Vysoká škola ekonomická v Praze
Fakulta informatiky a statistiky

Diplomová práce

Bc. David Haška
2018

Vysoká škola ekonomická v Praze
Fakulta informatiky a statistiky

Katedra informačních technologií

Vytvoření realtime webové aplikace pomocí frameworků ReactJS a Laravel

Autor diplomové práce: Bc. David Haška
Vedoucí diplomové práce: Ing. et Ing. Soňa Karkošková, Ph.D.
Rok obhajoby: 2018

Čestné prohlášení

Prohlašuji, že diplomovou práci na téma

„Vytvoření realtime webové aplikace pomocí frameworků ReactJS a Laravel“

jsem vypracoval samostatně a veškerou použitou literaturu a další prameny jsem řádně označil a uvedl v přiloženém seznamu.

V Praze dne ……

.....

podpis

Abstrakt

Tato diplomová práce se zabývá vytvořením webové aplikace pro objednávání materiálu od dodavatele se zpracováním v reálném čase za pomocí frameworků ReactJS a Laravel. Pomocí analýzy existujících řešení a rešerše informací pro tvorbu takovéto aplikace je proveden návrh celé aplikace. Na základě tohoto návrhu je postupně popisován způsob implementace takovéto aplikace do reálného prostředí. Práce je rozdělena do tří základních celků. První z nich je popis metodiky práce, kde je podrobně popsáno, jaké metody jsou použity pro splnění cílů této práce. Druhý celek je teoretická část, kde jsou popsány základy webových aplikací. Webové aplikace zpracovávají a zobrazují data v reálném čase pro všechny instance této aplikace a pak také popis nástrojů, které jsou použity při tvorbě samotné aplikace, což je v tomto případě třeba framework ReactJS nebo framework Laravel. Třetí a poslední celek je praktická část, což je samotná tvorba aplikace za použití informací a nástrojů z předchozích dvou celků.

Klíčová slova

webová aplikace, realtime, reactjs, laravel, framework, php, javascript

Abstract

This diploma thesis deals with the creation of a web application for ordering material from the supplier with real time processing using the ReactJS and Laravel frameworks. Using an analysis of existing solutions and information retrieval for creating such an application, a whole application design is performed, which means designing the application architecture and designing the data processing architecture. Based on this proposal, it is gradually described how to implement such application into a real environment. The work is divided into three basic units. The first is a description of the work methodology, which describes in detail what methods are used to meet the objectives of this work. The second is the theoretical part, which is a necessary prerequisite for the practical part. Here are the basics of web applications as such, web applications that process and display real-time data for all instances of this application, and then a description of the tools that are used to create the application itself, which is the framework of ReactJS or the Laravel framework . The third and final whole is practical, which is the actual creation of the application using information and tools from the previous two units.

Keywords

web application, realtime, reactjs, laravel, framework, php, javascript

Obsah

Úvod

Ve světě vývoje moderních webových aplikací dochází k velkým změnám a jednou z nich je vývoj aplikací, které zpracovávají data včetně výsledku v reálném čase (dále jen realtime) bez nutnosti obnovovat webovou stránku ručně. Důvod je jasný a to především, že uživatelé webových aplikací chtějí vědět, co nejvíce informací, za co nejméně času – ideálně okamžitě. Tato práce se zabývá konkrétními dvěma přístupy, respektive technologiemi, které vývoj realtime webových aplikací umožňují a podporují. Je zde zpracována teoretická část, tedy popsání použitých principů a jiných technologií, které s vývojem realtime aplikací souvisí a dále praktická část, ve které se podrobně zkoumá a popisuje vývoj za použití všech znalostí a technologií vyjmenovaných v teoretické části. Výsledkem této diplomové práce je naplnění stanovených cílů. Práce se zabývá návrhem a vývojem realtime webové aplikace, která využívá frameworků Laravel a ReactJS – tedy jazyk PHP a Javascript.

Cíle práce

Hlavní cíl práce je návrh a vývoj realtime webové aplikace pomocí frameworků ReactJS a Laravel.

Pro splnění hlavního cíle práce jsou definovány následující dílčí cíle:

1. Seznámení s realtime webovými aplikacemi včetně technologií k tomu určených a vysvětlení rozdílů oproti klasickým aplikacím.
2. Seznámení s JavaScriptovým frameworkem ReactJS (frontend[1]).
3. Seznámení s PHP frameworkem Laravel (backend[2]) a tvorbou REST API.
4. Definice rámce realtime webové aplikace a návrh vývoje realtime webové aplikace.
5. Vývoj aplikace pomocí definovaného návrhu a pomocí frameworků ReactJS a Laravel.
6. Ověření funkčních požadavků implementované aplikace.

Cílová skupina

Tato práce bude sloužit především programátorům nebo projektovým manažerům či investorům, kteří chtějí a uvažují o vývoji podobné aplikace. Může také inspirovat majitele, respektive správce webových aplikací, kteří by chtěli do budoucna svoje řešení

.....

[1] frontend je klientská část aplikace, tedy ta část, kterou uživatel vidí

[2] backend je databázová část aplikace, která pracuje na pozadí a zpracovává veškerá data předaná z klientské části aplikace

modernizovat pomocí těchto nástrojů a pomocí technologie zpracování a zobrazení dat v reálném čase.

1 Rešerše

V rámci této práce je provedena rešerše akademických prací a odborné literatury, které se zabývají podobným tématem.

1.1 Akademické práce

Real-time webové aplikace (Toman, 2012) akademická práce se zabývá webovými aplikacemi, které zpracovávají data v reálném čase. Teoretický základ je dobrý, jenomže problém je v tom, že práce je z roku 2012, tudíž hodně zastaralá, za posledních 6 let se technologie a principy zpracování dat v reálném čase zlepšily. Navíc tato práce nebere v potaz konkrétní frameworky ReactJS a Laravel, které dnes patří mezi nejlepší frameworky pro frontend a backend.

Vývoj moderních webových aplikací (Nezdara, 2018) akademická práce se zabývá vývojem moderních webových aplikací za použití frameworku ReactJS a Laravel, jenomže praktická část je věnovaná vývoji běžné aplikace, nikoliv realtime, takže teorie, a hlavně použité metody jsou naprosto odlišné od těch, které budou použity v této práci.

Architektura bezserverových jednostránkových aplikací v jazyku JavaScript (Zikmund, 2016), akademická práce se zabývá architekturou aplikací napsaných ve frameworku ReactJS, ale zároveň architekturou aplikací, které jsou bezserverové, takže už z principu se zde neuvažuje žádná datová výměna, natož datová výměna v reálném čase. Nicméně jako základ pro vývoj frontend části naší aplikace je to velmi užitečné, jelikož informace jsou poměrné aktuální (z roku 2016).

Internetová obchodní galerie – kooperace e-shopů pro maloobchodní prodej (Nguyen, 2017). Akademická práce se zabývá vývojem aplikace obchodní galerie za použití frameworku ReactJS. Opět se jedná o běžnou aplikaci, nikoliv o aplikací, která by zpracovávala data v reálném čase, takže v našem případě jsou použitelné pouze základy práce s tímto frameworkem. Výhoda je velice aktuální informace vzhledem k datu publikování této práce.

Platforma Meteor jako webové prostředí pro vývoj aplikaci (Zvyagintseva, 2017), akademická práce se zabývá vývojem běžných webových aplikací za použití platformy Meteor, což je konkurence pro zvolený framework ReactJS. Teoretické základy jsou opět shodné a my můžeme z této práce využít porovnání se zvoleným frameworkem.

Framework Laravel (Fiala, 2015), tato akademická práce se zabývá frameworkem Laravel, tedy frameworkem, který bude použit pro backend a bude data zpracovávat v reálném čase. Výsledky tohoto zpracování se poté zobrazí uživateli, respektive na frontend části. Práce popisuje framework velice podrobně, takže se některé poznatky budou hodit, ale nicméně se jedná v podstatě o rozšíření dokumentace tohoto frameworku. Navíc se jedná o

zastaralou práci (rok 2015), protože od té doby framework prošel již pěti novými verzemi (dnes 5.6 a roce 2015 to byla verze 5.1). A v neposlední řadě, práce nepopisuje práci s frameworkem při kompletním procesu vývoje webové aplikace už vůbec ne realtime.

Implementace webových aplikací pomocí vývojového frameworku Laravel (Ivaška, 2017) akademická práce se již zabývá samotným vývojem webových aplikací za použití frameworku Laravel, ale pouze o běžnou aplikaci s databází. Nebere se zde v potaz oddělení frontend a backend části. A jedná se o popis vývoje běžné aplikace, takže výměna dat neprobíhá v reálném čase.

1.2 Odborná literatura

React.js Essentials (Fedosejev, 2015) je publikace, která se zabývá frameworkem React.JS a popisuje základy a principy, na kterých tento framework pracuje. Dále také popisuje jeho možnosti využití na konkrétních příkladech. Každopádně tyto příklady nepopisují implementaci s backend aplikací a už vůbec ne pomocí realtime technologií, takže tyto příklady nelze využít pro tuto práci.

Laravel 5 Essentials (Bean, 2015) je publikace ze stejné série jako publikace React.js Essentials. Skvěle popisuje tvorbu databázové aplikace na ukázkových příkladech. Techniky postupů jsou zde popsány o trochu lépe než v dokumentaci, protože vycházejí ze zkušenosti mnoha vývojářů. Tyto postupy lze využít v této práci. Ale opět se tyto postupy nevztahují na práci s frontend aplikací pomocí technologie realtime, takže toto lze použít pouze jako základ pro backend aplikace. Mimo jiné je tato publikace doporučovaná samotným tvůrcem frameworku Laravel Taylorem Otwellem.

ReactJS by Example – Building Modern Web Applications with React (Vipul, 2016) je publikace, která se zabývá tvorbou moderních webových aplikací pomocí frameworku ReactJS a jelikož je tato publikace relativně nová, tak na těchto základech lze stavět realtime aplikace. Samotné funkce se budou lišit, protože princip realtime se aplikuje jiným způsobem. Tím se tato práce zabývá.

Getting Started with Laravel 4 (Saunier, 2014) jedná se o starší publikaci, která popisuje framework Laravel, ale ve verzi 4. Jelikož tato práce používá framework Laravel verze 5, který se liší téměř ve všem, tak je pro tuto práci nepoužitelná. Každopádně pro začátečníky, kteří se chtějí učit framework je naopak vhodná, protože ve verzi 5 se nachází mnoho funkci, které na funkce z verze 4 navazují.

Socket. IO Real-Time Web Application Development (Rai, 2013) je publikace, která se zabývá a popisuje technologii realtime, která je použita v této práci. Jsou zde skvěle popsány principy a způsoby implementace této technologie. Tato publikace tvoří souhrn teoretických znalostí, které se budou odrážet při tvorbě aplikace, jak ve frontend, tak v backend části, a i

když je tato publikace z roku 2013, tak je poměrně aktuální. Jediné, co zde chybí, je vývoj těchto dvou části nezávisle na sobě a jejich vzájemná integrace, a to řeším v rámci praktické části této diplomové práce.

2 Metodika práce

Pro dosažení cíle práce je použita metodika pro vytváření nových částí informačních systémů. Jedná se o metodiku Design Science Research (DSR) od autorů Ken Peffers, Tuure Tuunanen, Marcus A. Rothenberger a Samir Chatterjee.

Metodika DSR je založena na šesti základních krocích a ty jsou následující (Peffers, Tuunanen, Rothenberger a Chatterjee, 2008, s. 1-37):

1. Identifikace problému a motivace (Identify Problem & Motivate).
2. Definice cílů řešení (Define Objectives of a Solution).
3. Návrh a vývoj (Design & Development).
4. Předvedení (Demonstration).
5. Vyhodnocení (Evaluation).
6. Komunikace (Communication).

2.1 Identifikace problému a motivace (Identify Problem & Motivate)

Rozšíření projektů, které se zabývají zpracováním dat v reálném čase stále přibývá a přibývat bude, protože každý stojí o aktuální data. Největším problémem však je, že neexistuje příliš textových zdrojů, ze kterých by se dalo čerpat při vývoji informačních systémů a aplikací s tímto principem zpracování dat. Existuje spousty komerčních aplikací, kde se ale těžko budete dostávat ke zdrojovým kódům, jako například Facebook. Dalším problémem je ten, že architektura aplikací podobného typu není nijak standardizována, protože počet způsobů, jakými je možno takovou aplikaci naprogramovat je mnoho.

2.2 Definice cílů řešení (Define Objectives of a Solution)

Cílem je vytvoření návrhu aplikace, která bude odpovídat architektuře pro zpracování dat v reálném čase. To umožní vytvářet aplikace na základě tohoto návrhu. A také díky tomu budou moci vznikat vylepšené a rozšířené verze tohoto návrhu a aplikace. Dále to přispěje k tomu, že budou aplikace navrhovány dle definovaných návrhů a bude tak pro programátory jednodušší spravovat projekty napříč organizacemi.

2.3 Návrh a vývoj (Design & Development)

První použitou metodou je metoda rešerše a analýzy dostupných informačních zdrojů o frameworku Laravel, frameworku ReactJS, vývoji aplikací, které zpracovávají data

v reálném čase a technologie s tím spojené. Pomocí této metody zajistíme základ teoretické části. Další metodou je metoda explanace, která slouží pro vysvětlení vývoje webových aplikací zpracovávající a zobrazující data v reálném čase v prostředí jednostránkových aplikací. Návrh aplikace bude postupně zpracován v těchto krocích:

1. Návrh aplikace.
2. Návrh funkčních požadavků aplikace.
3. Návrh databáze.
4. Návrh Laravel (backend) aplikace a REST API.
5. Návrh ReactJS (frontend) aplikace.

2.4 Ukázka (Demonstration)

Výsledky návrhu aplikace budou použity pro samotnou implementaci a ukázku, tedy spuštění aplikace v konkrétním prostředí, která bude splňovat funkční požadavky specifikované v návrhu aplikace včetně popisu aplikace. Nefunkční požadavky nejsou brány v potaz, protože cíl této práce není grafický návrh aplikace s ohledem třeba na „user experience“, ale cíl je návrh a vývoj aplikace za použití zmíněných technologií zpracování a zobrazení dat v reálném čase. Samozřejmosti je přívětivé a jednoduché uživatelské rozhraní, které usnadní ovládání aplikace a s tímto bude navrženo grafické uživatelské rozhraní. Zdrojový kód je součástí přílohy A: Obsah CD se zdrojovým kódem, a také veřejného repositáře na Bitbucket.org.

2.5 Vyhodnocení (Evaluation)

Způsobů, jakými je možné navrhnout tuto aplikaci, je více a já vysvětluji, popisuji pouze jeden z nich. A jelikož jediná věc, o kterou při návrhu a vývoji nějaké aplikace jde, je to, aby fungovala, tak návrh a vývoj v rámci této práce bude ověřen testem všech funkčních požadavků na tuto aplikaci.

2.6 Komunikace (Communication)

Komunikace návrhu probíhá skrze tuto práci. Práce bude po odevzdání dostupná na stránkách VŠE, NUŠL a thesis.

3 Teoretická část

Teoretická část práce popisuje, co to jsou webové aplikace, jaké konkrétní technologie lze použít a jak vlastně tyto technologie fungují.

