VŠB – Technická univerzita Ostrava

Fakulta elektrotechniky a informatiky

Katedra informatiky

Absolvování individuální odborné praxe

Individual Professionla Practice in the Company

2017 Jiří Grussmann

Zadání bakalářské práce

(zde papír co asi nenajdu)

Prohlašuji, že jsem tuto bakalářskou/diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 12.2.2018 ……………………….

Souhlasím se zveřejněním této bakalářské/diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských/magisterských programech VŠB-TU Ostrava.

V Ostravě 12.2.2018 ……………………….

Rád bych poděkoval firmě Poski.com s.r.o. za možnost vykonávat bakalářskou praxi…

**Abstrakt**

Tato bakalářská práce popisuje mé absolvování bakalářské praxe ve firmě Poski.com, ve které jsem pracoval na pozici Front-end developer. V první části je popsáno pracovní zaměření firmy, mé pracovní prostředí a používané technologie. V druhé části jsou popsány jednotlivé úkoly…

**Klíčová slova**: JavaScript, Vue JS, Git, SVN, HTML5, CSS3, SASS, LESS, npm, node.js, Grunt.js, PHP, Laravel

**Abstract**

**Key words**: JavaScript, Vue JS, Git, SVN, HTML5, CSS3, SASS, LESS, npm, node.js, Grunt.js, PHP, Laravel

# Obsah

# Seznam použitých zkratek a symbolů

# Úvod

Bakalářskou práci jsem zvolil formou odborné praxe ve firmě Poski.com s.r.o. Firmu jsem si zvolil, protože nabízela pozici a technologie s kterými jsem měl již zkušenosti a chtěl bych se jim a v budoucnu věnovat. Takže jsem tuto možnost viděl jako dobrou příležitost pro nabytí nových zkušeností, které by mi mohly v budoucnu pomoci.

První část bakalářské práce budu věnovat firmě samotné, stručné ji představím a podrobněji popíšu pracovní pozici na které jsem praxi vykonával. Do této části zahrnu popis a seznámení s jednotlivými technologiemi které jsem při tvorbě všech úkolů použil.

Druhou část věnuji hlavnímu úkolu praxe, popíšu jeho zadání, analýzu, implementaci a problémy s kterými jsem se setkal během vykonávání úkolu.

Třetí část budu věnovat menším úkolům, které jsem v průběhu praxe obdržel, zejména pro seznámeni s jejich pracovním procesem, technologiemi a systémem.

V poslední části zhodnotím nabyté zkušenosti, popíšu co se mi povedlo, popřípadě co třeba na některých úkolech zlepšit.

# Popis společnosti a pracovní pozice

## Popis společnosti

Firma Poski.com s.r.o. sídlí v Ostravě-Mariánských horách. Tato společnost se zabývá vývojem a realizací webových stránek a veškerých marketingových věcí okolo toho.

Firma se dělí na dvě částí, první část se zabývá vývojem realitního software a druhá část se zabývá vývojem klasických webových prezentací a e-shopů na míru, kde jsem byl taky přidělen. Veškeré webové prezentace běží na modulovém redakčním systému Clevero vytvořený touto firmou. Redakční systém je vytvořen technologii PHP a je v neustálém vývoji, nyní ve verzi 7. Na front-endu běží na technologiích HTML5, CSS3, JavaScript a jeho frameworku jQuery a dále Vue.js, který jsem do části systému spolu s kolegou zavedl.

## Popis pracovní pozice

Do firmy jsem nastoupil na pozici Front-end developera. Po kontaktování firmy, jsem byl pozván na krátký pohovor, kde jsem zodpověděl na pár jednoduchých otázek z praxe, dále mi byl přidělen přijímací úkol, kde jsem musel nekódovat jednouchou stránku z přiděleného PSD návrhu. Po zaslání vyhotoveného úkolu jsem byl do firmy na pozici přijat.

