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

## Generování XML feedu

Abych se ze začátku seznámil se strukturou již běžícího projektu, mým prvním úkolem bylo vygenerování jednoduchého XML feedu pro slovenské srovnávače cen Pricemania.sk a Najnakup.sk. Oba tyto srovnávače cen mají na svých webových stránkách specifikovanou strukturu XML feedu, kterou vyžadují. Podle tohoto dokumentu jsem musel daný XML feed generovat, aby byl srovnávačem přijat a správně zpracován. Jelikož postup řešení je pro oba srovnávače cen téměř totožný, budu popisovat postup řešení pouze pro srovnávač Pricemania.sk.

## Cenový automat

Mým druhým úkolem byla implementace Cenového automatu. Cenový automat je webová aplikace, která slouží k automatickému upravování cen na straně e-shopu pro zajištění vyšší návštěvnosti ze srovnávačů zboží. Obdržel jsem předpřipravený zdrojový kód, ve kterém jsem měl za úkol doprogramovat funkcionalitu odesílání cen produktů z e-shopu do Cenového automatu a nastavení nových cen produktů v e-shopu odeslaných Cenovým automatem. Druhou částí tohoto úkolu bylo vytvoření nastavení v administraci e-shopu, které umožní cenový automat nakonfigurovat.

## Dárky k objednávce

Dárky k objednávce byl můj třetí a zároveň první komplexnější úkol. Měl jsem za úkol vytvořit plugin, ve kterém bude možno vytvořit dárky k objednávce, které zákazník dostane zdarma při nákupu nad určitou cenu. Každému dárku bylo potřeba nastavit název, který slouží jako interní pojmenování pro administrátora e-shopu, minimální hodnotu objednávky, ke které se dárky přidají a samotné dárky, které se vybírají z produktů nabízených e-shopem k prodeji.

## Platební brána GP webpay

Dostal jsem za úkol implementovat platební bránu GP webpay, která slouží pro online platby kartou. Od kolegy jsem obdržel implementační manuál a přístupové údaje k účtu GP webpay včetně testovací platební karty, abych mohl provádět testovací platby pro ověření správného fungování platební brány. Podle manuálu jsem musel zákazníka přesměrovat na URL adresu platební brány se správnými parametry a následně výsledek platby přijmout, zpracovat a zobrazit jej zákazníkovi.

# Postup řešení 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.

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 dotazů 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 se již můžu pustit do vytvoření samostatného průvodce.