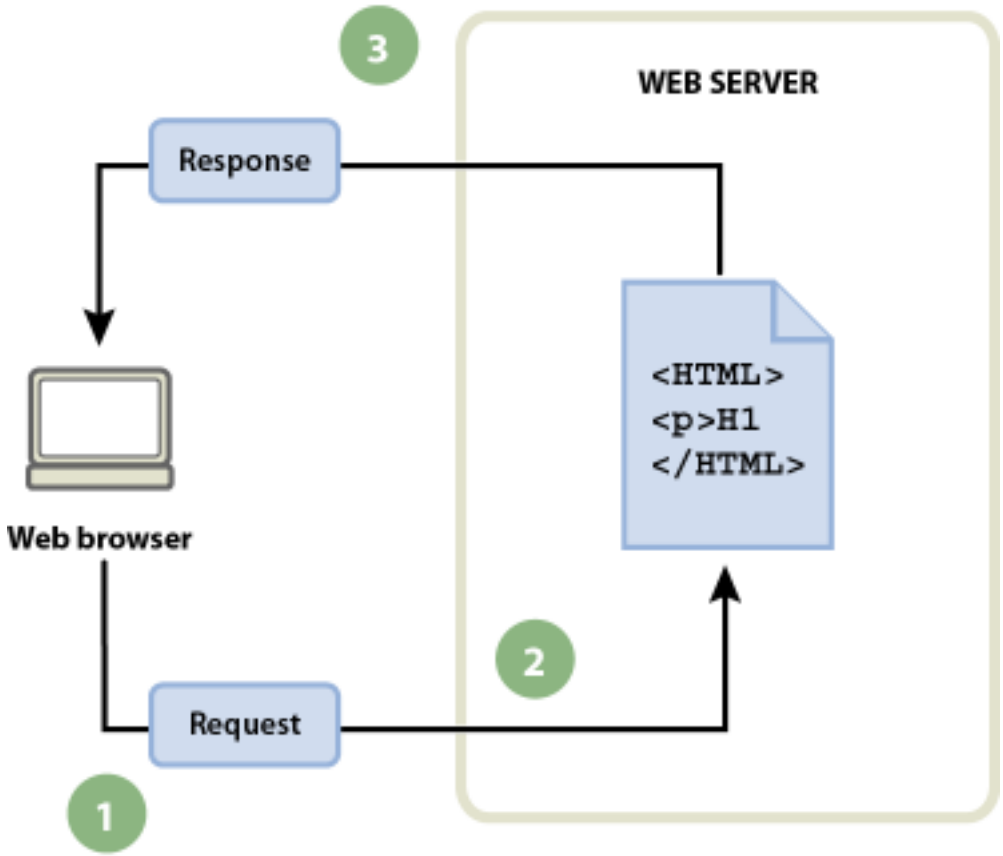
„Webová aplikace je kolekcí statických a dynamických webových stránek. Statická webová stránka se nemění, když o ni návštěvník webového místa požádá: webový server odešle stránku webovému prohlížeči, který o ni požádal, beze změny. Naproti tomu dynamickou webovou stránku server modifikuje před odesláním v prohlížeči, který o ni požádal. Proměnlivá povaha stránky je důvodem, proč se jí říká dynamická.“ (Adobe, 2017).

Statická stránka je dokument, který se vzhledem ke standardizovanému formátování webových dokumentů zobrazí v prohlížeči. Jedná se konkrétně o standard W3C[3], který využívá XML[4] pro tvorbu webových dokumentů (W3C, 2017). Zpravidla je takový dokument uložen na serveru dostupném z veřejné sítě a poté pomocí URL zobrazen uživateli v prohlížeči viz. [Obrázek 1].

.....

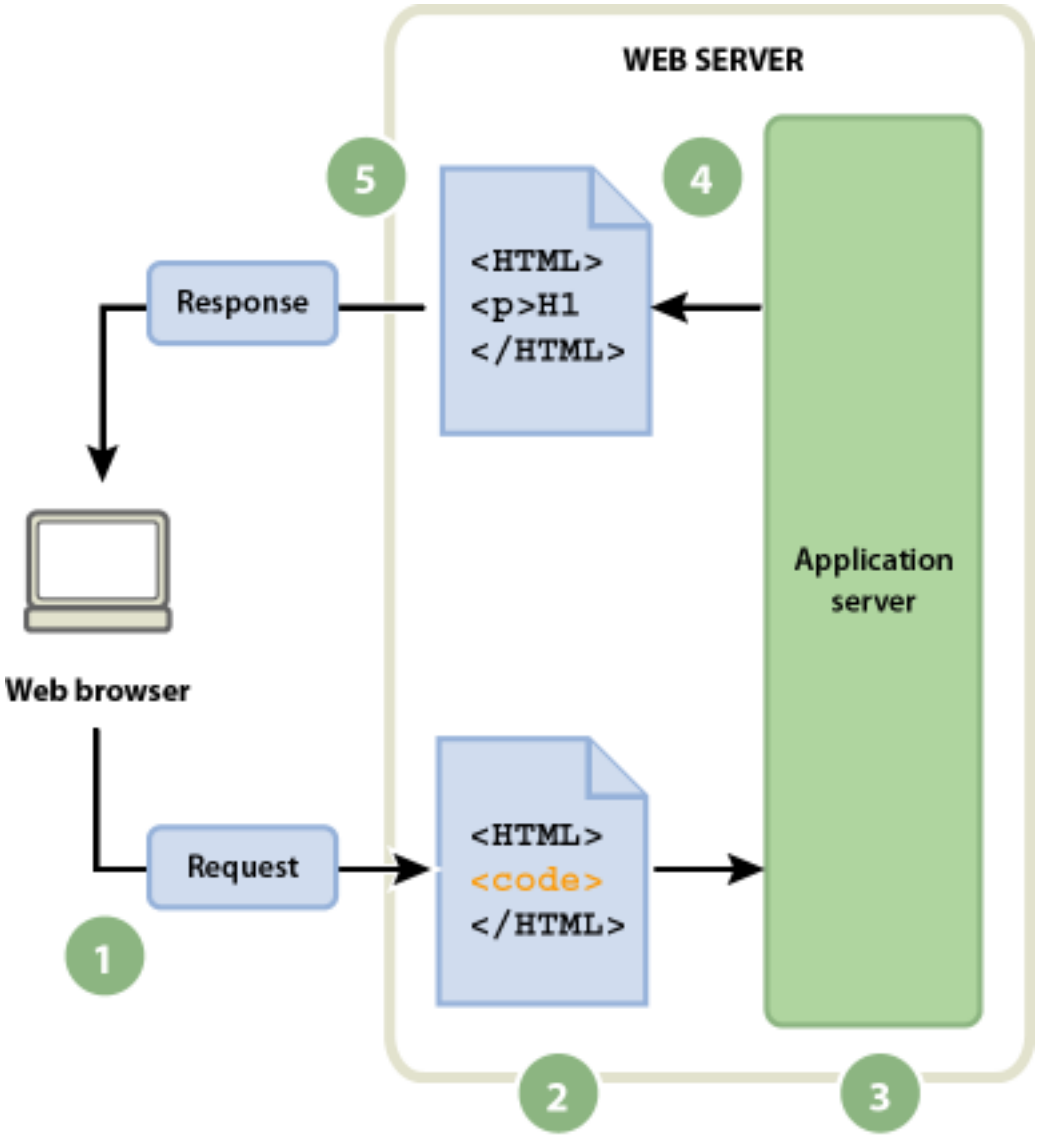
[3] W3C je zkratka pro World Wide Web Consortium, což je organizace pro standardizaci webu

[4] XML je zkratkou pro eXtensible Markup Language, což je jazyk pro tvorbu dokumentů – standardizováno organizací W3c



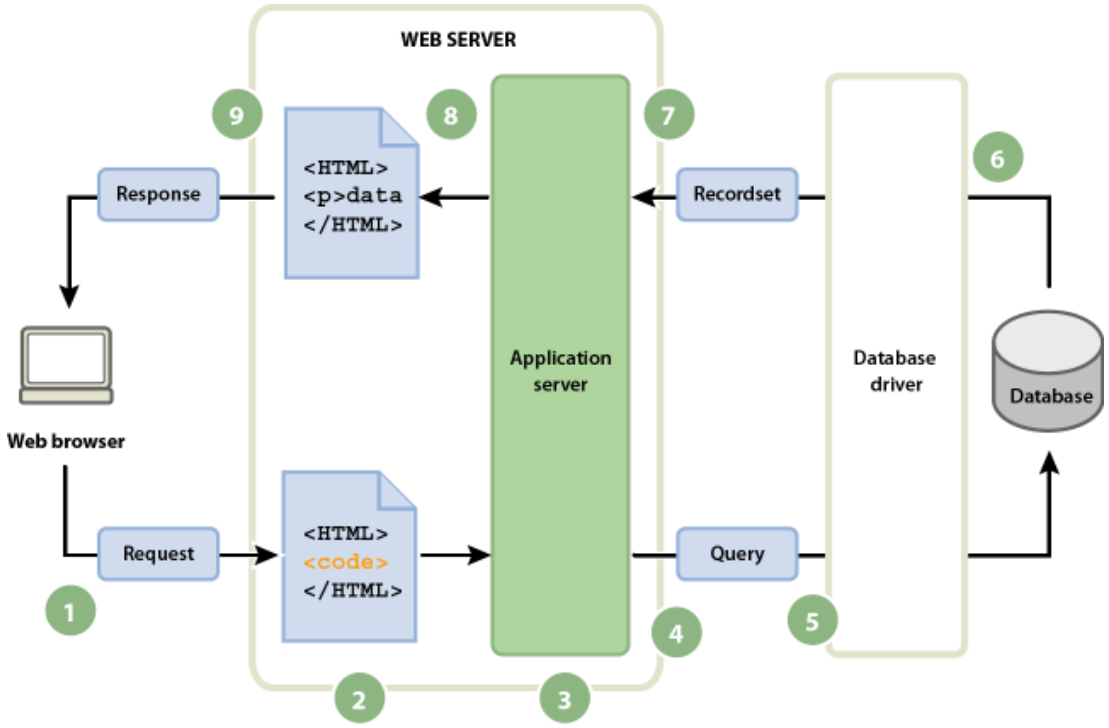
Obrázek 1 Statická stránka, zdroj (Adobe, 2017)

Dynamická stránka bez databáze funguje podobně jako statická stránka, akorát její obsah je generován aplikací přímo na serveru viz. [Obrázek 2]. Jde o to, že při každém načtení stejné URL se může zobrazit jiná stránka, v závislosti na logice konkrétní aplikace. Například, kdyby stránka vypisovala dnešní datum s konkrétním časem, tak s každým načtením se zobrazí stránka s jiným časem, což u statické stránky nelze docílit, protože statická stránka je v jedné podobě uložena na serveru a ta se zobrazuje, pokud jí administrátor nebo vývojář nezmění.



Obrázek 2 Dynamická stránka bez databáze, zdroj (Adobe, 2017)

Dynamická stránka s databází je téměř totožná jako dynamická stránka bez databáze, akorát je zde rozdíl v tom, že aplikace v případě dynamické stránky s databází načítá, upravuje a přidává data pomocí databáze, respektive databázové serveru viz. [Obrázek 3]. Této architektuře se říká třívrstvá a rozděluje se na prezenční, aplikační a datovou. Třívrstvá architektura odstraňuje nevýhodu přílišného propojení dvou na sobě nezávislých vrstev. Kvalifikace vývoje, provozu a údržby aplikace, kde pro každou vrstvu může existovat jedna oddělená osoba nebo skupina osobu, která toto zajišťuje. Tím pádem je tato architektura ideální architekturou pro tvorbu otevřených flexibilních informačních systémů, které lze pružně přizpůsobovat (Pavlík, 2018, s. 77-78).



Obrázek 3 Dynamická stránka s databází, zdroj (Adobe, 2017)

3.1 Webové aplikace

Webové aplikace fungují na principu dynamických stránek. Nyní budou popsány jednotlivé vrstvy, které jsou potřeba pro fungování webové aplikace.

3.1.1 HTML a CSS

HTML, což je zkratka HyperText Markup Language, je technologie, která umožňuje specifikovat strukturu vizuálních prvků webové aplikace, které uživatel této aplikace vidí a vede s nimi interakci pomocí webového prohlížeče (Purewal, 2014, s. 25).

Vizuálními prvky jsou myšleny tlačítka, textové pole, zaškrtávací pole, odkazy nebo také samotný obsah, respektive texty na webové stránce. Pomocí samotného HTML umíme navrhnout, jaké má aplikace rozložení, co se týče již zmíněných prvků, ale již tato technologie neumí popsat, jak tyto prvky a celá webová stránka má vypadat. K tomu pomůže druhá technologie zvaná CSS.

CSS, což je zkratka Cascading Style Sheet, je technologie, která umožňuje pomocí různých pravidel specifikovat, jak se má obsah webové stránky včetně elementů zobrazit uživateli (Duckett, 2011, s. 226). Například lze specifikovat jednotnou barvu pozadí pod všemi prvky, ale zároveň pomocí identifikátorů lze měnit pozadí pouze jednotlivým prvkům.

Kromě barev lze také měnit typy písma, zarovnání textu nebo třeba velikost mezer mezi jednotlivými prvky.

3.1.2 JavaScript

HTML a CSS umožňuje specifikovat co a jakým způsobem se mají na webové stránce zobrazit všechny prvky, ale už to neumožňuje definovat, jak tyto prvky reagují při interakci s uživatelem. Kdybychom si takovou stránku zkusili spustit, tak zjistíme, že třeba při kliknutí na tlačítko, se na webové stránce vůbec nic nestane. K tomu pomůže následující technologie, která se nazývá JavaScript (dále jen JS). Výhodou je, že tato technologie je dnes podporovaná všemi prohlížeči.

JavaScript je technologie, která umožňuje definovat, jak se mají prvky na webové stránce chovat a jak mají reagovat při interakci s uživatelem (Purewal, 2014, s. 27). Jak již bylo zmíněno, tak například při kliknutí na tlačítko. Můžeme také říct, že pomocí HTML a CSS vznikají statické webové stránky, ale když se k tomu přidá technologie JS, tak vznikají dynamické webové stránky, což je synonymum pro webové aplikace. Webová aplikace je totiž webová stránka, která má definované chování, ať už při interakci s uživatelem, či nikoliv.

3.1.3 PHP

Aplikace bude zpracovávat data, ale jelikož technologie HTML, CSS a JS (dále jen frontend) nemají žádnou nativní podporu pro práci s databázemi, tak je potřeba využít mezivrstvy, která aplikaci tyto data předá, respektive mezivrstva, které data bude aplikace předávat.

K tomu poslouží serverová technologie zvaná PHP, což je zkratka pro „PHP (Personal Home Page): Hypertext Preprocessor“. Jedná se o technologii skriptovacího jazyka na straně serveru, a to umožňuje práci s databází, konkrétně MySQL, právě pomocí skriptů. Nespornou výhodou této technologie, že stejně jako frontend je podporován všemi operačními systémy, které mohou být nainstalovány na serveru jako například Windows nebo Linux (Ponkrác, 2007, s. 18-19).

3.1.4 MySQL

MySQL je tedy systém řízení báze dat, který pracuje na modelu relačních databází. Komunikace s touto databází probíhá pomocí jazyka SQL[5], což je zkratka pro strukturovaný dotazovací jazyk, který je dnes standardem většiny systémů řízení báze dat (Ponkrác, 2007, s. 23).

.....

[5] SQL je zkratka pro Structured Query Language, v překladu tedy strukturovaný dotazovací jazyk

3.2 Realtime webové aplikace

Realtime webové aplikace se od těch klasickým liší tím, že nemusíme znovu stránku načítat, takže data se aktualizují na základě obousměrného síťového spojení mezi serverem a webovým prohlížečem. Takovému spojení říkáme fullduplex (Nyklíček, 2013, s. 15).