Praxi jsem vykonával v jejich kancelářích, kde mi byl přidělen stůl s počítačem a dvěma monitory a veškerým softwarem potřebný k vykonávání mé práce. Pracoval jsem zde s dalšími lidmi z týmu redakčního systému, kde jsem měl možnost libovolné konzultace úkolů s programátory a senior Front-end developery. Taktéž jsem spolupracoval s dalším studentem VŠB TUO, který zde taktéž vykonával odbornou praxi.

Hlavní náplní mé pracovní pozice bylo realizování a nasazování webových prezentací a e-shopů z grafického návrhu na jejich redakční systém. Dále pak příprava dynamických, interaktivních a komfortních rozhraní do jádra jejich redakčního systému. K některým úkolům jsem musel udělat i analýzu a zvolit, vhodnou technologii, nad kterou budu úkol realizovat. Veškeré technologie, které jsem během praxe použil popíši v následující kapitole.

Použité technologie

# Seznam zadaných úkolů

## Hlavní úkol: Implementace komfortního průvodce objednáním produktů

Hlavním úkolem během mé praxe bylo vytvoření komfortního průvodce objednáním produktů což zahrnuje vše od kliknutí na tlačítko přidat do košíku, zobrazení rychlého detailů produktu, samostatné přidání produktů do košíku, až po zpracování kontaktních údajů objednávky, načtení a výběr dopravy a platby a samostatné odeslání zpracované objednávky. Toto všechno realizováno v JavaScriptu bez nutnosti znovu načtení stránky.

# Postup řešení zadaných úkolů

## Hlavní úkol: Implementace komfortního průvodce objednáním produktů

Prvním krokem toho to úkolu bylo vybrání si vhodného JavaScript frameworku a studování jeho dokumentace, dále analýza již existujících košíků na webu, a soupis vhodných a potřebných polí, které by měly položky košíku a jednotlivé kroky objednávky obsahovat. Po vykonáni uvedeného bylo nutno začít pracovat na implementaci průvodce. Jelikož jejich současný systém obsahoval jen statický košík, tak jsem si musel vytvořit pomocné API pro implementaci košíku, abych viděl průběh všech možných stavů v průběhu objednávky a následně aby ve výsledku byl košík připraven k funkční prezentaci. Po vytvoření API jsem se mohl věnovat samotnému JavaScript průvodci, který probíhal v několika fázích (vytvoření vzhledu a struktury v HTML5/CSS3, následně zprovoznění jeho funkčností v JavaScriptu a posledně ladění vzhledu pro mobilní zařízení).

Následující kapitoly budu věnovat popisu průběhu problematiky a řešení jednotlivých částí úkolů, které jsem zmínil výše.

### Výběr vhodného front-end frameworku

DOPSAT

### Analýza jednotlivých kroků

DOPSAT

### Tvorba pomocného API

Stávající jádro systému má košík řešen staticky, každý krok a každá akce je řešena přes statickou stránku a statické formuláře, takže při jakékoliv akci se načítá celá stránka znovu. Z tohoto důvodu jsem nemohl košík napojit rovnou na jejich jádro, ale musel sem si vytvořit pomocné API mimo, které bude simulovat průchod celým košíkem, umožní mi prezentovat pak plně funkční košík a průchod celou objednávkou. Po tomto schválení bude vše předáno back-end developerovi, který pak napojí mou připravenou strukturu dat na jejich jádro.

Jelikož ve firmě nejsem na pozici back-end developera a toto API má být jen dočasné potřeboval jsem zvolit co nejefektivnější řešení. V minulosti jsem měl zkušenosti s PHP frameworkem Laravel, což je MVC (model-view-controller) framework, který spoustu věci vyřeší za mě. Má implementováno routování, zpracování requestu od klienta, jednoduchou práci s databázi pomocí Eloquent ORM a velmi dobře si rozumí s datovou strukturou JSON, jelikož veškerý výstup je v tomto formátu již defaultně, toto mi vyhovuje, jelikož klientská část aplikace bude přijímat a odesílat veškerá data v JSON struktuře a dále si je zpracovávat do šablon a výsledného výstupu na straně klienta.

