavením jména zkoušky, dnem a časem zkoušky (obr. 12). Během zadávání těchto údajů se data rovnou mění v zobrazeném rozvrhu. Po kliknutí na „uložit“ se dialog uzavře a data zůstanou vykreslena v rozvrhu. Pokud uživatel klikne na tlačítko zavřít, data se smažou. Pro editaci stačí na zkoušku kliknout a uživateli se opět zobrazí stejný dialog s možností zkoušku i smazat.

Samotný rozvrh je pak vykreslen pomocí třech tabulek, které jsou absolutně pozicovány na sebe. První tabulka vykresluje ohraničení a horní řádek tabulky s časy a popisky dnů. Druhá tabulka vyznačuje otevírací hodiny, takže první sloupec a řádek tabulky jsou prázdné, aby nepřekrývaly popisky. Zbytek buněk na každém řádku je pak sloučen do jedné a v rámci takové buňky jsou absolutně pozicovány blokové prvky, které mají červené pozadí a označují nepřístupnost zkušebny v daném intervalu. Obdobně je potom řešeno i zobrazení jednotlivých zkoušek.



Obrázek 10: Komponenta pro evidenci vybavení

Obrázek 11: Komponenta pro vložení rozvrhu zkušebny

4.7 Inzerce zkušeben

Nejdůležitější součástí celé aplikace je určitě inzerce zkušeben. Návštěvníkům musejí být zkušebny přehledně zobrazeny, aby je mohl snadno porovnávat. Proto jsou zkušebny zobrazeny na mapě a lze nad nimi filtrovat, aby se uživatel na webu neztratil.

Domovská stránka aplikace

Když uživatel navštíví aplikaci s tím, že chce vyhledávat zkušebny, nejprve se ocitne na domovské stránce aplikace. Domovská stránka aplikace je vytvořena tak, aby byla co nejjednodušší. Uživatel zadává pouze obec, ve které chce vyhledávat,

Obrázek 12: Dialogové okno pro vložení zkoušky

a aplikace ihned vyhledá dané zkušebny. Vzhled domovské stránky je na obrázku 13. Napovídání adres je zajištěno díky *Maps API* od společnosti Google.