3.2.1 Protokol Websocket

Jedná se o komunikační spojení, navázané mezi dvěma procesy, pomocí IP technologie. Socket je popsán dvojicí IP adres, protokolem a portem, který je pro komunikaci použit. Služby na vyšší úrovni tento socket používají většinou k obousměrnému přenášení dat.

Web Sockets je obdoba této technologie, přenesená do světa webových aplikací. Pomocí WebSockets můžeme napsat webovou aplikaci, kde aplikace v prohlížeči může navázat obousměrné spojení se serverem, a po tomto spojení si mohou vyměňovat informace v reálném čase. Není tedy nutné používat techniku, někdy zvanou „heartbeat“, kdy se klient v pravidelných intervalech dotazoval serveru „Máš pro mne něco?“ (a server většinu času odpovídal „Ne“). S Web Sockets je mnohem snazší vytvářet realtime webové aplikace, jako jsou online chaty či jiné služby.

Web Sockets vychází ze starších technologií, jako AJAX (asynchronní jednorázový požadavek od klienta na server přes HTTP protokol) či Comet (dlouhodobé vyhrazené HTTP spojení) a nabízí programátorům to, co tyto technologie nedokázaly. Jednoduché rozhraní pro navázání spojení a vzájemnou výměnu zpráv mezi klientem a serverem. Na straně klienta jsou WebSockets implementovány v JavaScriptu jako třída WebSocket. Programátor může vytvořit instanci této třídy, a tím navázat spojení se serverem (pokud server tuto technologii podporuje).

Zprávy nejsou u WebSockets žádné sofistikované objekty – jde o prosté řetězce. Objekt WebSockets nabízí tři události, nazvané onopen, onmessage a onclose.

Událost onopen je volaná při otevření spojení. Může sloužit jako příznak toho, že již lze posílat zprávy. Událost onclose zase oznamuje uzavření spojení. Pro vlastní výměnu dat slouží událost onmessage. Jak už název napovídá, je zavolána ve chvíli, kdy ze serveru přijde zpráva. Tělo zprávy je předáno ve vlastnosti data předané události.

Posílání zpráv má na starosti metoda send(). Jejím parametrem je řetězec, který má být poslán. Na konci běhu aplikace je možno spojení uzavřít, k čemuž slouží metoda disconnect().

Na straně serveru je zapotřebí víc než běžný HTTP server – je zapotřebí takový, který podporuje technologii WebSockets (Malý, 2009, online).

3.2.2 NodeJS

Node.js je událostmi řízený Framework pro engine V8. V8 engine je open source high-performance JavaScript a WebAssembly engine, napsán v C++. Tento engine rozšířený o pár funkcí umožňuje prováděným skriptům přistupovat k souborům či síťovým funkcím. V praxi to znamená, že můžeme vytvořit server, který „naslouchá“ na určeném portu téměř stejným způsobem, jakým vytváříme například obslužné metody pro události v prohlížeči.

Node.js je stále populárnější, i když jej nelze rozhodně považovat za nějakou vyzrálou technologii vhodnou pro podnikové nasazení. Jeho výhodou je, že je malý, snadno použitelný, využívá jazyk, který je webovým vývojářům známý, aplikace v něm napsané lze snadno škálovat a poměrně rychle v něm napíšete např. obslužný server pro WebSockets.

Node.js nabízí, na rozdíl od modelu „co spojení, to vlákno“, model založený na událostech a jejich asynchronním zpracování. Je v něm tedy mnohem jednodušší psát vlastní servery, které se neblokuji než např. se o totéž pokoušet v PHP. Svým zaměřením se řadí někam k frameworkům, jako je třeba EventMachine v Ruby. Existují pro něj i nadstavby, které umožní soustředit se na vlastní obsluhu požadavků, jako je například Sinatrou inspirovaný ExpressJS nebo Laravelu blízký Laravel Echo Server (Malý, 2010, online).

Je implementován ve WebOS. Je dostupný pro .NET. Vznikají pro něj nové moduly v C++.

3.2.3 Socket.io

Je důležité poskytovat uživatelům ve vaší webové aplikaci včasnou zpětnou vazbu. Všechno začalo se zavedením XMLHttpRequest od společnosti Microsoft, které se stalo tím, co nyní známe jako AJAX. AJAX long-polling býval standardním způsobem, jak načítat data odeslaná serverem pro aplikaci, ačkoli to nebylo nejideálnější řešení. Dlouhodobá volba zahrnuje zasílání periodických požadavků HTTP na data, zavádění latence a zvýšení zatížení serveru.

Normalizované webové servery IETF v roce 2011 poskytují vývojářům možnost odesílat a přijímat data prostřednictvím zásuvky TCP. Všechny hlavní prohlížeče začaly rozšiřovat podporu standardu a vývojáři ji začali používat ve svých projektech.

Socket.IO je obousměrná komunikační vrstva založená na událostech pro webové aplikace v reálném čase postavené na motoru Engine.IO. Odhaluje mnoho transportů, včetně AJAX long-polling a WebSockets, do jediného rozhraní API. Umožňuje vývojářům odesílat a přijímat data bez obav o kompatibilitě mezi prohlížeči.

První hlavní vydání

Socket.IO konečně dosáhla verze 1.0 dne 28. května 2014. Projekt Socket.IO obsahoval dvě části před 1.0: implementace zpracování dopravy a API na vysoké úrovni. Manipulace s dopravou byla přesunuta do samostatného rámcového agnostického projektu: Engine.IO. To umožňuje dalším vývojářům vytvářet nové rozhraní API a projekty pro web v reálném čase, aniž by objevili kolo.

Kromě architektonických změn přináší Socket.IO 1.0 mnoho změn orientovaných na uživatele, včetně:

- Vylepšená podpora pro horizontální škálování.
- Ve výchozím nastavení je v konzole odebrána chybová zpráva.
- Podpora datových toků Node.js prostřednictvím modulu socket.io-stream.

V tomto článku se rychle podíváme na to, jak může Socket.io používat k odesílání a přijímání dat v reálném čase. Socket.IO mimo jiné poskytuje součásti na straně serveru i na straně klienta s podobnými rozhraními API.

Serverová část (server-side)

Na straně serveru pracuje Socket.IO přidáním posluchačů událostí do instance serveru http.Server. Chcete-li přidat podporu Socket.IO instanci http.Server, měli byste napsat tento kód [Obrázek 4].

```
var server = require("net").createServer();
var io = require("socket.io")(server);

var handleClient = function (socket) {
    // we've got a client connection
    socket.emit("tweet", {user: "nodesource", text: "Hello, world!"});
};

io.on("connection", handleClient);

server.listen(8080);
```

Obrázek 4 Socket.io - přidání podpory pro http.Server, zdroj (autor)

S touto všestranností je možné připojit server Socket.IO k jiným rámcům HTTP. Je-li potřeba například použít Socket.IO s Express, používá se tento kód [Obrázek 5].

```
var app = require("express");
var server = require("http").Server(app);
var io = require("socket.io")(server);

io.on("connection", handleClient);

app.listen(8080);
It's also possible to use Socket.IO with Hapi:

var server = require("hapi").createServer(8080);
var io = require("socket.io")(server.listener);

io.on("connection", handleClient);

server.start();
```

Obrázek 5 Socket.io - přidání podpory pro http.Server pomocí knihovny Express, zdroj (autor)

Socket.IO je kompatibilní s většinou rámců, které vystavují jejich instanci http.Server. Více informací lze nalézt v oficiální dokumentaci na stránkách https://socket.io/.

3.2.4 Redis

Redis je open source (BSD licence) uložištěm pro datové struktury, nejčastěji používané jako databáze, mezipamět (cache) a zprostředkovatelem zpráv. Podporuje spousty variací struktur dat jako například textové řetězce, listy, sady a tak dále (Nelson, 2016, s. 1-14). Ale jinak se jedná v podstatě o normální databázi, kde se pod unikátní klíč ukládá nějaká hodnota, akorát může být variací více struktur, nikoliv pouze jedné hodnoty. Redis je napsán v programovacím jazyce C, jedná se tedy o kompilovaný jazyk, a proto je databáze velice rychlá. Jedná se o NoSQL databázi. V současné době je využívána technologickými společnostmi, jako je GitHub, Pinterest, StackOverflow nebo Flickr. Je velice přívětivá vzhledem k vývojářům, protože je podporována mnoha programovacími jazyky jako třeba PHP, JavaScript, Java, C#, C++ nebo Objective-C.

Klientská část (client-side)

Globální proměnná soketu je objekt typu EventEmitter. Můžeme připojit posluchače, když jsme se připojili k serveru takhle [Obrázek 6].

```
socket.on("connect", function () {
    console.log("Connected!");
});
```

Obrázek 6 Socket.io - připojení posluchače (listenera), zdroj (autor)

Vzhledem k tomu, že serverový i klientský objekt Socket fungují jako EventEmitters, můžete události vysílat a poslouchat obousměrně. Například můžeme na serveru vysílat událost "tweet" a naslouchat na straně klienta [Obrázek 7].

```
io.on("connection", function (socket) {
    var tweet = {user: "nodesource", text: "Hello, world!"};

    // to make things interesting, have it send every second
    var interval = setInterval(function () {
        socket.emit("tweet", tweet);
    }, 1000);

    socket.on("disconnect", function () {
        clearInterval(interval);
    });
});
```

Obrázek 7 Socket.io - vysílání zprávy, zdroj (autor)

Když je potřeba zpracovávat data v prohlížeči, musí aplikace v kódu naslouchat událost "tweet" [Obrázek 8].

```
socket.on("tweet", function(tweet) {
    // todo: add the tweet as a DOM node

    console.log("tweet from", tweet.username);
    console.log("contents:", tweet.text);
});
```

Obrázek 8 Socket.io - poslouchání zprávy, zdroj (autor)

Aplikace může odeslat libovolný JSON serialisable objekt na server a ze serveru. To zahrnuje řetězce, čísla, pole a boolean (hodnota true nebo false, respektive pravda nebo nepravda). Může také odeslat Node.js Buffer objekty začínající Socket.io 1.0.

Pokud by aplikace měla odesílat zprávy na určitém kanále přímo z prohlížeče a nechat je zpracovávat, musel by být na straně serveru napsán následující kód [Obrázek 9].

```
var tweet = {user: "nodesource", text: "Hello, world!"};
socket.emit("tweet", tweet);
```

Obrázek 9 Socket.io - odesílání zprávy přímo z prohlížeče, zdroj (autor)

3.3 Framework ReactJS

V oblasti webu a webových aplikací je React (někdy React.js nebo ReactJS) knihovnou JavaScript pro vytváření uživatelských rozhraní. Je udržována společností Facebook, Instagram a komunitou jednotlivých vývojářů a korporací.

React lze použít při vývoji jednostránkových aplikací a mobilních aplikací. Cílem je především poskytnout rychlost, jednoduchost a škálovatelnost. Jako knihovna uživatelského rozhraní se React často používá ve spojení s dalšími knihovnami, jako je Redux nebo Axios.

3.3.1 Redux

Redux je předvídatelný stavový kontejner pro aplikace napsané v programovacím jazyce JavaScript. Pomáhá programátorům psát aplikace, které běží v různých prostředích (klient, server a nativní) a jsou předvídatelné a snadno testovatelné. Kromě toho nabízí Redux vývojářům skvělé možnosti, jako je například úprava kódu v reálném čase, takže všechny provedené změny v editoru jsou ihned vidět v prohlížeči nebo také time traveling

debugging[6]. Redux lze použít společně s kteroukoliv knihovnou, která slouží pro zobrazení informací uživateli, například Angular a v našem případě React (Bugl, 2017, s. 1-49).

3.3.2 Axios

Axios je knihovna nebo Promise-based[7] HTTP klient určený pro programovací jazyk JavaScript, kterým mohou být psaný frontend aplikace, jako v našem případě, nebo také backend aplikace pomocí Node.js technologie. Pomocí Axios klienta je možné jednoduše odesílat asynchronní HTTP požadavky na server nebo na REST službu a vytvářet tak tzv. CRUD operace. Knihovna se může používat samostatně v čistém JavaScript kódu nebo za použitích jiných knihoven či frameworku jako třeba React (Passaglia, 2017).

3.4 Framework Laravel

Laravel je open source PHP framework pro webové aplikace vyvinutý programátorem Taylorem Otwellem. První vydání se datuje k únoru 2012. Jedná se o framework poskytovaný zdarma jako open source projekt pod licencí MIT. Laravel využívá softwarové architektury MVC, což je zkratka pro model-view-controler architekturu. Model = aplikační data a funkce, View = prezentace dat například v HTML, Controller = spravuje interakce mezi uživatelem modelem a pohledem. Laravel je význačný především tím, že je optimalizovaný pro reálný svět. Tedy disponuje často užívanými procedurami, které jsou nutné při vývoji webové aplikace.

Důvod, proč je Laravel v posledních letech tak oblíbeným PHP frameworkem, je ten, že se opírá o frameworkové velikány a podpůrné systémy jako jsou: Symfony, Composer, Eloquent ORM a Blade. Mimo jiné také disponuje ověřenými postupy inspirovanými technologiemi jako je Ruby on Rails, ASP.NET MVC a Sinatra.

Základním stavebním kamenem Laravelu je Symfony, který poskytuje moduly Browserkit, Console, Debug a FileSystem. Další důležitou komponentou je Composer (PHP dependencies manager), který deklaruje vztahy v JSON souboru a umožňuje migrace projektů. V neposlední řadě zde figuruje také velmi důležitý Eloquent ORM (object relation mapper), který byl speciálně vyvinut pro framework Laravel, ale je možné ho využít i mimo framework. Využívá návrhového vzoru ActiveRecord, který zajišťuje Insert, Update, Delete na databázi. Dokáže spravovat tabulkové vztahy 1:1, 1:n nebo n:m. Poslední důležitou

.....

[6] Time traveling debugging je způsob ladění kódu, při kterém je možno procházet zdrojový kód tak, jak byl prováděn v čase

[7] Promise je pattern nebo vzor, který umožňuje programovat asynchronně, určovat pořadí, v jakém jsou obslužné návratové funkce prováděny, a dokonce definovat více podmínek, které musí být současně splněny pro spuštění těchto návratových funkcí.

součástí Laravelu je komponenta Blade, což je šablonovací systém, který spravuje veškeré pohledy s příponou .blade.php.

3.4.1 API

Lze také říct, že jde o celou řadu procedur, funkcí a tříd, které může programátor využívat. API zároveň určuje, jakým způsobem jsou funkce volány.

API, respektive Application Programming Interface znamená programovací rozhraní pro aplikace. Rozhraní API může poskytnout uživatelům, partnerům nebo vývojářům třetích stran bod pro přístup k datům a službám pro rychlé vytváření aplikací, jako jsou aplikace pro mobilní telefony atd. API služby Facebook, Twitter nebo Instagram jsou slavnými příklady. K dispozici jsou rozhraní. které jsou otevřeny všem vývojářům. Ty jsou otevřené pouze pro partnery, anebo které jsou přístupné pouze interním uživatelům, což může pomáhat lépe provozovat firmu nebo usnadnit spolupráci mezi týmy.

API je tedy v podstatě smlouva. Jakmile je taková smlouva uzavřena, vývojář je vyzván k použití API, protože vědí, že se na ně mohou spolehnout. Smlouva zvyšuje důvěru, což zvyšuje její využití. Smlouva rovněž činí propojení mezi poskytovatelem a spotřebitelem mnohem efektivnější, protože rozhraní je zdokumentováno, konzistentní a předvídatelné (Jackobson, 2012, s. 4-8).