#### API - migrace

Tento framework v sobě má integrovaný program s názvem Artisan, který celý proces výroby ještě zrychli, jelikož umožňuje například vytvářet jednotlivé předpřipravené soubory pomocí jednoduchých příkazů v konzoli.

Například jako první bylo zapotřebí vytvořit jednotlivé tabulky do databáze. To se zde dělá pomocí migrací, což jsou jednoduché PHP soubory, které vytvoříme příkazem: php artisan create:migrate create\_vouchers\_table. Toto nám vytvoří migraci pro vytvoření tabulky s názvem vouchers. Tento soubor pak otevřeme a doplníme metodu create() o potřebné sloupce tabulky, což by pro tabulku voucher vypadalo takto:

DOPLNIT KOD

Tímto postupem jsem vytvořil všechny potřebné migrace pro potřebné tabulky, které jsou: products, variants, vouchers, deliveries, deliveries\_places, payments. Po vytvoření všech migrací pak jen stačí spustit příkaz: php artisan migrate, tímto jednoduchým způsobem je databáze zmigrována a připravena k použití.

#### API – model

Pro každou takto vytvořenou tabulku je dále nutno vytvořit model. Model nám reprezentuje každou tabulku a operace nad ní. Například model pro tabulku vouchers vytvoříme příkazem: php artisan make:model Voucher. Model se musí jmenovat stejně jako tabulka, jen v jednotném čísle. Vytvořenou třídu modelu můžeme ponechat prázdnou, nebo zde lze nastavit například různé gettery a settery pro úpravu vybraných hodnot sloupců, můžeme definovat sloupce, které se mají ignorovat, nebo lze nastavit jakou tabulku má model brát, pokud nechceme tabulku odvozovat z názvu modelu.

Pro účely API mi postačí pro každou tabulku prázdný model, jelikož každý model dědí veškeré základní operace z rodičovské třídy Model a to pro naše API postačí. Nyní můžeme například pro vytažení všech voucherů použit tento kód: Voucher::all(); a framework se o všechno postará za nás.

#### API – kontrolér

Databázi a veškeré operace nad ní máme připraveny, takže už nám jen zbývá vytvořit logiku mezi tím, k tomu nám slouží takzvaný kontrolér. Kontrolér si můžeme představit jako prostředníka mezi modelem a view. View slouží pro zpracování dat do šablon, toto v API používat nebudu, jelikož budu pracovat pouze s čistými daty v JSON formátu.

V kontroléru se tedy zpracovávají data z modelů a posílají dále na výstup, nebo naopak se zde zpracovávají data z dotazů a ukládají dále do databáze. Jako vše v tomto frameworku si i kontrolér můžeme vytvořit artisan příkazem, v tomto případě make:controller. Pro účel API mi postačí jeden kontrolér, který jsem nazval ApiController. Jednotlivé metody tohoto kontroléru dále reprezentují jednotlivé akce, v mém případě akce košíku. Například chci vypsat všechny metody platby, takže metoda v kontroléru by vypadala nějak takto:

DOPLNIT OBR8ZEK

Na příkladu kódu můžeme vidět, že si pomocí Elloquentu načteme z databáze všechny metody platby upravíme si je na potřebnou strukturu dat, na jakou ji budeme potřebovat na klientovi a pak jí vrátíme na výstup jako JSON. Na tomto stejném principu bude fungovat i zbytek našeho API, ať už pro přidání produktu do košíku, jeho odebrání, změnění počtů kusů, nebo vypsaní jednotlivých doprav.