Obrázek 13: Domovská stránka aplikace

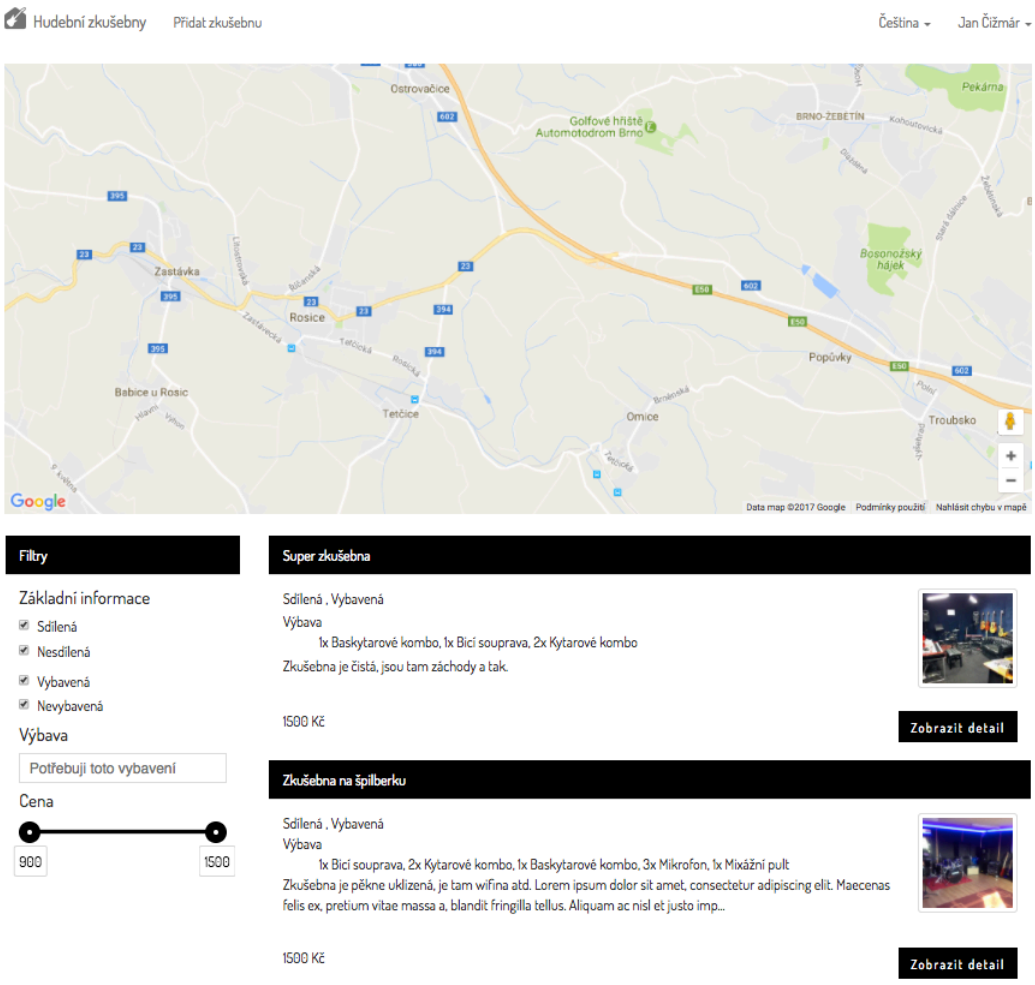
Výpis zkušeben v daném městě

Poté co na domovské stránce uživatel vybere město, je přesměrován na stránku se zobrazením zkušeben (obrázek 14). Zkušebny jsou na serveru automaticky indexovány do systému ElasticSearch, který mj. umožňuje vyhledávat záznamy dle vzdálenosti od určitých souřadnic, a to velmi rychle. Díky tomu snadno získáme indexované záznamy v závislosti na jejich poloze. O získání dat se stará následující funkce (ElasticSearch, 2017):

```
1  public function getRehearsalsByGeoLocation($lat, $lng)
2  {
3      $query = [
4          "filtered" => [
5              "filter" => [
6                  "geo_distance" => [
7                      "distance" => "50km",
8                      "location" => [
9                          "lat" => $lat,
10                         "lon" => $lng,
11                     ],
12                 ],
13             ],
14         ],
15     ];
16     return \Response::json(
17             RehearsalRoom::elasticSearch($query)
18     );
19 }
```

Ze serveru se stáhnou data ve formátu JSON se všemi zkušebnami v daném městě. Zde je možné namítnout, že v případě, že ve městě bude skutečně hodně zkušeben, bude soubor značně velký. Ze zkušenosti ale vím, že například v Brně je maximálně 20 zkušeben, které někdo inzeruje. I kdybychom se bavili o opravdu velkých městech s mnoha zkušebnami, soubor nikdy nebude větší než 500 kB. Pokud budeme uvažovat *gzip* komprimaci, bude tak soubor ještě menší (max. 150 kB). Na oplátku získáme velmi velké omezení dotazů na server. V ideálním případě se klient zeptá pouze jednou, případně dvakrát, když chce zobrazit detail. V horní části stránky je zobrazena mapa, na které jsou zkušebny vyznačeny. Pod ní jsou možnosti filtrování (obrázek 15) a samotný seznam zkušeben (obrázek 16).

Pokud nad zkušebnami začneme filtrovat, ihned se aktualizuje seznam zkušeben i s vyznačením na mapě. Hodnoty z nastavení filtru jsou také ihned přenášeny do adresy prohlížeče, aby uživatel mohl odkaz na vyfiltrovaný obsah někomu poslat. Jak je vidět na obrázku 15, můžeme zvolit, zda chceme zobrazit zkušebny sdílené,



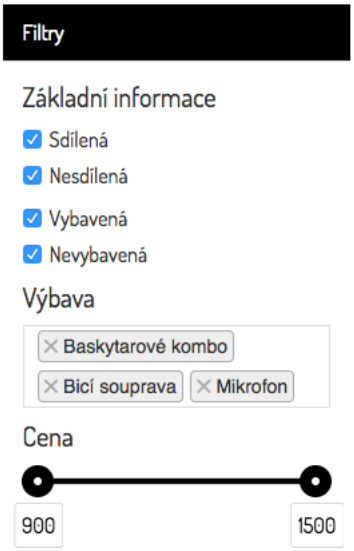
Obrázek 14: Zmenšená stránka s výpisem zkušeben

nesdílené, zkušebny vybavené, nevybavené, zkušebny disponující určitým vybavením nebo zkušebny s maximální či minimální cenou.