3.4.2 REST API

REST API nebo REST znamená Representational State Transfer a představuje architektonický styl pro síťovou komunikaci mezi aplikacemi, který se spoléhá na protokol bez stavový protokol (obvykle HTTP) pro interakci. V REST webové službě lze pro provádění operací CRUD použít metody HTTP jako GET, POST, PUT, PATCH a DELETE. REST je velmi jednoduchý v porovnání s jinými metodami jako SOAP, CORBA, WSDL atd. REST je, na rozdíl od známějších XML-RPC či SOAP, orientován datově, nikoli procedurálně. Webové služby definují vzdálené procedury a protokol pro jejich volání, REST určuje, jak se přistupuje k datům (Massé, 2012, s. 5-7).

Rozhraní REST je použitelné pro jednotný a snadný přístup ke zdrojům (resources). Zdrojem mohou být data, stejně jako stavy aplikace (pokud je lze popsat konkrétními daty). Všechny zdroje mají vlastní identifikátor URI a REST definuje čtyři základní metody pro přístup k nim. V nástrojích REST API používáme slovesa HTTP jako akce a koncové body jsou zdroji, na něž se konkrétní akce vztahuje. Jak již bylo zmíněno, používají se následující HTTP metody s následujícími významy:

- GET (Retrieve resource).
- POST (vytvořit zdroj).
- PUT (upravit zdroj).

- PATCH (částečně upravit zdroj).
- DELETE (smazat zdroj).

Po každém zavolání akce je potřeba zobrazit stav zobrazení zdroje, a to pomocí následujících stavových kódů:

- 200 OK.
- 201 Created.
- 304 Not Modified.
- 400 Bad Request.
- 401 Unauthorized.
- 403 Forbidden.
- 404 Not Found.
- 422 Unprocessable Entity.
- 500 Internal Server Error.

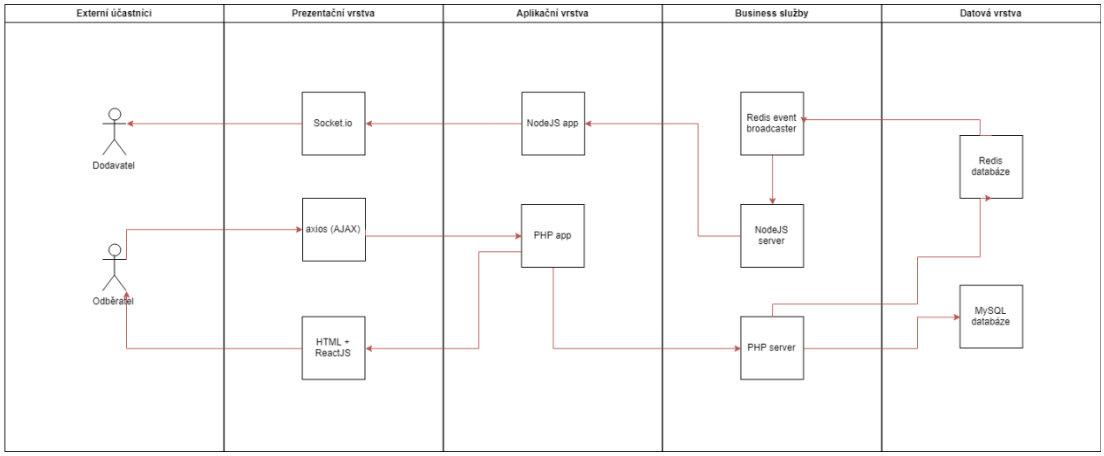
4 Praktická část

Poslední úsek práce se zabývá praktickou části této práce. Praktická část zahrnuje návrh, vývoj, implementací naprogramované aplikace a testování aplikace vzhledem k návrhu a vzhledem k požadavkům v něm navržené.

4.1 Návrh

Tato část se zabývá návrhem celé aplikace včetně architektury jak backend části včetně databáze a REST API, tak frontend části.

Architektura celé aplikace je založena na třívrstvé architektuře s tím, že na aplikační vrstvě se nachází více aplikací, které tuto vrstvu obsluhují viz. [Obrázek 10]. Na tomto obrázku můžeme vidět datovou vrstvu v podobě MySQL databázového serveru, která komunikuje pouze s aplikací napsanou v programovacím jazyce PHP – uloženo na PHP serveru (backend a API). Jedná so o požadavky na databázi, a to konkrétně načtení nebo přidání dat. Pokud zákazník načte webovou aplikaci, tak se mu zobrazí dynamická stránka spolu se skripty napsané v JavaScript (ReactJS Framework), což umožňuje pomocí asynchronních požadavků tuto stránku stále upravovat. V průběhu práce s aplikací se tedy pomocí těchto asynchronních požadavků zobrazují data. Po přidání dat se do práce zapojuje NodeJS server, který pomocí fronty v databázi Redis postupně zpracovává všechny požadavky a rozesílá zpět všem, kteří si aktuálně prohlíží aplikaci. Na základě aktuální pozice se požadavky dále zpracují. Například zákazník zadá objednávku a to znamená, že se pomocí asynchronního požadavku tzv. AJAX požadavku odešle požadavek spolu s daty na backend (API) aplikace. Backend objednávku uloží do databáze a pomocí NodeJS serveru a Redis databáze rozešle informaci o úspěšném přidání zpět všem uživatelům včetně dodavatele. Na základě logiky aplikace v ReactJS se poté informace o úspěšném přidání zobrazí pouze zákazníkovi, který tuto objednávku odeslal a zároveň dodavateli, že má novou objednávku v systému. Celý model je založen na upravené notaci podle (EA)2 Enterprise Architecture Modeling Framework (http://www.ea2.us/EA2_SampleDiagrams/ApplicationStructureDiagram.html).



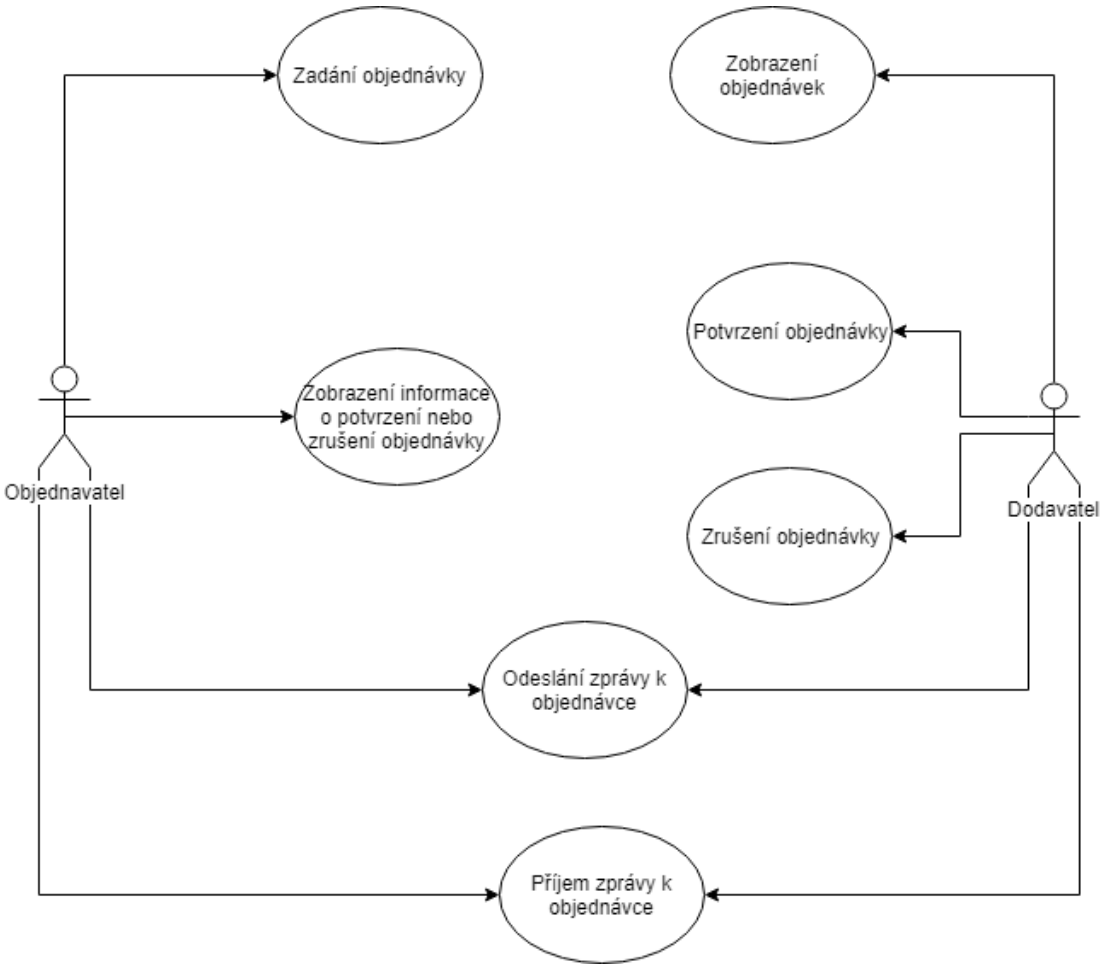
Obrázek 10 Architektura aplikace, zdroj (autor)

4.1.1 Popis aplikace

Jedná se o ukázkovou aplikaci, která bude propojovat dvě firmy nebo i více firem, kde jedna firma (dále jen firma A) se zabývá prodejem výrobku, který je závislý na odběru materiálu od firmy druhé (dále jen firma B). Princip aplikace je ten, že jakmile firmě A dojde materiál, tak pomocí aplikace tento materiál od firmy B pomocí aplikace objedná. Firmě B se v aplikaci ihned po odeslání objednávky objeví tato objednávka(y) s tím, že na to může ihned reagovat a tím se liší od běžné aplikace, kde je potřeba znovu načíst stránku (aplikaci). Další možností bude okno pro výměnu zpráv (též možnost aplikace zpracování a zobrazení dat v reálném čase) a v této aplikaci se bude jednat o možnost upřesnění firmy B k objednávce firmy B.

4.1.2 Funkční požadavky aplikace

Jedná se požadavky, které musí aplikace splňovat po její implementaci viz. UML use case diagram [Obrázek 11].



Obrázek 11 UML Use case diagram, zdroj (autor)

Zadání objednávky objednavatelem

Aplikace umí zadat, uložit a zpracovat objednávku (např. materiálu) objednavatelem na konkrétního dodavatele.

Zobrazení objednávek u dodavatele

Aplikace umí zobrazit u dodavatele všechny zadané objednávky od všech objednavatelů v chronologickém pořadí podle data odeslání.

Potvrzení objednávky dodavatelem

Aplikace umí na straně dodavatele objednávku potvrdit. To znamená, že objednávka bude zpracována a odeslána objednavateli.

Zrušení objednávky dodavatelem

Aplikace umí na straně dodavatele objednávku zrušit. To znamená, že objednávka nebude zpracována a odeslána objednavateli.

Zobrazení informace o potvrzení nebo zrušení objednávky

Aplikace umí zobrazit informaci na straně objednavatele, u něhož je objednávka potvrzena nebo zrušena.

Odeslání zprávy k objednávce (objednavatelem i dodavatelem)

Aplikace umí odeslat zprávu ke konkrétní objednávce.

Příjem zprávy k objednávce (objednavatelem i dodavatelem)

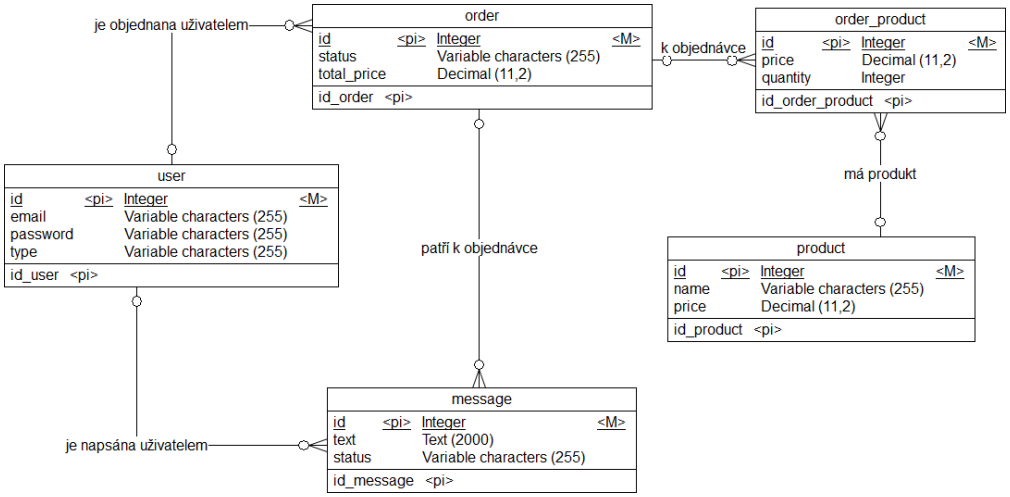
Aplikace umí zobrazit odeslané zprávy ke konkrétní objednávce.

4.1.3 Databáze

Primární úkol navržené aplikace je práce s daty, takže je potřeba uložiště dat, odkud tyto data budeme brát a také kam data budeme ukládat. Pro práci s aplikací, která zpracovává data v reálném čase není třeba nějaké speciální databáze. Pro tuto aplikaci jsem zvolil databázový server MySQL je nejrozšířenější a také zdarma, což jsou dva hlavní důvody výběru právě tohoto systému řízení báze dat.

Návrh databáze vychází z analýzy případu užití a celkové architektury aplikace. Abychom mohli ukládat vše v rámci všech procesů, je potřeba vytvořit v databázi celkem 5 tabulek, které toto umožní.

První tabulkou, respektive entitou je „user“, tedy uživatelé. V rámci konceptuálního modelu [Obrázek 12] se jedná o entitu, za kterou se skrývá kterýkoliv uživatel, který se přihlásí do aplikace. A jelikož se jedná o aplikaci, ve které figurují dvě strany, tedy dodavatel a odběratel, tak jednoduše tohoto zákazníka odlišíme textovým atributem, který může nabývat dvou hodnot, a to konkrétně „customer“ jako odběratel, respektive zákazník a „supplier“ jako dodavatel. Toto umožní pomocí API načíst informace a dle toho zobrazit frontend část aplikace.



Obrázek 12 Konceptuální model návrhu databáze, zdroj (autor)

Další entitou je „order“, tedy objednávky. V rámci konceptuálního modelu [Obrázek 12] se jedná o entitu, která ukládá informace o odeslané objednávce odběratelem, tedy kdo vlastně tuto objednávku odeslal, celková cena, ale také informaci o jejím stavu, protože v rámci navržených funkcí je nutnost mít možnost měnit její stav dodavatelem.

Další entitou je „product“, tedy produkty. V rámci konceptuálního modelu se jedná o entitu, která ukládá informace o produktech např. jejich cenách. Každý produkt má identifikátor, který je použit v další tabulce „order_product“, která uchovává informace o tom, jaké produkty byly objednány v konkrétní objednávce.