#### API – router

Nyní, když je připravena databáze a veškerá potřebná logika, zbývá už jen aby při dotazu na konkrétní adresu zavolat konkrétní kontrolér s konkrétní metodou. K tomu je již v Laravelu připraven router. Což znamená že například při požadavek GET na adresu /cart/payments/get, chceme aby se nám zavolala metoda getPayments() v kontroléru ApiController.php. Takže výsledná routa bude vypadat takto:

Route::get('cart/payments/get', 'ApiController@getPayments');

### Založení projektu pro klientskou část

Po vytvoření API jsem se již mohl pustit do vytvoření samostatného JavaScript průvodce nákupem. Jako první bylo potřeba rozložit projekt a určit si strukturu adresářů. V projektu budu používat NPM, takže jsem si inicializoval soubor package.json a definoval zde všechny potřebné dependencies a skripty které budu v projektu používat, což je samotný Vue.js framework, funkci lodash.findindex, která mi pomůže jednoduše vyhledávat konkrétní prvek v poli objektů pomocí libovolného atributu objektu a jako poslední Laravel-mix, což je nadstavba pro Webpack. Po definování všech potřebných dependencies jsem spustil konzolový skript npm install, který mi do projektu stáhl všechny tyto dependencies a dependencies, které jsou na nich závislé.

V tomto projektu budu používat takzvaně nový JavaScript, tento pojem budu referovat v celé práci a nebude se jednat o nic jiného, než JavaScript ES6 a výše. Jelikož bude potřeba mít aplikaci kompatibilní i ze všemi starými webovými prohlížeči, budu muset skripty kompilovat dolů na JavaScript ES5. S tímto mi pomůže Babel a Webpack, který je součástí již nainstalovaného Laravel-mix. Tento nástroj mi pomůže i s kompilací SASS souborů do výsledného CSS. Jediné, co mi již chybí je definování vstupního JavaScript a SASS souboru a nalinkování jej v laravel-mixu, aby aplikace věděla, která soubory má aplikace kompilovat a do jakého výstupního adresáře. Laravel-mix mi toto velmi usnadňuje a k nastavení tomu všemu mi stačí pouze těchto pár řádku scriptu:

DOPLNIT KOD

Můžeme vidět že Laravel-mix nám toto velmi usnadňuje, stačí zadat typ zdroje a určit jako parametry vstupní a výstupní soubor. Tímto by byl projekt připraven a mohl jsem se pustit do samotné tvorby aplikace.

### Úvodní stránka

Jako první jsem si vytvořil hlavní stránku. Hlavní stránka bude jako jediná statická stránka tohoto projektu. Bude tvořena jednoduchým php souborem, zbytek stránky jako rychlý detail produktu, celý košík a průchod objednávkou budou pak dynamické komponenty přidávány JavaScriptem. Tato stránka by měla byt velmi jednoduchá a bude zde jen výpis tří produktů. Produkty se načítají z mnou vytvořeného API pomocí php funkce file\_get\_contents() a následně se pomocí funkce json\_decode() převedou do PHP objektu a cyklem vypíšou do stránky.

### Hlavní .js soubor aplikace

Celá front-end aplikace začíná v souboru app.js. Zde si jako první naimportuji všechny potřebné frameworky a JavaScripty, které jsem si pomocí npm již stáhl do projektu. Budu používat modulový přístup z nového JavaScriptu. Takže jednotliví části aplikace budou rozloženy do jednotlivých souborů (modulů). Pokud budeme chtít následně s konkrétním modulem pracovat v každém souboru ho bude potřeba importovat. Například jako první potřebuji naimportovat Vue.js framework, na jehož základu bude celá aplikace fungovat. Import se provádí následovně: import Vue from 'vue'; Následně jsem si připravil a naimportoval další komponenty a moduly, kterým se budu jednotlivě věnovat v dalších kapitolách.

Jako další jsem si definoval pár globálních proměnných a objektů. Globální proměnné jsou v JavaScriptu ty, které jsou navázány na object window a jsou přístupné z celé aplikace, aniž by je bylo nutno je dále v každém modulu importovat. Globální proměnné by se neměli ve velké míře definovat, jelikož hrozí přepsaní již globálních funkci a proměnných nebo nám je může přepsat nějaká jiná knihovna, která se může v aplikaci objevit. Nicméně při dobrém uvážení může být pár takových proměnách a objektů velmi užitečných, zejména pokud na ně referujeme v aplikaci často. V mojí výsledné aplikaci jich mám čtyři. Jedna je pro samotný Vue framework, abych ho nemusel v každé komponentě definovat znovu a dále mám jako globální definovanou flash message, překladovou funkci a EventBust, ke kterým se ještě budu dále v práci věnovat.