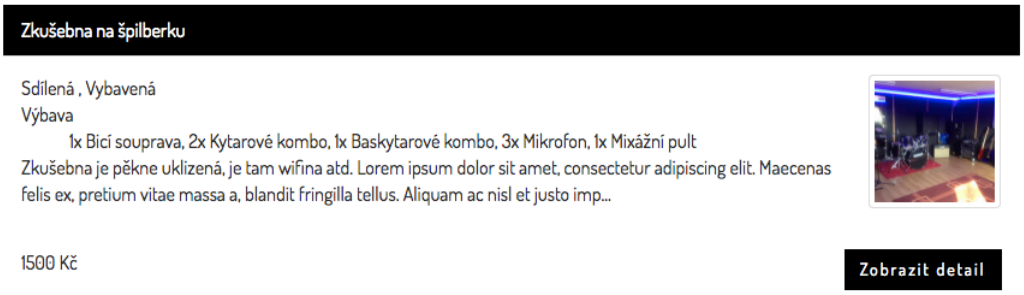
Pro filtraci vybavení jsem použil komponentu ng-select, která umožňuje výběr více položek z rozbalovacího seznamu (Berg, 2017), pro filtrování nad cenou jsem použil komponentu NoUiSlider (Gersen, 2017).

V samotném výpisu zkušeben je vidět nadpis zkušebny, základní údaje, výbava zkušebny, komentář a cena. Ostatní informace uživatel může získat až po kliknutí na tlačítko zobrazit detail.

Při označení zkušebny v mapě se mapa zmenší (obrázek 17) a vedle ní se objeví zmenšený detail zkušebny (obrázek 18). V rámci tohoto detailu jsou zobrazeny časy zkoušek a otevírací hodiny zkušebny, protože grafická podoba rozvrhu by se do vybraného prostoru nevešla; lze ji však zobrazit po kliknutí na tlačítko zobrazit rozvrh.



Obrázek 15: Možnosti filtrování ve výpisu zkušeben



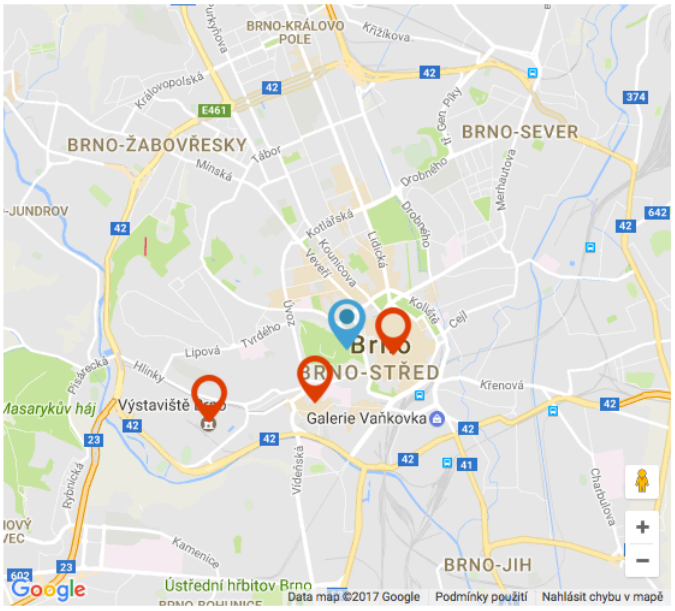
Obrázek 16: Zobrazení zkušebny ve výpisu zkušeben

Detail zkušebny

Po kliknutí na „zobrazit detail“ je uživatel přesměrován na stránku s detailem zkušebny. V horní části stránky je zobrazen střídající se výřez fotografií zkušebny, v jeho rámci je potom zobrazen název a cena zkušebny. Aby byl text viditelný na jakémkoli pozadí, je za ním stín. To umožňuje CSS3 vlastnost *text-shadow* (obrázek 19).

Dále je v detailu zobrazeno vybavení a rozvrh (obrázky 20, 21). Na tyto oddíly navazuje zobrazení fotografií zkušebny. Na stránce jsou fotografie ve zmenšené podobě (obrázek 22). Po jejich rozkliknutí se zobrazí obrázek v plné velikosti. Pod galerií je umístěna mapa s označením polohy zkušebny.