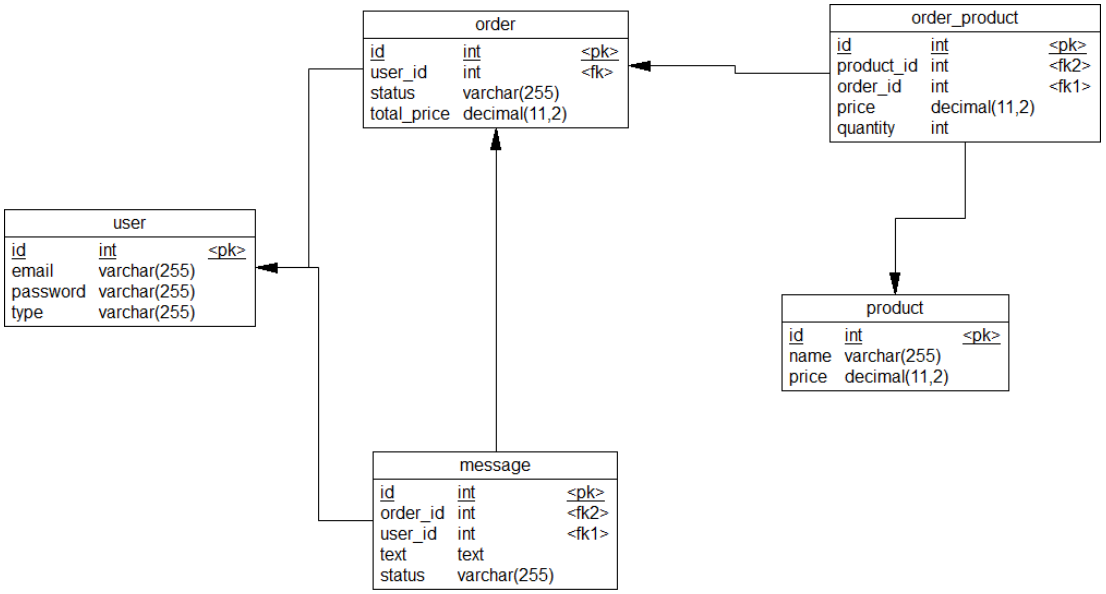
Poslední entitou je „message“, která reprezentuje zprávy odeslané, jak odběratelem, tak dodavatelem ke konkrétní objednávce. Zpráva má atribut „stav“, který může nabývat dvou hodnot a to konkrétně „new“ jako nová zpráva a „read“ jako zpráva přečtená.

Pro vytvoření databáze v naší databázi řízení dat je nutno převést konceptuální model do modelu fyzického. Většina CASE nástrojů tento převod provede již za nás, ale je třeba znát, co se při tomto procesu děje v případě nutností nějaké změny. Aby nebyly žádné složené atributy, tak je třeba všechny složené atributu rozložit do složek. Každý atribut musí mít přiřazený vhodný datový typ. Všechny entitní typy musí obsahovat primární klíče, což umožní propojovat tabulky na základě cizích klíčů. Do samostatné tabulky musí být připojeny každé samostatné entitní typy. Je třeba správně zvolit, respektive rozhodnout o dědičností jednotlivých entitních typů, jelikož máme 3 možnosti (Palovská, 2011, online).

1. **Absorpce do nadtypu** – bude jedna tabulka, ve které bude vše.

2. **Rozdělení do podtypů** – Každý podtyp bude tvořit jednu tabulku, ve které bude všechno pro tento podtyp, včetně zděděných vlastností. Takže nebude žádná tabulka pro nadtyp.
3. **Separace vlastností** – Bude jedna tabulka pro nadtyp, a pro každý podtyp další tabulka se sloupci specifickými pro tento podtyp a s cizím klíčem ukazujícím na "mateřský" záznam v tabulce nadtypu. Tyto cizí klíče budou zároveň unikátní v tabulkách podtypu.

Ve fyzickém modelu [Obrázek 13] můžeme nyní vidět, jak se vztahy mezi entitami převedly na cizí klíče v jednotlivých tabulkách. Například vztah entity order „je objednáno uživatelem“ se převedl na atribut tabulky order s název „user_id“, což nyní reprezentuje primární klíč tabulky user.



Obrázek 13 Fyzický model návrhu databáze, zdroj (autor)

Laravel framework pomocí technologie tzv. „migrations“ umožňuje navrhnout a postupně verzovat celou databázi. Vytvoříme si tedy ke každé tabulce třídu, kde definujeme její návrh viz. níže. Poté pomocí jednoduchého příkazu (php artisan migrate) ve složce projektu spustíme třídy, které jsme vytvořili [Obrázek 14] a necháme vytvořit databázi.

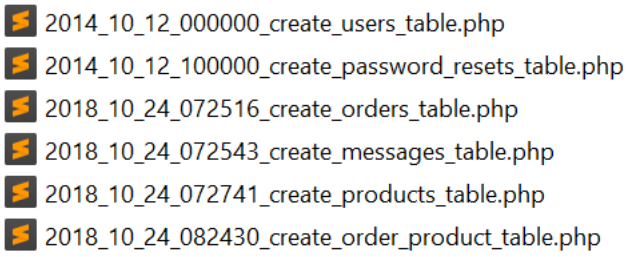
1 Třída pro vytvoření schématu tabulky, zdroj (autor)

```
1. <?php
2.
3. use Illuminate\Support\Facades\Schema;
4. use Illuminate\Database\Schema\Blueprint;
```

```php
use Illuminate\Database\Migrations\Migration;

class CreateOrdersTable extends Migration
{
    public function up()
    {
        Schema::create('orders', function (Blueprint $table) {
            $table->increments('id');
            $table->integer('user_id');
            $table->string('status');
            $table->decimal('total_price', 11, 2);
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::dropIfExists('orders');
    }
}
```

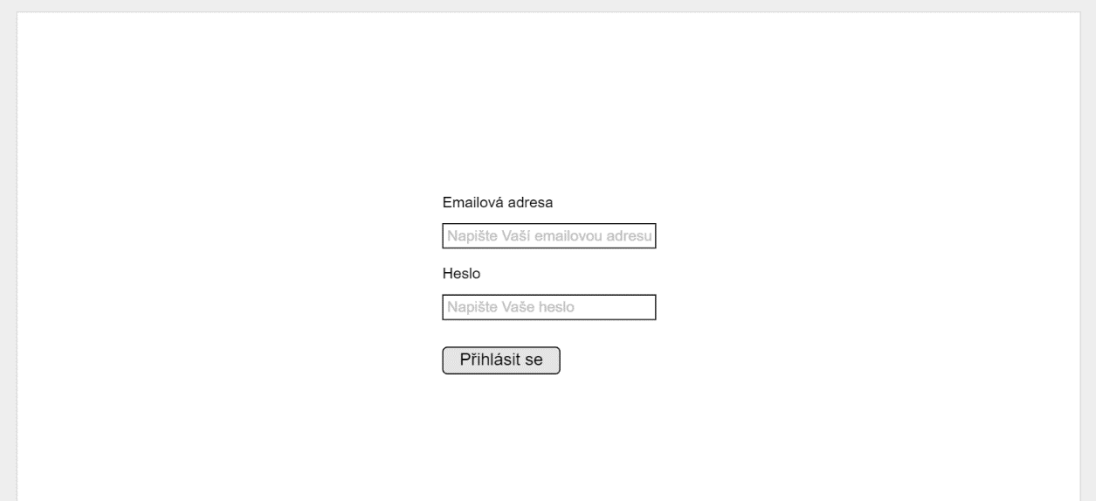


Obrázek 14 Struktura migračních tříd databáze, zdroj (autor)

4.1.4 GUI

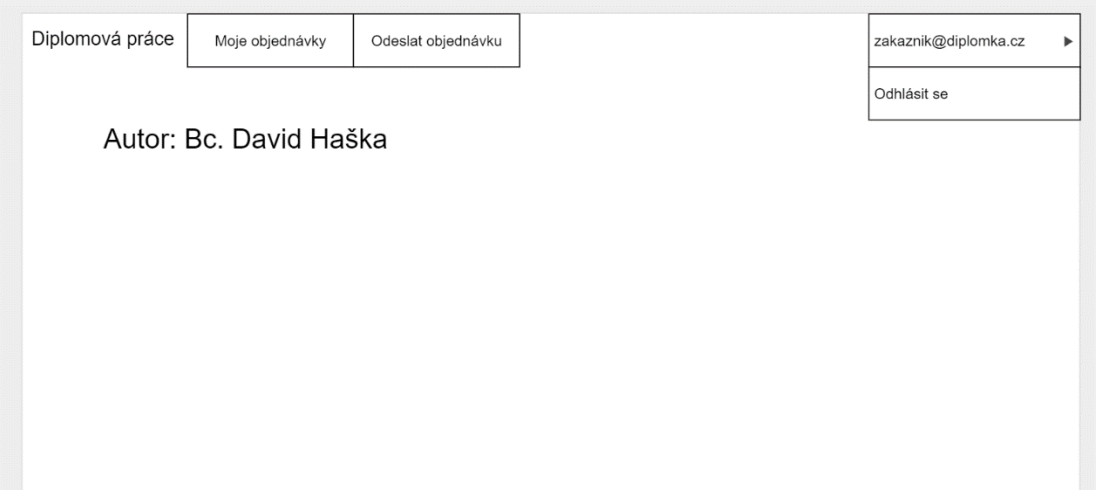
Pro návrh GUI, respektive graphical user interface, v českém překladu uživatelské rozhraní, byl použit software pro tvorbu drátěných modelů tzv. wireframů, Moqups. Konkrétně se jedná o neplacenou verzi tohoto software.

Prvním drátěným modelem je model stránky pro přihlášení. K přihlášení je potřeba email a heslo a z toho vychází také návrh, který počítá se dvěma vstupními textovými poli včetně tlačítka, který při zadání správných údajů uživatele přihlásí a v opačném případě nikoliv viz. [Obrázek 15].

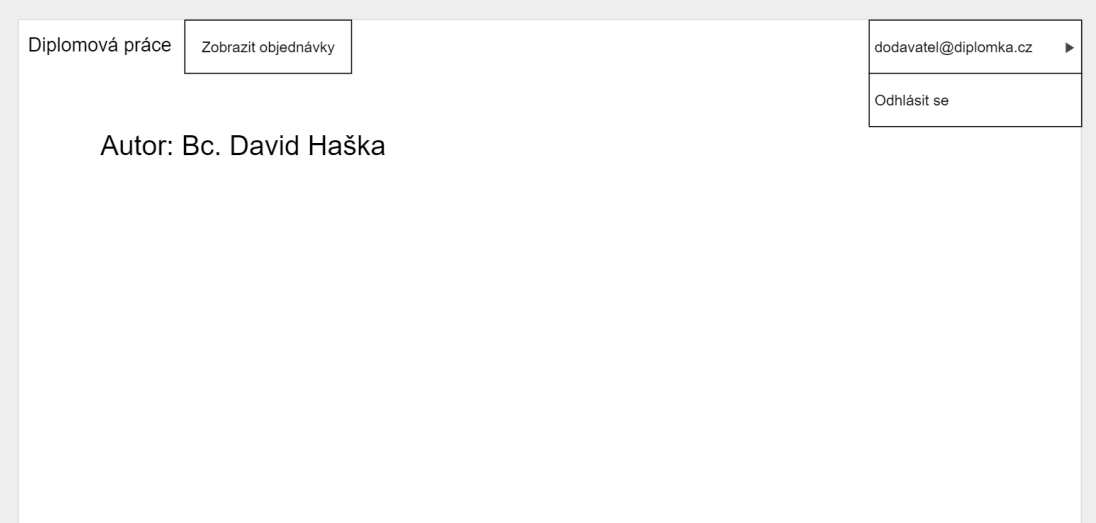


Obrázek 15 Návrh GUI – Přihlášení, zdroj (autor)

Dalším drátěným modelem jsou úvodní stránky po přihlášení uživatele. Stránky se liší podle role uživatele. Odběratelovi se zobrazí stránka s odkazy, které umožní dostat se buď na výpis objednávek nebo na formulář pomocí kterého lze objednat produkty nabízené dodavatelem viz. [Obrázek 16]. Dodavatel má možnost vidět pouze výpis všech objednávek, tedy objednávek od všech uživatelů viz. [Obrázek 17]. Společným odkazem pro obě role je odkaz pro odhlášení se stránky, což uživatele přesměruje zpět na přihlašovací formulář viz. [Obrázek 15].



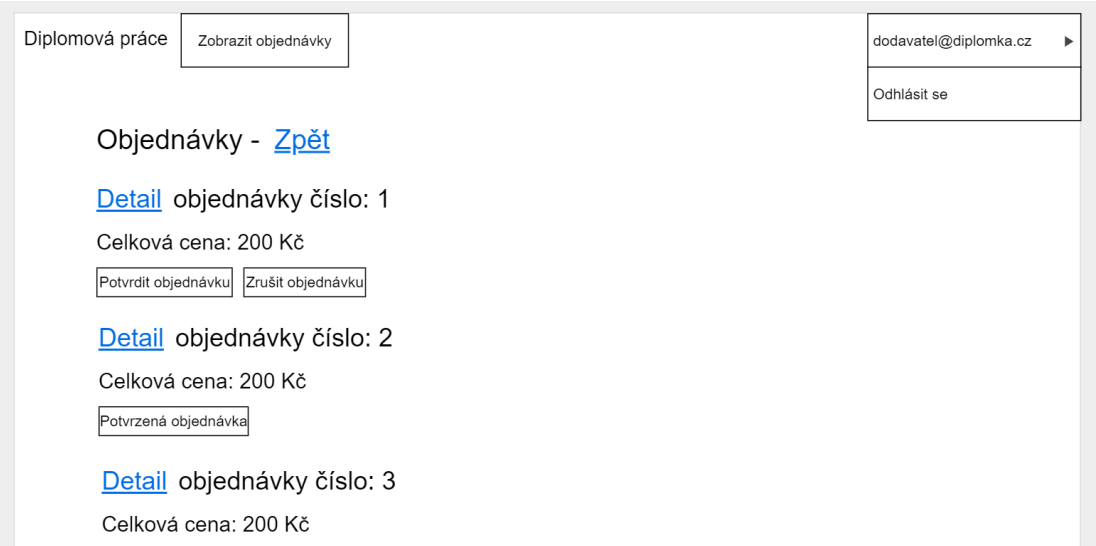
Obrázek 16 Návrh GUI – úvodní stránka odběratele, zdroj (autor)



Obrázek 17 Návrh GUI – úvodní stránka dodavatele, zdroj (autor)

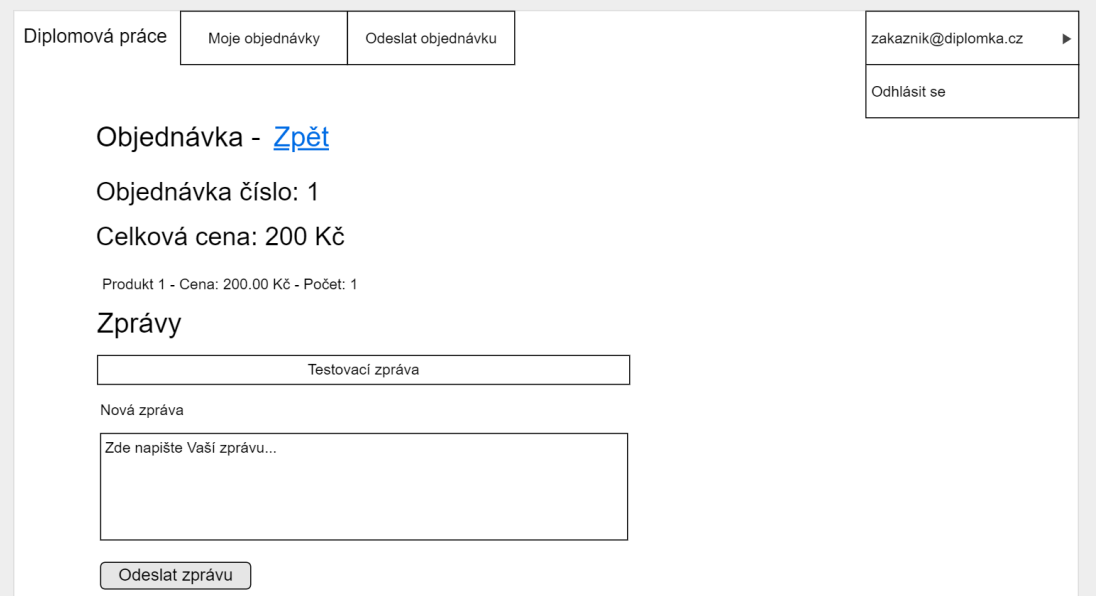
Dalšími drátěnými modely jsou stránky pro výpis objednávek. Odběratel vidí všechny své objednávky s tím, že má možnost pomocí odkazu u objednávky si zobrazit detail objednávky. Dále vidí číslo objednávky, celkovou cenu a status objednávky viz. [Obrázek 18]. Stejnou stránku vidí i dodavatel s tím rozdílem, že má k dispozici tlačítka pro potvrzení nebo zrušení objednávky viz. [Obrázek 19].