Dále si zde definuji 2 základní Vue instance. Jednu, která bude definovat celou aplikaci a druhou, která bude sloužit jako sběrnice pro události. Sběrnice události bude sloužit jako prostředek, kterým budou mezi sebou komunikovat Vue komponenty, které nejsou ve vztahu rodič/potomek. V instanci aplikace si pouze definuji HTML root element naší aplikace což je div#root, a vytvořím objekt components, do kterého postupně během tvorby aplikace přiřadím jednotlivé komponenty aplikace.

Jako poslední v tomto souboru přiřadím click eventy na tlačítka jednotlivých produktů, zde použiju klasická JavaScript přístup, jelikož produkty nebudou dynamickou komponentou, ale budou se staticky vypisovat za pomocí PHP do stránky, a to z toho důvodu že veškeré výpisy produktů bude nutno indexovat z důvodu SEO optimalizace. V obrázku níže je ukázka klasického přidání click eventu v JavaSciprtu.

Tímto je vstup aplikace hotov a můžu se pustit do tvorby samotných komponent aplikace.

### Vue.js komponenty obecně

Komponenta reprezentuje každou část aplikace a dále každá komponenta může mít své další pod komponenty. V mém případě bude komponenta cely nákupní košík, který bude mít své pod komponenty, to jsou jednotlivé kroky košíku a ty můžou mít i další pod komponenty. Všem těmto komponentám se budu věnovat v dalších kapitolách práce.

Každá komponenta Vue má svůj vlastní soubor s příponou .vue a je tvořena ze tří části. První část je šablona, což je vlastně HTML kod který je uzavřen do dvojice značek <template></template>, dále je to samotný skript aplikace a v poslední řadě stylopis, který není povinný a v projektu ho ani používat nebudu, a budu volit klasický přístup, kde veškeré SASS soubory budou ve svém vlastním souboru a to z důvodu že nejsem příznivcem mícháni skriptů a stylopisů dohromady a navíc v nynějším jádře firemního systému, je už zavedený určitý code style pro styly.

Skript samotné aplikace se skládá z několika částí, hlavní část tvoří data objekt, zde se uchovávají veškerá data aplikace. Tyto data jsou reaktivní, to znamená že pokud změníme data v tomto objektu změny se okamžitě projeví i ve veškerých šablonách a pod komponentách, aniž bychom je museli znovu aktualizovat ručně. Toto nám značně urychli vývoj celé aplikace, jelikož klasickým JavaScript přístupem bychom si museli ručně najit v DOMu element, který chceme aktualizovat a pokaždé znovu aktualizovat data elementu.

Dále komponenta může obsahovat watcher, kterým můžeme sledovat stav konkrétních dat, a při každé změně můžeme provést nějakou akci.

Pokud si v komponentě potřebujeme definovat vlastní metody, můžeme si je definovat do objektu methods. Komponenta má definované i pár hook metod, které se vyvolávají na základě stavu komponenty. Například metodu created(), která se spustí vždy při vytvoření komponenty nebo activated(), která se spustí při přechodu komponenty do aktivního stavu.

DOPLNIT OBRAZEK PRAZDNE KOMPONENTY

### Komponenta – rychlý detail produktů

Při výpisu produktů, jsem vymyslel tlačítko „Rychlý nákup“, což bude sloužit pro rychlý výpis produktu s nejzákladnějším popisem, aniž bychom se museli prokliknout do celého detailu. Tento produkt se načte do lightboxu kde bude hlavní fotka samotného produktu, krátký popis, kód, výrobce a výběr varianty, na základě které se bude dynamicky měnit cena a počet kusů na skladě. V neposlední řadě bude pole pro zvolení počtů kusů a tlačítko přidat do košíku.

V hlavním app.js mám již navázány click eventy na data aplikace.