Konečně v dolní části je zobrazen formulář pro odpověď na inzerát (obrázek 23). Formulář je vybaven ověřením, že uživatel není robot. Tato ověření se používají zejména proto, že na internetu existují roboti, kteří automaticky vyhledávají takovéto formuláře, do obsahu vkládají spamové zprávy a následně je odesílají. Uživatelé by tak mohli dostávat nevyžádané e-maily z této stránky. Kdyby pak



Obrázek 17: Zmenšená mapa zkušeben při výběru zkušebny

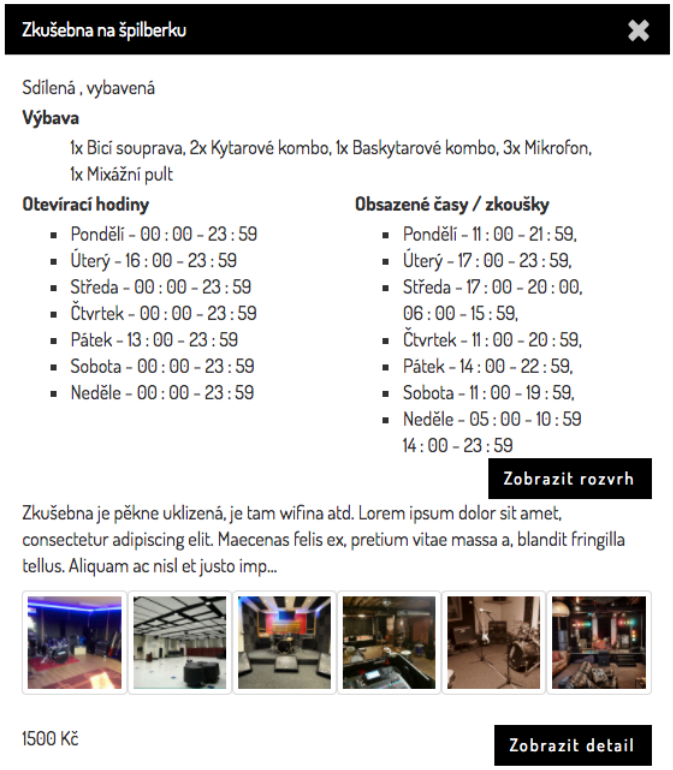
takové zprávy označovali jako spam, mohl by nastat problém, že by velké SMTP servery vyhodnocovaly i relevantní odpovědi jako nevyžádané. Tato ochrana je realizována díky službě *reCaptcha* od společnosti Google (Google, 2017). Odeslání formuláře je tak povoleno až uživatelům, kteří touto ochranou projdou. Kdyby však někdo zachytil odchozí dotaz z aplikace a opětovně jej odesílal, mohl by tuto ochranu snadno obejít. Proto je na serveru ještě před osedláním e-mailu ověřeno, zda uživatel, od kterého dotaz pochází, skutečně ochranou prošel. Samotné odesílání e-mailů zabezpečuje třída *Mail* Laravelu. Na základě nastavení SMTP server e-mail odešle požadovanou službou.

Díky těmto komponentám může návštěvník najít zkušebnu v požadované lokalitě, získat o ní všechny potřebné informace a v případě zájmu odpovědět na inzerát.

4.8 Úprava zkušeben

Pokud je uživatel přihlášen a má vytvořené nějaké zkušebny, jejich přehled snadno zobrazí díky položce „Moje zkušebny“ v rozbalovacím menu pod svým jménem v horní části obrazovky. Uživatel se tak dostane na stránku s výpisem svých zkušeben, kde může buďto zobrazit detail zkušebny, editovat zkušebnu, nebo ji smazat (obrázek 24).

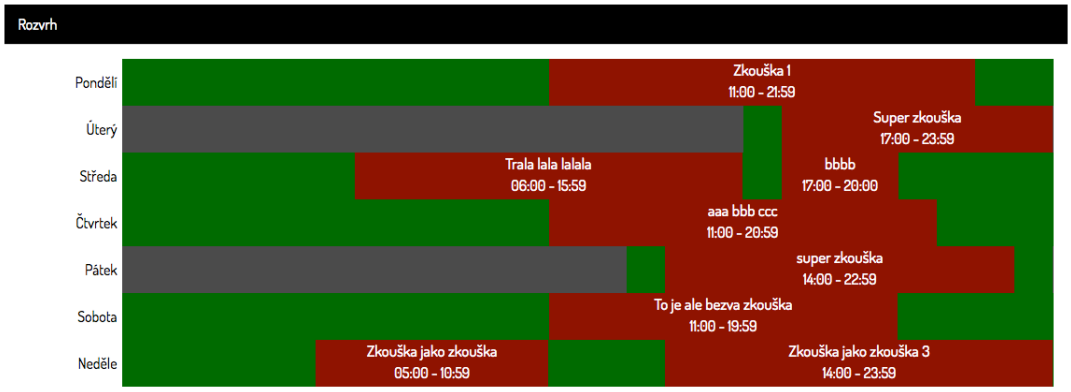
Po kliknutí na tlačítko editace je přesměrován na stránku, kde může zkušebnu editovat. Jednotlivé formuláře jsou stejné jako při vytváření profilů zkušeben, a to díky tomu, že je pro jejich zobrazení využita stejná komponenta Angularu, jen se vstupním parametrem *edit*. Komponenta se tak chová jinak. Zejména tlačítka