Obrázek 18 Návrh GUI – výpis objednávek odběratele, zdroj (autor)



Obrázek 19 Návrh GUI – výpis objednávek dodavatele, zdroj (autor)

Dalším drátěným modelem je stránka detailu objednávky, na které je také možnost výměny zpráv mezi dodavatelem a odběratel. Detail objednávky zobrazuje číslo objednávky, celkovou cenu a objednané produkty včetně ceny a množství viz. [Obrázek 20]. Blok pro zprávy obsahuje jednotlivé zprávy včetně textového pole pro napsání nové zprávy včetně tlačítko pro odeslání této zprávy.



Obrázek 20 Návrh GUI – detail objednávky, zdroj (autor)

Posledním drátěným modelem je stránka pro odeslání objednávky. Stránka zobrazuje výpis produktů. U produktů, které nejsou v objednávce je možnost jejich přidání pomocí tlačítka „Přidat do košíku“. Také je zde možnost zvolit množství. U produktů, které jsou již v objednávce je informace o vloženém množství a tlačítko pro odebrání z objednávky. Stránka také zobrazuje celkovou cenu objednávky a tlačítko s možností odeslat objednávku viz. [Obrázek 21].

Obrázek 21 Návrh GUI – odeslání objednávky, zdroj (autor)

4.1.5 Backend a REST API

Jelikož aplikace pracuje s databází a samotný frontend, tedy JavaScript, nemá vrstvu pro práci s databází, je potřeba právě pomocí jazyka PHP a frameworku Laravel vytvořit na straně serveru mezivrstvu, která bude zpracovávat požadavky od uživatele a zpět mu odesílat výsledky na tyto dotazy. Vytvoříme tzv. REST službu, která bude zpracovávat http metody zmíněné již v teoretické části práce.

První je potřeba vytvořit „routes“ neboli cesty aplikace. Tyto cesty aplikace určují, jak se bude aplikace chovat a co bude vracet v případě návštěvy konkrétní URL[8]. V případě služby je to specifické tím, že akci musíme definovat svojí vlastní cestu. Laravel toto ulehčuje zadáním pouze jednoho řádku kódu a zbytek cest uloží za nás s tím, že první argument určuje začátek cesty a druhý argument, která PHP třída tuto cestu zpracovává.

2 Řádek kódu pro vytvoření cest všech funkcí dané třídy, zdroj (autor)

```
1. Route::resource('orders', 'OrderController');
```

Automaticky Framework také vytvoří již zmíněnou třídu s předpřipravenými metodami, takže programátor může ihned psát kód, aniž by se musel zabývat, jak vytvořit tuto třídu.

.....

[8] URL je zkratka z anglického Uniform Resource Locator. Používá se pro přesnou identifikaci dokumentů na internetu.

3 Třída kontroleru vytvořená pomocí příkazu, zdroj (autor)

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class OrderController extends Controller
{

    public function index(Request $request)
    {
        // funkce pro akci read metodou POST
    }

    public function create()
    {
        // funkce pro akci create metodou POST
    }

    /* a tak dále */
```

Dále je potřeba vytvořit „events“, tedy události, které umožní vracet výsledek zpět do aplikace, respektive uživateli, aniž by musel aktualizovat webovou stránku. Jde o to, že pomocí JavaScriptu lze odeslat asynchronní požadavek na REST službu, ale REST služba už nemůže odeslat zpět asynchronní požadavek JavaScriptu. Musíme použít další mezivrstvu, která by toto umožnila pomocí socket.io a Redis databáze, jak je již zmíněno v teoretické části práce. REST služba může do Redis databáze odeslat a uložit zpracovaný požadavek. V reálném čase pak NodeJS server tento zpracovaný požadavek zachytí a pomocí socket.io odešle všem připojeným. A jelikož je připojený pouze konkrétní uživatel, již při spuštění webové aplikace se pomocí unikátního klíče připojí k socket.io, tak se tento požadavek odešle pouze tomuto uživateli a tomu se již výsledek zpracuje pomocí kódu v JavaScriptu. Třída události vytvořené objednávky potom může vypadat takto.

4 Třída události, zdroj (autor)

```php
<?php

namespace App\Events;

use Illuminate\Broadcasting\Channel;
use Illuminate\Queue\SerializesModels;
use Illuminate\Foundation\Events\Dispatchable;
use Illuminate\Broadcasting\InteractsWithSockets;
```

```
9.  use Illuminate\Contracts\Broadcasting\ShouldBroadcast;
10. 
11. class OrderWasCreated implements ShouldBroadcast
12. {
13.     use Dispatchable, InteractsWithSockets, SerializesModels;
14. 
15.     public $order;
16. 
17.     public function __construct($order)
18.     {
19.         $this->order = $order;
20.     }
21. 
22.     public function broadcastOn()
23.     {
24.         return new Channel('orders');
25.     }
26. }
```

Proměnná $order v konstruktoru třídy určuje jaké data se budou přeposílat dále na Redis databázi, respektive na NodeJS server a poté přímo do aplikace. Název kanálu v třídě Channel a metodě broadcastOn definuje na jaký kanál se budou tyto data odesílat. Samotný název celé této třídy určuje název události, které je třeba naslouchat na straně JavaScriptu viz. níže.

5 Instance třídy Echo pro naslouchání událostí, zdroj (autor)

```
1.  Echo.channel('orders')
2.      .listen('OrderWasCreated', (data) => {
3.          this.AddCreatedOrderToList(data.order.content);
4.      });
```

Naslouchání na kanále „orders“ pro události „OrderWasCreated“ zaručí, že při vytvoření objednávky a jejím zpracováním skrz Redis a NodeJS server, se spustí metody „AddCreatedOrderToList“, která v tomto případě objednávku v reálném čase přidá do výpisu všech objednávek.

4.1.6 Frontend

Tímto se již dostáváme k samotnému frontendu, tedy JavaScriptové části. Prvně je potřeba nastavit cesty, obdobně jako u API „routes“, které uživatel bude moci navštívit a na základě čehož se bude zpracovávat kód a zobrazovat výsledek.

6 Nastavení cest ke komponentám, zdroj (autor)

```
ReactDOM.render(
    <Provider store={ store }>
        <Router history={ browserHistory } render={ applyRouterMiddleware( useScroll() ) }>
            <Route path="/" component={ Layout }>
                <IndexRoute component={ Home }/>
                <Route path="send-order" component={ OrderCreate }/>
                <Route path="my-orders" component={ MyOrderList }/>
                <Route path="get-orders" component={ OrderList }/>
                <Route path="show-order/:id" component={ OrderDetail }/>
            </Route>
        </Router>
    </Provider>,
    app);
```

V rámci toho, co naše aplikace musí umět, tak stačí pět těchto cest. Výchozí cestou je, pokud uživatel navštíví aplikaci bez parametrů v URL adrese. Na této stránce se zobrazí komponenta s názvem Home. Komponenta s názvem Layout je v tomto případě rodič všech komponent a zobrazuje se tedy pokaždé. Na základě funkčnosti těchto komponent se zobrazí buďto formulář s přihlášením uživatele do systému a pokud se uživatel přihlásí nebo je již přihlášený, tak stránka s uvítacím textem a odkazy na další stránky. K této funkčnosti je potřeba upravit metodu render komponenty layout následovně.

7 Vykreslovací funkce komponenty Layout, zdroj (autor)

```
render() {
    const { authenticating, isAuthenticated } = this.props;

    let content = null;

    if (authenticating) {
        content = "Přihlašuji...";
    } else {
        if  ( ! isAuthenticated) {
            content = <LoginForm/>;
        } else {
            content = this.props.children;
        }
    }

    return(
        <div>
            <div className="container">
                <div className="row">
```

```
                    <div className="col-md-12">
                        {content}
                    </div>
                </div>
            </div>
        </div>
    );
}
```

Redux umožňuje držet stav aplikace v jedné instanci a v tomto případě používáme dvě proměnné authenticating a isAuthenticated. Proměnná authenticating uchovává informaci o tom, zdali se uživatel momentálně přihlašuje či nikoliv. Proměnná isAuthenticated určuje, zdali je uživatel přihlášený či nikoliv. Jak můžeme vidět z kódu, tak na základě těchto dvou proměnných se definuje obsah proměnné content, která je poté vykreslena v bloku metodou render. Z podmínek lze vyčíst, že pokud se uživatel přihlašuje, vypíše se hláška, že se přihlašuje a nic jiného. Pokud se momentálně uživatel nepřihlašuje, tak musíme zjistit, jestli je již přihlášený či nikoliv. Pokud není přihlášený, tak vypíšeme komponentu přihlašovacího formuláře LoginForm viz. níže a pokud přihlášení je, tak vypíšeme všechny potomky této komponenty, což už víme, že je komponenta Home, která je nadefinována v nastavení cest aplikace.

8 Komponenta přihlašovacího formuláře, zdroj (autor)

```
import React from "react";

import { connect } from 'react-redux';

import { logIn } from '../actions/authActions';

export class LoginFormRaw extends React.Component {
    constructor(props) {
        super(props);

        this.state = {
            email: '',
            password: '',
        };
    }

    handleSubmit(event) {
        event.preventDefault();

        let submit = {
            email: this.state.email,
            password: this.state.password,
```

```
23.         };
24.
25.         this.setState({ email: '' });
26.         this.setState({ password: '' });
27.
28.         this.props.logIn(submit);
29.     }
30.
31.     render() {
32.         return(
33.             <form>
34.                 <div className="form-group">
35.                     <label htmlFor="emailInput">Emailová adresa</label>
36.                     <input type="email"
37.                            className="form-control"
38.                            name="email"
39.                            id="emailInput"
40.                            placeholder="Napište Vaší emailovou adresu"
41.                            value={this.state.email}
42.                            onChange={(e) => this.setState({ [e.target.nam
   e]: e.target.value })}/>
43.                 </div>
44.                 <div className="form-group">
45.                     <label htmlFor="passwordInput">Heslo</label>
46.                     <input type="password"
47.                            className="form-control"
48.                            name="password"
49.                            id="passwordInput"
50.                            placeholder="Napište Vaše heslo"
51.                            value={this.state.password}
52.                            onChange={(e) => this.setState({ [e.target.nam
   e]: e.target.value })}/>
53.                 </div>
54.                 <div className="form-check">
55.                     <button type="submit"
56.                             className="btn btn-primary"
57.                             onClick={(e) => this.handleSubmit(e)}>Přihlás
   it se</button>
58.                 </div>
59.             </form>
60.         );
61.     }
62. }
63.
64. const mapStateToProps = state => ({});
65.
66. const mapDispatchToProps = {
67.     logIn,
68. };
69.
70. export const LoginForm = connect(
71.     mapStateToProps,
72.     mapDispatchToProps,
```

```
73. )(LoginFormRaw);
```

Komponenta LoginForm pouze zpracovává vstupy od uživatele a pomocí Redux akce login tyto informace odesílá na REST službu, která uživatele buďto přihlásí nebo nepřihlásí.

4.2 Vývoj a implementace

Tato část se zabývá vývojem, výsledkem implementace do testovacího prostředí včetně samotného testování vůči specifikovaným požadavkům v návrhu aplikace.

4.2.1 Testovací data

Návrh aplikace je vytvořen tak, že jediná testovací data, které k provozu aplikace potřebujeme jsou dva uživatelé a dva testovací produkty, které si může odběratel objednat.

Prvním uživatelem je tedy odběratel, který se přihlašuje do aplikace pomocí emailu „odberatel@diplomka.cz“ a hesla „test123“. Uživatel má v databázi uloženou informaci o svojí roli, což je konkrétně „customer“ a to pomocí logiky v aplikaci zaručí, že se tomuto uživateli zobrazí správná stránka a že bude mít přístup pouze k těm stránkám, které jsou pro něj určeny.

Druhým uživatelem je dodavatel, který se přihlašuje do aplikace pomocí emailu „dodavatel@diplomka.cz“ a hesla „test123“. Tento uživatel má stejně jako odběratel informaci o jeho roli.

Posledními daty jsou dva produkty, kde první z nich se nazývá „Produkt 1“ a druhý z nich se nazývá „Produkt 2“. První má cenu 200,- Kč a druhý 400,- Kč a to z důvodu ověření funkčnosti jako je součet celkové objednávky pro dva odlišné produkty. Skladovou dostupnost aplikace neřeší.

4.2.2 Příkazy aplikace

Aby mohla aplikace fungovat musíme před jejím nasazením znát specifické příkazy, které se zadávají přímo na serveru do konzole. Jedná se o příkazy, které zajistí správné vytvoření tabulek databázi, správně naplnění databáze testovacími daty nebo také správné fungování přeposílání informací o událostech v aplikaci, které jsou určeny pro správný chod aplikace a jedná se o tyto příkazy (Otwell, 2018, online):

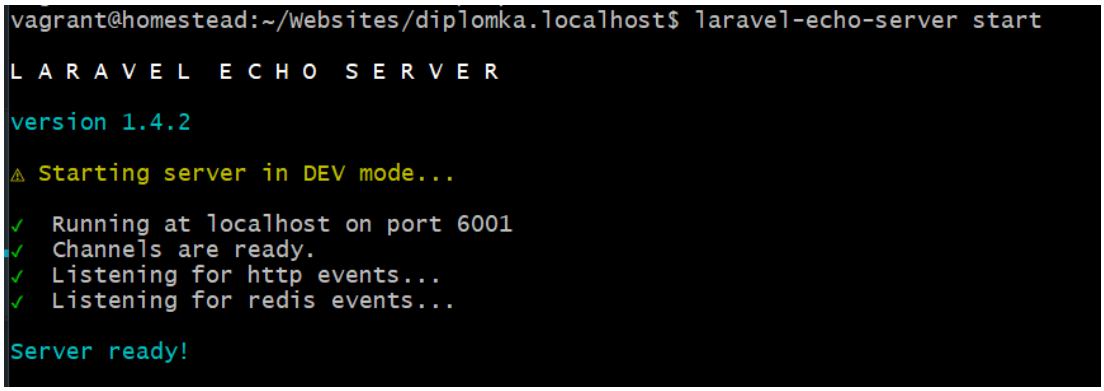
- php artisan migrate
    - Vytvoří definované tabulky v databázi.
- php artisan db:seed

- Naplní tabulky testovacími daty, které jsou definovány ve třídě DatabaseSeeder viz ukázka testovacích dat uživatelů.

9 Database seeder, zdroj (autor)