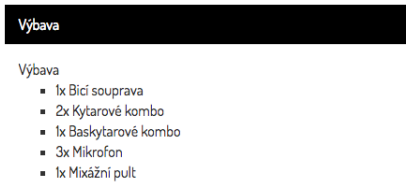
Obrázek 18: Detail zkušebny zobrazený vedle mapy

pro odeslání formuláře jsou změněna. Na rozdíl od vytvářecích formulářů, kde jsou u každého kroku tlačítka „předchozí“ a „další“, v editaci je pouze tlačítko „uložit“. Po kliknutí na toto tlačítko se změněná data odešlou na server, kde jsou následně uložena do databáze. V případě obrázků se na server odesílá pole s přidanými obrázky a se smazanými obrázky. Přidané obrázky se tak uloží stejně jako při vytváření profilu zkušebny.

Obrázek 19: Střídající se výřez fotografií zkušebny v detailu



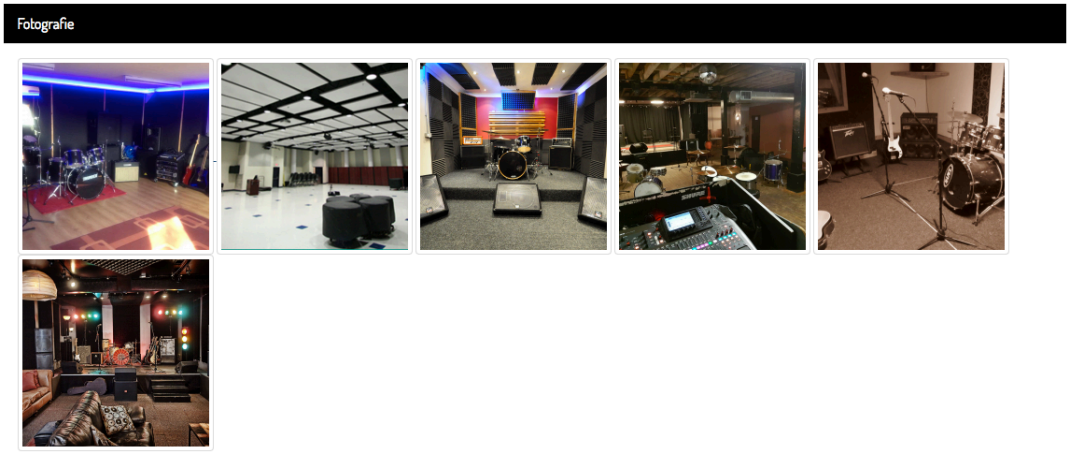
Obrázek 20: Zobrazení rozvrhu v detailu



Obrázek 21: Zobrazení vybavení zkušebny v detailu

Administrace

Velmi podobně jako seznam zkušeben uživatele vypadá administrační panel aplikace. Navíc disponuje vyhledávacím okénkem, díky kterému může administrátor vyhledávat nad zkušebnami. Toto vyhledávání je umožněno díky již zmiňované službě *ElasticSearch*.



Obrázek 22: Fotografie zkušeben zobrazené v detailu

Obrázek 23: Formulář pro odpověd na inzerát v detailu

Obrázek 24: Zobrazení zkušeben uživatele

5 Testovaní, nasazení a zabezpečení aplikace

5.1 Testování aplikace

Vedle toho, že jsem celou aplikaci ručně otestoval, je testována také automatickými testy pomocí testovacího frameworku Protractor, který testuje aplikaci v reálném prohlížeči a pracuje s ní jako skutečný uživatel (Protractor, 2017). Zatím je takto otestován pouze proces přihlašování uživatele a proces vytváření profilu zkušebny. Jednotlivé testy v Protractoru vypadají například takto:

```
1      it('should login', () => {
2          element(by.css('#username'))
3                    .sendKeys('tester@test.com');
4          element(by.css('#password'))
5                    .sendKeys('tajneheslo');
6          element(by.buttonText('Login')).click();
7          expect(page.getTitleText())
8            .toEqual('Najdi místo pro svoji hudbu.');
9      });
```

Do příslušného formuláře se tak automaticky zadají potřebné hodnoty a formulář se odešle. Pokud je po odeslání formuláře provedeno přesměrování na domovskou stránku, tedy na stránku obsahující požadovaný text, znamená to, že uživatel byl přihlášen, neboť v opačném případě by byla zobrazena chyba už na přihlašovacím formuláři.

Tento testovací framework je obsažen v Angularu 2+ a není ho třeba doinstalovávat. Testy v něm navíc fungují velmi rychle, není je většinou třeba vybavovat žádnou čekací dobou. Protractor automaticky pozná, kdy Angular přestal pracovat na překreslování stránky a na úlohách na pozadí, a sám pokračuje v testování.

5.2 Nasazení aplikace

Pro nasazení aplikace bylo potřeba upravit konfigurační soubory frameworku Laravel a Angular tak, aby byly aplikace připraveny pro produkční prostředí. Celá frontendová část aplikace se také musí přeložit, aby ji bylo možné otevřít v prohlížečích, bez spuštěného vývojového serveru. Na serveru potom musí běžet minimálně PHP 5.6, MySQL 5.7 a ElasticSearch 2.4. Pokud tyto požadavky zabezpečíme, stačí importovat jen databázi a naindexovat zkušebny z MySQL databáze do ElastiSearche.

Pro to, aby byla aplikace bezpečná, je třeba, aby serverová část aplikace nezobrazovala PHP chyby. To je zabezpečeno díky konfiguračnímu souboru Laravelu.

Aplikace je momentálně nasazená na virtuálním serveru (VPS) s 15 GB SSD diskem, 1 GB RAM a jedním vláknem procesoru Intel Xeon o frekvenci 1,7 GHz.

Adresa aplikace je nyní `https://test.hudebnizkusebny.cz`. V budoucnu však aplikace nahradí původní verzi na adrese `https://hudebnizkusebny.cz` a dalších zahraničních doménách.

5.3 Zabezpečení aplikace

HTTPS

Nejdůležitějším zabezpečením aplikace tak, aby veškerá komunikace mezi uživatelem a serverem byla soukromá, zajišťuje protokol HTTPS. Jedná se o protokol, kde jsou http požadavky a odpovědi přenášeny zašifrovaně pomocí SSL nebo TLS a identita serveru je ověřena díky certifikátu poskytnutého certifikační autoritou (Kayce, 2017).

Díky certifikační autoritě Let's Encrypt lze nyní zabezpečit jakýkoli server zcela zadarmo (Let's Encrypt, 2017). Takto je zabezpečena i tato aplikace.

Prevence SQL injection

Velmi jednoduchou technikou, jak napadnout databázi veřejné aplikace, je *SQL injection*. Útočník například změní adresu URL požadavku tak, aby na vstup zareagovala samotná databáze a udělala něco, co programátor nezamýšlel. Například pokud v aplikaci použijeme takovýto kód:

```
statement = "SELECT * FROM users
                        WHERE name ='" + userName + "';"
```

a uživatel jako *userName* zadá:

```
' or '1'='1
```

SQL dotaz bude vypadat takto:

```
SELECT * FROM users WHERE name = '' OR '1'='1';
```

Takže místo toho, aby útočník kódem získal pouze uživatele s určitým jménem, získá úplně všechny uživatele nehledě na jméno. Takto se dají skládat mnohem sofistikovanější dotazy, které například úročníkům umožní smazat databázi nebo získat obsah celé databáze včetně hesel (Kulman, 2014).

ORM Eloquent však všechny vstupy do databáze proti těmto útokům chrání, proto není možné, aby útočník získal z databáze jiná data, než která získat může.

Prevence proti XSS útokům

XSS (Cross-Site-Scriptiong) útok je technika, kdy útočník na stránku pomocí nějakého vstupu vloží svůj javascriptový kód, který se pak jinému uživateli v prohlížeči spustí a pak například odesílá jeho citlivá data nebo zobrazuje na stránce jiný obsah.

Angular 2+ však XSS útokům brání sám tak, že při jakémkoli výpisu odstraní z proměnné vše, co by mohlo způsobit potíže (Angular.io, 2017).

Prevence proti CSRF útokům

Podle J. Pejší je CSRF typ útoku na webovou aplikaci nebo službu. Zkratka CSRF (Cross-Site Request Forgery) znamená „podvržení požadavku mezi různými stránkami“. Někdy se také můžeme setkat k s dalšími termíny jako XSRF, Cross-Site Reference Forgery, Session Riding nebo Confused Deputy attacks. (Pejša, 2017)

Díky tomu, že k provedení akce autorizovaného uživatele je nutné k požadavku připojit autorizační token, který je uložen v lokálním úložišti prohlížeče (local storage), nelze za přihlášeného uživatele cokoli dělat ze stránky stažené z jiné domény, subdomény či dokonce protokolu (http, https), protože k těmto hodnotám mají přístup pouze skripty stažené z dané subdomény stejného protokolu.