```
1.  $user = new \App\User;
2.  $user->email = 'odberatel@diplomka.cz';
3.  $user->password = bcrypt('test123');
4.  $user->type = 'customer';
5.  $user->save();
6.
7.  $user = new \App\User;
8.  $user->email = 'dodavatel@diplomka.cz';
9.  $user->password = bcrypt('test123');
10. $user->type = 'supplier';
11. $user->save();
```

- php artisan queue:work
  - Spustí Redis databázový server.
- laravel-echo-server start
  - Spustí NodeJS server viz. [Obrázek 22].

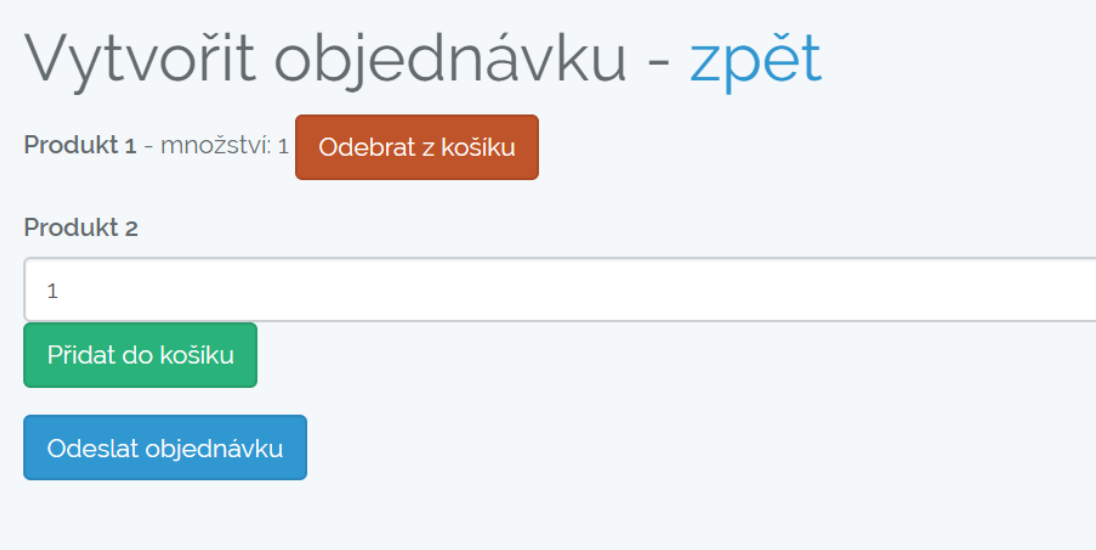


Obrázek 22 Spuštění NodeJS serveru, zdroj (autor)

4.3 Ovládání aplikace a ověření funkčních požadavků aplikace

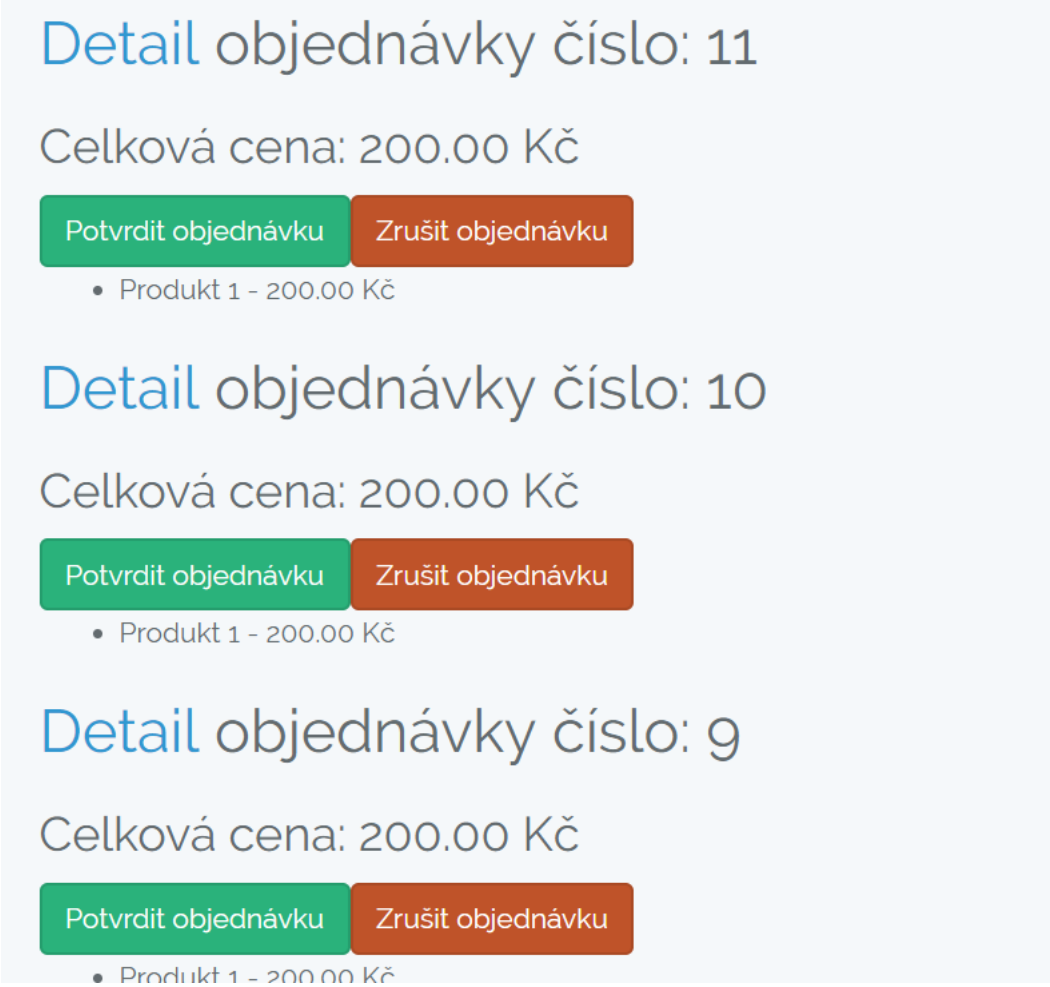
Aplikace byla tvořena tak, aby pokryla jednotlivé funkční požadavky definované v návrhu. Každá funkce má svoji stránku a vždy se liší podle role aktuálně přihlášeného uživatele – role jsou tedy dvě, zákazník, respektive odběratel a dodavatel. Zákazník je na stránce z důvodu objednání produktů, a proto vývoj začal stránkou s formulářem objednávky viz. [Obrázek 23]. Na této stránce můžeme kromě možnosti kroku zpět, také přidání nebo odebrání produktu z košíku, s tím, že v případě přidání produktu lze volit množství tohoto produktu. Poslední možností je odeslání celé objednávky, která je asynchronně zpracována,

takže po úspěšném odeslání je dodavatel informován o nové objednávce bez nutnosti znovu načítat aplikaci.



Obrázek 23 Aplikace – vytvoření objednávky, zdroj (autor)

Jak objednávky vidí uživatel můžeme vidět na stránce přehledu objednávek pro dodavatele viz. [Obrázek 24]. Kromě toho, že si dodavatel může zobrazit detail objednávky, tedy co, a v jakém počtu si zákazník objednal, tak také může objednávku potvrdit nebo odmítnout v závislosti třeba na stavu skladu nebo v závislosti na vytíženosti výroby.

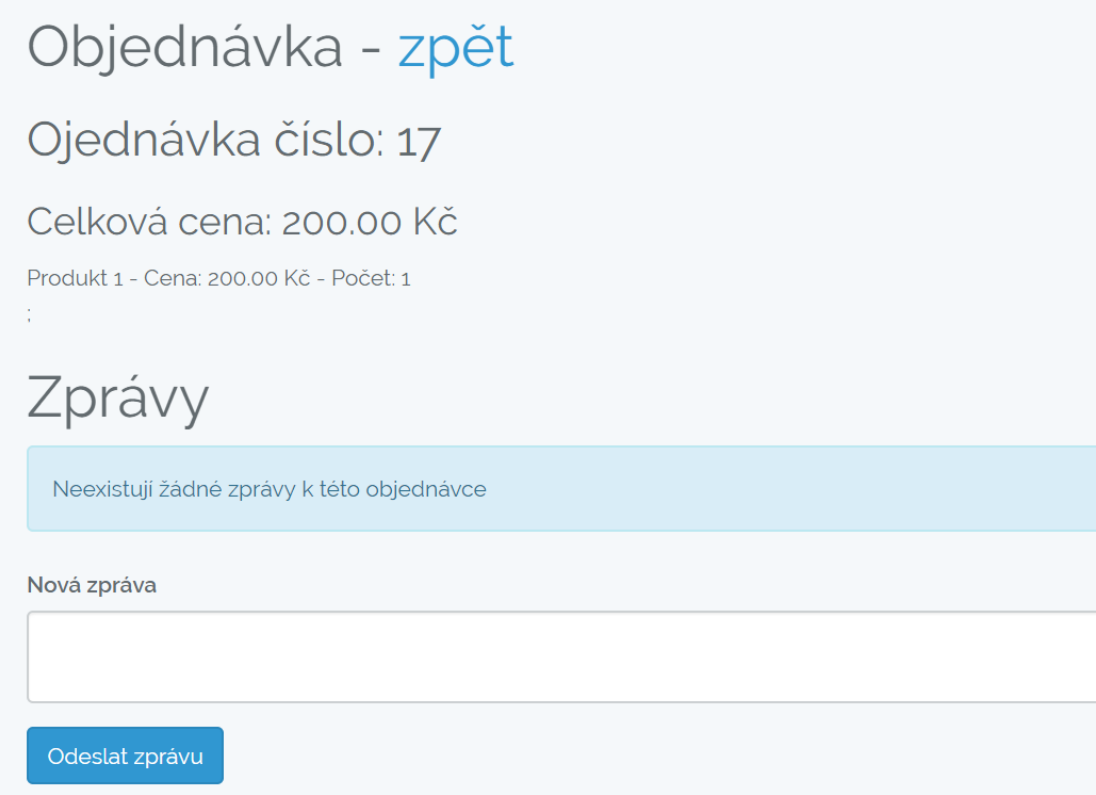


Obrázek 24 Aplikace – přehled objednávek v roli dodavatele, zdroj (autor)

Odběratel stránku přehledu objednávek vidí na první pohled podobně, ale jsou zde dva rozdíly viz. [Obrázek 25]. První rozdíl je ten, že odběratel vidí pouze svoje objednávky, zatímco dodavatel vidí objednávky od všech odběratelů. Druhý rozdíl je ten, že nemá možnost objednávku potvrdit nebo zrušit, ale pouze vidí aktuální stav, který se v reálném čase mění podle toho, jestli dodavatel objednávku potvrdí nebo zruší.

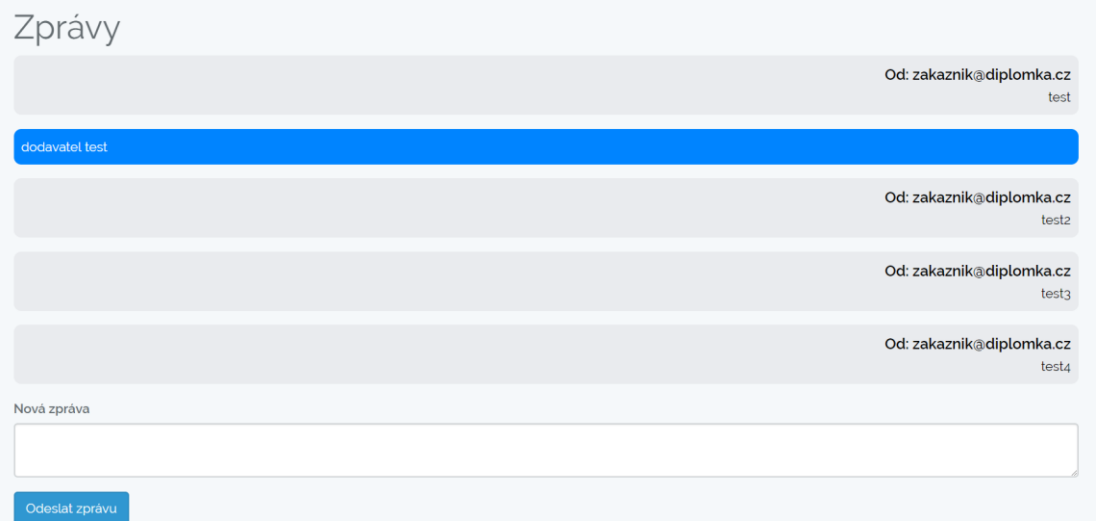
Obrázek 25 Aplikace – přehled objednávek v roli zákazníka, zdroj (autor)

Detail objednávky naopak vidí obě role stejně, a kromě výpisu objednaných produktů včetně počtu je na stránce také formulář na odesílání vzájemných zpráv k této objednávce viz. [Obrázek 26].



Obrázek 26 Aplikace – detail objednávky, zdroj (autor)

Tento formulář, stejně jako celé aplikace pracuje v reálném čase, takže odesílání a příjem zpráv zde dochází bez nutnosti znovu načítání celé stránky. Zobrazení zpráv zde dochází dle času jejich odeslání. Zpráva s modrým pozadím znázorňuje uživatelem odeslanou zprávu, což je konkrétně na tomto obrázku dodavatel viz. [Obrázek 27]. Pokud by si detail této objednávky načetl odběratel, který tuto objednávku vytvořil, tak se zprávy načtou s opačným pozadím.



Obrázek 27 Aplikace – přehled zpráv u objednávky včetně formuláře na odeslání nové zprávy, zdroj (autor)

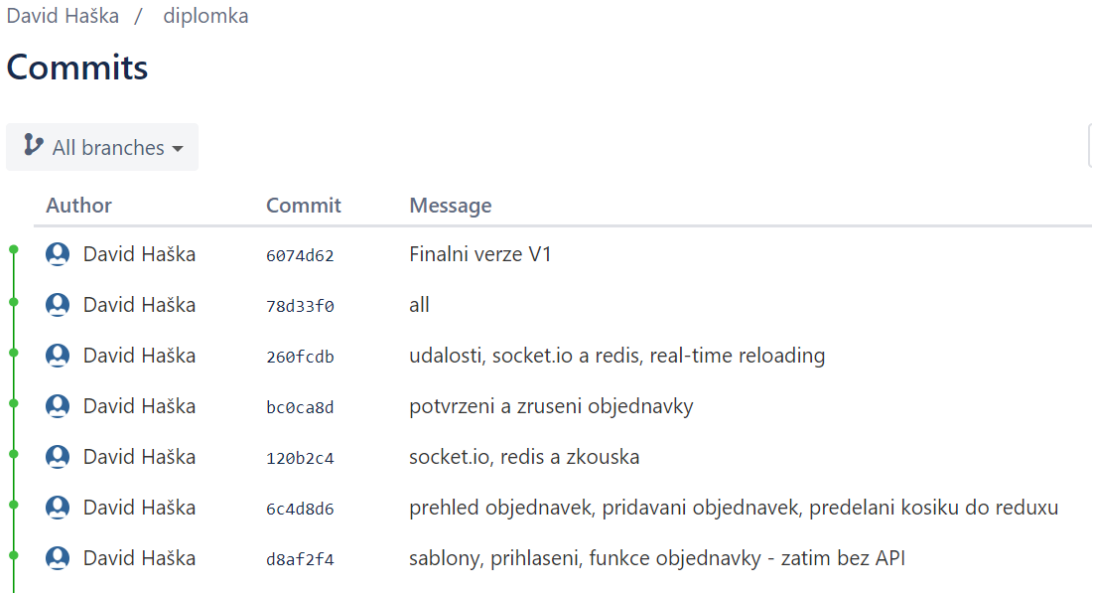
4.4 Další použité technologie

Kromě technologií, které přímo souvisejí s funkčností aplikace byly použity také technologie, které umožnili a ulehčili vývoj. Případně umožní další rozvoj.

4.4.1 Git a Bitbucket

Veškerý vývoj probíhal přes Bitbucket, což je portál, který zajišťuje git repositáře pro správu projektů viz. https://bitbucket.org/davidhaska/diplomka.

Git jako takový slouží vývojářům zejména pro správu webových projektů, a tedy možnost například shlédnout historii projektu a jeho verzí, případně se k nějaké verzi vracet. Další věcí je samotná záloha celého projektu, což jsem využil já při tvorbě tohoto projektu viz. [Obrázek 28]. V neposlední řadě toto může usnadnit provoz a údržbu aplikace přímo na serveru, protože pomocí synchronizace tohoto repositáře můžeme docílit aktuálnosti projektu, jak na repositáři, tak přímo na serveru (Valkovič, 2014, online).



Obrázek 28 Git repositář na bitbucket.org, zdroj (autor)

4.5 Možnosti dalšího rozvoje

Výsledná aplikace slouží především jako demonstrace technologie realtime webových aplikací, a tak je zde spousty možností, jak tuto aplikace dále rozvíjet. V následujících podkapitolách popíšu jednotlivé možnosti.

4.5.1 Frontend a GUI

GUI neboli grafické rozhraní aplikace může být rozšířena o návrh na podporu uživatelské přívětivosti a uživatelského zážitku z užívání aplikace zvaného UX, tedy user experience.

4.5.2 Backend a mobilní aplikace

Backend a API jsou tvořeny, tak aby ho do budoucna mohly jako službu využívat jiné aplikace. Tímto se zde otevírá možnost pro návrh mobilní aplikace, která by stejně jako webová aplikace pracovala realtime.

4.5.3 OAuth

OAuth je prokol, který standardizuje proces autentizace a autorizace. Pomocí protokolu OAuth je možno aplikaci rozšířit o přihlášení pomocí služeb, které tento protokol používají. Mohlo by se do aplikace přihlašovat například pomocí Google účtu.

4.5.4 Vícejazyčnost

Aplikace zobrazuje texty pouze v českém jazyce a to tak, že jsou na pevno nastaveny v jednotlivých šablonách komponent. Zde je možnost o rozšíření na více jazyků včetně administrace jednotlivých jazyků.

4.5.5 Skupiny uživatelů

Nyní aplikace pracuje pouze se dvěma rolemi, a to odběratel a dodavatelem. Ve větších firmách je možno se setkat s více rolemi jako například skladník, účetní nebo dokonce externí kurýr či přepravce, který se stará o dodání zboží. V tomto směru lze aplikaci rozšířit o více rolí.

4.5.6 Více dodavatelů

Aplikace je nyní navržena tak, že jí může používat pouze jeden dodavatel. Rozšířením o možnosti přihlášení více dodavatelů lze aplikaci nabízet jako službu, respektive SAAS.

4.5.7 Filtrování

Objednávky a produkty se nyní zobrazují abecedně bez možnosti jiného řazení. Aplikaci je tedy možné rozšířit o filtrování dle jednotlivých parametrů jako například cena.

4.5.8 Doprava

Rozšíření se může týkat také volby dopravy při odesílání objednávky.

4.5.9 Platba

Rozšíření o možnost platby včetně volby konkrétní možnosti. Nyní s tímto aplikace nepočítá.

4.5.10 Správa a administrace

Tato kapitola souvisí se všemi předešlými. Aplikaci lze rozšířit o administraci jednotlivých části, což by umožnilo celkovou správu aplikace. Toto rozšíření by znamenalo, že by šlo v aplikaci měnit názvy nebo ceny produktů, přidávat a odebírat odběratele.

4.6 Dostupnost aplikace

Aplikace je dostupná na veřejném git repositáři společnosti Bitbucket.org (https://bitbucket.org/davidhaska/diplomka) a také na webové adrese http://46.101.240.67/.

4.7 Využití aplikace

Aplikaci může využít kdokoliv, kdo bude chtít dále aplikaci rozšiřovat nebo jí dokonce provozovat, protože zdrojový kód je veřejný.

4.8 Zhodnocení

Nejdříve byl vytvořen návrh celé aplikace. Tedy včetně popisu aplikace, návrhu funkčních požadavků, návrhu databáze, návrhu grafického uživatelského rozhraní, backendu, REST API a frontendu. Na základě tohoto komplexního návrhu byla vytvořena aplikace spolu s testovacími daty. Pomocí návrhu funkčních požadavků byla ověřena funkčnost aplikace. Dále byly sepsány další technologie, které byly použity při vývoji aplikace, ale nesouvisely přímo s funkčnosti aplikace jako třeba uložiště pro zdrojové kódy. Byly zmíněny možnosti rozšíření aplikace, což může pomoci nejenom tomu, kdo bude aplikaci dále rozšiřovat, ale také tomu, kdo bude psát kvalifikační práci na podobné téma. Bylo sepsáno, kde lze aplikaci vyzkoušet a kde lze nalézt zdrojové kódy, které jsou veřejné.

Závěr

V úvodní části diplomové práce byl definován hlavní cíl navrhnout a vyvinout realtime aplikaci pomocí frameworků ReactJS a Laravel, který byl rozdělen na dílčí cíle:

1. Seznámení s realtime webovými aplikacemi včetně technologií k tomu určených a vysvětlení s rozdíly oproti klasickým aplikacím.
2. Seznámení s JavaScriptovým frameworkem ReactJS (frontend).
3. Seznámení s PHP frameworkem Laravel (backend) a tvorbou REST API.