6 Závěr

V rámci práce byla vytvořena webová aplikace, která splňuje funkční a nefunkční požadavky, jež byly pro práci stanoveny. Uživatelé mohou jednoduchým způsobem provést všechny úkony potřebné k nalezení vhodné zkušebny a k její inzerci. Vzhled a chování aplikace odpovídá nejmodernějším trendům v oblasti webového designu. Stejně tak technologie použité pro implementaci aplikace jsou na úrovni technologií využívaných např. největšími e-shopy u nás.

Hlavním přínosem aplikace je to, že umožňuje inzerovat a spravovat zkušebny komplexním způsobem, který neposkytoval žádný doposud používaný systém na internetu.

Uživatel, který stránku navštíví, si může zobrazit všechny zkušebny ve svém městě a mezi nimi snadno a přehledně filtrovat. Zkušebny jsou také zobrazeny na interaktivní mapě, která kapelám velmi ulehčí zvolení optimální zkušebny vzhledem k poloze. Návštěvníci mohou také zobrazovat detaily zkušeben nebo odpovídat na vybrané inzeráty pomocí vloženého kontaktního formuláře. Inzerenti zkušeben mohou po registraci nebo přihlášení do svého účtu vytvářet profily hudebních zkušeben pomocí přehledného sedmikrokového průvodce, kde mohou mj. vkládat obrázky, vytvářet rozvrhy nebo přidávat vybavení zkušebny. Zadané údaje potom mohou v příslušné sekci upravovat. Administrátor pak může všechny zkušebny upravovat nebo mazat.

Aplikace potom disponuje funkcemi pro zpětnou vazbu uživatelů, kteří například naleznou chyby v systému nebo jim chybí určitá funkcionalita. V plánu je také umožnit inzerentům přeložit aplikaci do jejich rodného jazyka výměnou za zvýhodnění v pořadí zobrazení zkušeben. Tím se aplikace může rozšířit do dalších zemí světa.

Určitě je možné aplikaci dále rozšiřovat o další funkce. V současném stavu jsou například velmi omezené možnosti rozvrhu, který by se do budoucna mohl rozrůst o nějaký systém rezervace jednorázových časů zkoušek. Stejně tak by se mohla rozrůst funkcionalita spojená s evidencí vybavení.

7 Reference

[Angular.io, 2016] ANGULAR.IO. *QUICKSTART* [online]. 2016 [cit. 2017-04-23]. Dostupné z: https://angular.io/docs/ts/latest/quickstart.html.

[Angular.io, 2017] ANGULAR.IO. *SECURITY.* [online]. 2017 [cit. 2017-05-17]. Dostupné z: https://angular.io/docs/ts/latest/guide/security.html/.

[Atlantida s.r.o., 2014] ATLANTIDA S.R.O. *Hudební zkušebny.* MUSICstage. [online]. © 2014 – 2017 [cit. 2017-01-14]. Dostupné z: http://www.musicstage.cz/hudebni-zkusebny/.

[Bean, 2015] BEAN, M. *Laravel 5 Essentials.* Birmingham: Packt Publishing, 2015. ISBN 978-1-78528-301-7.

[Berg, 2017] BERG, BASTIAAN VAN DEN. *basvandenberg/ng-select: A select component for angular (based on select2).* [online]. 2017 [cit. 2017-04-27]. Dostupné z: https://github.com/basvandenberg/ng-select.

[Boronczyk, 2016] BORONCZYK, T. *MySQL okamžitě.* Brno: Computer Press, 2016. ISBN 978-80-251-4737-5.

[Combe, 2016] COMBE OLIVER. *@ngx-translate/core: The internationalization (i18n) library for Angular 2+..* [online]. 2017 [cit. 2017-04-27]. Dostupné z: https://github.com/ngx-translate/core.

[ElasticSearch, 2017] ELASTICSERACH. *Geo Distance Query.* [online]. 2017 [cit. 2017-04-29]. Dostupné z: https://www.elastic.co/guide/en/elasticsearch/reference/5.0/query-dsl-geo-distance-query.html.

[Gersen, 2017] GERSEN, LEON. *noUiSlider: A JavaScript Range Slider.* [online]. 2017 [cit. 2017-04-29]. Dostupné z: https://refreshless.com/nouislider/.