4. Definice rámce realtime webové aplikace a návrh vývoje realtime webové aplikace.
5. Vývoj aplikace pomocí definovaného návrhu a pomocí frameworků ReactJS a Laravel.
6. Ověření funkčních požadavků implementované aplikace.

Dílčí cíl č. 1 spočívá ve vysvětlení všech pojmů a technologií související s problematikou realtime webových aplikací. Tato část také seznamuje s rozdíly oproti klasickým webovým aplikacím. Tento dílčí cíl je splněn v kapitolách 3.1 a 3.2.

Dílčí cíl č. 2 spočívá v seznámení s frameworkem ReactJS, který slouží jako frontend část aplikace. Tento dílčí cíl je splněn v kapitole 3.3.

Dílčí cíl č. 3 spočívá v seznámení s frameworkem Laravel, který slouží jako backend část aplikace včetně REST API. Tento dílčí cíl je splněn v kapitole 3.4.

Dílčí cíl č. 4 spočívá v definování rámce funkčních požadavků aplikace včetně vytvoření use case diagramu a vytvoření návrhu realtime webové aplikace. Tento dílčí cíl je splněn v kapitole 4.1.

Dílčí cíl č. 5 spočívá ve vývoji aplikace na základě definovaného návrhu v kapitole 4.1. za použití frameworku ReactJS a Laravel. Tento dílčí cíl je splněn v kapitole 4.2.

Dílčí cíl č. 6 spočívá v ověření funkčních požadavků definovaných v návrhu aplikace v kapitole 4.1.2. Tento dílčí cíl je splněn v kapitole 4.3.

Diplomová práce může sloužit jako zdroj pro další potřeby v budoucnu. Ať již inspirací pro jiné návrhy vývoje realtime aplikace nebo jako reference obsahující komplexní informace pro vývoj podobné aplikace. Tento návrh a postup vývoje lze dále rozšiřovat, upřesňovat a přizpůsobovat pro potřeby projektu. Jako možnost dalšího rozvoje práce lze rozšířit aplikaci o podporu skladového nebo účetního systému nebo rozšíření o internetový obchod. Zajímavá může být analýza návrhů, jež určí, pro jaké typy projektu jsou dané návrhy nejvhodnější, případně jakým způsobem bylo potřeba návrh upravit pro potřeby projektu.

Seznam literatury

TOMAN, František. Real-time webové aplikace [online]. Praha, 2012 [cit. 2018-03-14]. Dostupné z: https://insis.vse.cz/zp/portal_zp.pl?podrobnosti_zp=33191. Bakalářská práce. Vysoká škola ekonomická v Praze.

NEZDARA, Vojtěch. Vývoj moderních webových aplikací [online]. Praha, 2017 [cit. 2018-03-14]. Dostupné z: https://insis.vse.cz/zp/portal_zp.pl?podrobnosti_zp=57704. Diplomová práce.

ZIKMUND, Marian. Architektura bezserverových jednostránkových aplikací v jazyku JavaScript [online]. Praha, 2016 [cit. 2018-03-14]. Dostupné z: https://insis.vse.cz/zp/portal_zp.pl?podrobnosti_zp=55782. Diplomová práce. Vysoká škola ekonomická v Praze.

NGUYEN, Viet Bach. Internetová obchodní galerie – kooperace e-shopů pro maloobchodní prodej [online]. Praha, 2017 [cit. 2018-03-14]. Dostupné z: https://insis.vse.cz/zp/portal_zp.pl?podrobnosti_zp=62758. Diplomová práce. Vysoká škola ekonomická v Praze.

ZVYAGINTSEVA, Daria. Platforma Meteor jako webové prostředí pro vývoj aplikaci [online]. Praha, 2017 [cit. 2018-03-14]. Dostupné z: https://insis.vse.cz/zp/portal_zp.pl?podrobnosti_zp=61626. Bakalářská práce. Vysoká škola ekonomická v Praze.

FIALA, Lukáš. Framework Laravel [online]. Olomouc, 2015 [cit. 2018-03-14]. Dostupné z: https://theses.cz/id/6un50g/framework-laravel.pdf. Bakalářská práce.

IVAŠKA, Filip. Implementace webových aplikací pomocí vývojového frameworku Laravel [online]. Zlín, 2017 [cit. 2018-03-14]. Dostupné z: https://portal2.utb.cz/portal/studium/prohlizeni.html. Bakalářská práce.

FEDOSEJEV, Artemij. React.js Essentials. Birmingham: Packt Publishing, 2015. ISBN 978-1-78355-162-0.

BEAN, Martin. Laravel 5 Essentials. Birmingham: Packt Publishing, 2015. ISBN 978-1-78528-301-7.

VIPUL, A M. ReactJS by Example - Building Modern Web Applications with React. Birmingham: Packt Publishing, 2016. ISBN 978-1-78528-964-4.

SAUNIER, Raphaël. Getting Started with Laravel 4. Birmingham: Packt Publishing, 2014. ISBN 978-1-78328-703-1.

RAI, Rohit. Socket. IO Real-Time Web Application Development. Birmingham: Packt Publishing, 2013. ISBN 978-1-78216-078-6.

PUREWAL, Semmy. Learning web app development. Beijing: O'Reilly, 2014. ISBN 978-1-449-37019-0.

PEFFERS, Ken, Tuure TUUNANEN, Marcus ROTHENBERGER a Samir CHATTERJEE. Journal of Management Information Systems: A Design Science Research Methodology for Information Systems Research [online]. 2008 [cit. 2018-11-19]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.535.7773&rep=rep1&type=pdf

W3C -: World Wide Web Consortium [online]. 2017 [cit. 2018-11-18]. Dostupné z: https://www.w3.org/standards/

ADOBE. Co jsou to webové aplikace a dynamické webové stránky? [online]. Adobe, 2017 [cit. 2018-11-15]. Dostupné z: https://helpx.adobe.com/cz/dreamweaver/using/web-applications.html

PAVLÍK, Lukáš. Informační systémy [online]. Olomouc: Moravská vysoká škola Olomouc, 2018 [cit. 2018-11-18]. Dostupné z: https://mvso.cz/wp-content/uploads/2018/02/Informa%C4%8Dn%C3%AD-syst%C3%A9my-studijn%C3%AD-text.pdf

DUCKETT, Jon. HTML & CSS: design and build websites. Indianapolis, IN: Wiley, c2011. ISBN 978-1-118-00818-8.

PONKRÁC, Miloslav. PHP a MySQL: bez předchozích znalostí. Brno: Computer Press, 2007. ISBN 978-80-251-1758-3.

NYKLÍČEK, Jaromír. *Webové aplikace pracující v reálném čase*. Brno, 2013. Diplomová práce. Masarykova univerzita. Vedoucí práce RNDr. Radek Ošlejšek, Ph.D.

MALÝ, Martin. Zdrojak.cz: Web Sockets [online]. 2009 [cit. 2018-11-28]. Dostupné z: https://www.zdrojak.cz/clanky/web-sockets/

MALÝ, Martin. Zdrojak.cz: Node.js – s JavaScriptem na server [online]. 2010 [cit. 2018-11-28]. Dostupné z: https://www.zdrojak.cz/clanky/node-js-s-javascriptem-na-server/

NELSON, Jeremy. Mastering Redis. 2016. Packt Publishing. ISBN 978-1-78398-818-1.

BUGL, Daniel. Learning Redux. 2017. Birmingham, UK: Packt Publishing. ISBN 978-1-78646-239-8.

PASSAGLIA, Andrea. Learning Redux. 2017. Birmingham, UK: Packt Publishing. ISBN 978-1-78646-809-3.

JACKOBSON, Daniel. APIs: A Strategy Guide. 2012. Birmingham, UK: O'Reilly Media. ISBN 978-1-449-30892-6.

MASSÉ, Mark. REST API Design Rulebook. 2012. Sebastopol: O'Reilly Media. ISBN 978-1-449-31050-9.

PALOVSKÁ, Helena. Krokodýlovy databáze: Převod do databázových struktur [online]. Praha, 2011 [cit. 2018-11-16]. Dostupné z: https://krokodata.vse.cz/DM/Mapovani

VALKOVIČ, Petr. Itnetwork.cz: Git – Historie a principy [online]. 2014 [cit. 2018-11-18]. Dostupné z: https://www.itnetwork.cz/software/git/git-tutorial-historie-a-principy

OTWELL, Taylor. Laravel: Migrations [online]. 2018 [cit. 2018-11-19]. Dostupné z: https://laravel.com/docs/5.6/migrations

Seznam obrázků

Seznam kódu

Příloha A: Obsah CD se zdrojovým kódem

app .....složka s obsahem všech funkčních tříd aplikace

bootstrap.....složka s obsahem souborů nutných pro spuštění Laravel frameworku

config.....složka s obsahem konfiguračních souborů

database.....složka s obsahem databázových migračních tříd včetně seederů

public.....složka s hlavním souborem index.php, který spouští stránku a dále se zde nachází zkompilované CSS, JS a také obrázky

resources .....složka se všemi šablonami / komponentami včetně JS a CSS souborů, které se kompilují

routes.....složka se soubory, kde se nastavují "routes", respektive cesty aplikace jako například API

storage .....složka pro ukládání souborů nebo také cache včetně zkompilovaných šablon pro urychlení načítání stránek

tests.....složka se soubory, které obsahuje testy pro aplikaci

.env.example.....ukázkový soubor pro nastavení aplikace v konkrétním prostředí

.gitattributes.....soubor pro konfiguraci chování při práci s git repositářem

.gitignore.....soubor pro ignorování souborů při přidávání do git repositáře

composer.json.....soubor s PHP balíčky pro composer

laravel-echo-server.json .....konfigurační soubor pro NodeJS server

package.json.....soubor s JS balíčky pro npm

phpunit.xml .....konfigurační soubor pro spouštění unit testů

readme.md.....soubor pro popis projektu / aplikace na git repositáři

server.php.....soubor pro spuštění lokálního serveru pro potřeby testování (bez databáze)

webpack.mix.js.....konfigurační soubor pro kompilování CSS a JS souborů ze složky "resources"