[getbootstrap.com, 2016] GETBOOTSTRAP.COM. *CSS:Global CSS settings, fundamental HTML elements styled and enhanced with extensible classes, and an advanced grid system.* [online]. 2019 [cit. 2017-04-27]. Dostupné z: http://getbootstrap.com/css/.

[Google, 2017] GOOGLE.COM. *reCAPTCHA: Tough on bots. Easy on humans.* [online]. 2017 [cit. 2017-04-29]. Dostupné z: https://www.google.com/recaptcha/intro/invisible.html.

[Jones, 2016] JONES, Paul M.*A History Of PHP Frameworks/Library Collections: A list of the years-of-introductions of notable (to me) PHP framework and library projects.* [online]. 2016 [cit. 2017-04-23]. Dostupnéz:https://github.com/pmjones/php-history.

[JWT.io, 2016] JWT.io. *Introduction to JSON Web Tokens.* [online]. 2016 [cit. 2017-04-27]. Dostupné z: https://jwt.io/introduction/.

[Kayce, 2017] Basques, Kayce. *Why HTTPS Matters.* [online]. 2017 [cit. 2017-05-17]. Dostupné z: https://developers.google.com/web/fundamentals/security/encrypt-in-transit/why-https.

[Kulman, 2014] Kulman, Igor. *SQL Injection pre každého.* [online]. 2014 [cit. 2017-05-17]. Dostupné z: https://www.zdrojak.cz/clanky/sql-injection-pre-kazdeho/.

[Laravel.com, 2016] Laravel.com. *Eloquent: Getting Started* [online]. 2016 [cit. 2017-04-23]. Dostupné z: https://laravel.com/docs/5.1/eloquent .

[Lerner, 2013] LERNER, A. *Ng-book: the complete book on AngularJS.* London: Fullstack.io, 2013. ISBN 978-0-99134-460-4..

[Let's Encrypt, 2017] Let's Encrypt. *Let's Encrypt.* [online]. 2017 [cit. 2017-05-17]. Dostupné z: https://letsencrypt.org/.

[Müller, 2016] Müller, Sebastian. *Getting started: Let's start from zero and build an Angular 2 app with angular2-google-maps* [online]. 2016 [cit. 2017-04-27]. Dostupné z: https://angular-maps.com/docs/getting-started.html.

[O'Shannessy, 2013] O'SHANNESSY, Paul. *React* [online]. 2013 [cit. 2017-04-23] Dostupné z: https://github.com/facebook/react/tree/v0.3.0.

[PrimeTek, 2017] PrimeTek, 2017. *PRIMENG* [online]. 2016 [cit. 2017-04-27]. Dostupné z: https://www.primefaces.org/primeng/#/.

[Rehearsal Space Finder, 2017] *Rehearsal Space Finder.* [online]. 2017 [cit. 2017-01-14]. Dostupnéz:http://rehearsalspacefinder.com.

[Resig, 2009] RESIG, John. *jQuery 1.3 and the jQuery Foundation* [online]. 2009-01-14, [cit. 2009-07-10]. Dostupné z: http://blog.jquery.com/2009/01/14/jquery-13-and-the-jquery-foundation/.

[Pejša, 2017] Pejša, Jan. *Co je Cross-Site Request Forgery a jak se mu bránit.* [online]. 2017 [cit. 2017-05-17].
Dostupné z: `https://www.zdrojak.cz/clanky/co-je-cross-site-request-forgery-a-jak-se-branit/`.

[Shekhovtsov, 2017] Shekhovtsov, Dmitriy. *valor-software/ng2-file-upload* [online]. 2017 [cit. 2017-04-27].
Dostupné z: `https://github.com/valor-software/ng2-file-upload`.

[Protractor, 2017] Protractor. *Protractor: end to end testing for AngularJS.* [online]. 2017 [cit. 2017-05-17].
Dostupné z: `http://www.protractortest.org/`.

[Sklar, D.] SKLAR, D. *Learning PHP: A Gentle Introduction to the Web's Most Popular Language.* New York City: O'Reilly Media, 2016.
ISBN 978-1-49193-357-2..

[Skvorc, 2015] SKVORC, B. *The Best PHP Framework for 2015: SitePoint Survey Results.* Sitepoint. [online]. 30. 9. 2015 [cit. 2017-01-14].
Dostupné z: `https://www.sitepoint.com/best-php-framework-2015-sitepoint-survey-results/`.

[Winder, 2017] Winder, Lee. *leewinder/ng2-file-drop* [online]. 2017 [cit. 2017-04-27].
Dostupné z: `https://github.com/leewinder/ng2-file-drop`.