Mendelova univerzita v Brně
Provozně ekonomická fakulta

Komplexní webové administrační rozhraní pro firmu IdeaTech

Diplomová práce

Vedoucí práce:

Ing. Jiří Lýsek, Ph.D. **Bc. Jan Vodák**

Brno 2017

Rád bych zde poděkoval Ing. Jiřímu Lýskovi, Ph.D. za možnost vypracování této práce pod jeho odborným vedením a za jeho cenné rady nejen při dohledu nad danou prací, ale i v průběhu celého mého studia na Provozně ekonomické fakultě Mendlovy univerzity. Dále mé poděkování patří společnosti IDEATECH s.r.o., jmenovitě Petru Vojáčkovi a Adamovi Kyselovi za cenné rady a aktivní přístup při konzultacích od samotného počátku zpracování tohoto projektu. Celé společnosti patří poděkování za poskytnutou příležitost vypracovat toto téma pod záštitou jejich jména a za vloženou důvěru ve mě.

Čestné prohlášení

Prohlašuji, že jsem tuto práci: **Komplexní webové administrační rozhraní pro firmu IdeaTech**
vypracoval/a samostatně a veškeré použité prameny a informace jsou uvedeny v seznamu použité literatury. Souhlasím, aby moje práce byla zveřejněna v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů, a v souladu s platnou *Směrnicí o zveřejňování vysokoškolských závěrečných prací.*

Jsem si vědom/a, že se na moji práci vztahuje zákon č. 121/2000 Sb., autorský zákon, a že Mendelova univerzita v Brně má právo na uzavření licenční smlouvy a užití této práce jako školního díla podle § 60 odst. 1 Autorského zákona.

Dále se zavazuji, že před sepsáním licenční smlouvy o využití díla jinou osobou (subjektem) si vyžádám písemné stanovisko univerzity o tom, že předmětná licenční smlouva není v rozporu s oprávněnými zájmy univerzity, a zavazuji se uhradit případný příspěvek na úhradu nákladů spojených se vznikem díla, a to až do jejich skutečné výše.

V Brně dne 18. května 2017 _____

Abstract

Vodák, J. Complex web administration system for IDEATECH company. Diploma thesis. Brno: Mendel University, 2017.
The thesis deals with the design and implementation of a complex web administration system for IDEATECH company, focusing on the universality of solution and the possibility of its repeated uses across various company projects. The design and subsequent implementation are based on the company's written requirements.

Keywords

Administration system, CMS, modular system, Nette framework, PHP, MVC, Bootstrap, UML language, e-shop, advanced web technologies.

Abstrakt

Vodák, J. Komplexní webové administrační rozhraní pro firmu IDEATECH. Diplomová práce. Brno: Mendelova univerzita v Brně, 2017.
Práce se zabývá návrhem a následnou implementací komplexního webového administračního rozhraní pro společnost IDEATECH s.r.o., se zaměřením na univerzálnost navrhovaného řešení a možnosti jeho opakovaného využití napříč různými projekty společnosti. Návrh a následná implementace vychází ze sepsaných požadavků společnosti.

Klíčová slova

Administrační systém, CMS, modulární systém, Nette framework, PHP, MVC, Bootstrap, jazyk UML, internetový obchod, pokročilé webové technologie.

Obsah

1 Úvod 11

2 Cíl práce 12

3 Technologie pro tvorbu webových aplikací 13

3.1 Základní nástroje pro tvorbu webových aplikací ... 13

3.2 Frameworky ... 15

3.2.1 Typy frameworků a jejich hodnocení ... 15

3.2.2 Příklady PHP frameworků ... 17

3.3 Architektura MVC ... 19

3.4 Pokročilé vývojářské nástroje ... 20

3.5 Metodiky a přístupy při vývoji systému ... 23

4 Metodika 26

5 Specifikace požadavků společnosti IDEATECH s.r.o. 27

5.1 Neformální specifikace požadavků ... 27

5.2 Formální specifikace požadavků ... 30

5.2.1 Formální specifikace správy obsahu a nastavení ... 30

5.2.2 Formální specifikace modulu pro internetový prodej ... 32

6 Analýza požadavků 35

6.1 Volba technologie ... 36

6.2 Funkční a nefunkční požadavky ... 37

6.3 Analýza rizik ... 38

7 Návrh systému 41

7.1 Rozvržení do modulů ... 44

7.1.1 Core modul ... 44

7.1.2 CMS modul ... 48

7.1.3 E-shop modul ... 49

7.1.4 Front modul ... 51

7.2 Návrh databáze ... 54
7.3 Návrh vzhledu administrační sekce ... 58

8 Implementace a zavedení rozhraní 60
8.1 Implementace ... 61
8.1.1 Autowiring a tvorba komponent ... 61
8.1.2 Dashboard ... 63
8.1.3 Uživatelé, nastavení oprávnění a aktivnosti modulů ... 65
8.1.4 Nastavení měn a nákupní proces na frontendu ... 69
8.2 Testování ... 71
8.3 Praktické využití systému ... 73
8.4 Ekonomické zhodnocení projektu a doba návratnosti ... 75

9 Závěr 77

10 Literatura 79
10.1 Knižní zdroje ... 79
10.2 Internetové zdroje ... 80

11 Seznam obrázků 82

12 Seznam tabulek 83

A Kompletní zadání požadavků společnosti 85

1 Úvod

Na současných, moderních a rychle se rozvíjejících konkurenčních trzích vzniká a zaniká velké množství společností, jejichž úspěch se odvíjí od mnoha faktorů. Jedním z těchto faktorů může být prezentace firmy s využitím internetových technologií, která se stala již její nezbytnou součástí. Doby, kdy stačila jednoduchá, statická webová stránka, jsou již dávno minulostí. Naprostá většina firem nyní využívá pokročilé webové aplikace, poskytující vlastníkům pestrou škálu možností, jak se svou webovou prezentací pracovat. Internet jako takový se postupem času stal velmi silným nástrojem, obsahující obrovský potenciál, a pokud jej firmy dokáží správně využít, mohou posílit svou stabilitu a získat konkurenční výhodu oproti ostatním účastníkům trhu.

Prezentace firem, potažmo jednotlivců skrze internet, je de facto nikdy nekončící proces, který by se neměl po dokončení jedné fáze považovat za hotovou záležitost, ale naopak by se do této oblasti mělo dále investovat a ke každému procesu přistupovat s naprostou profesionalitou od samého počátku, nehledě na to, jestli se jedná o počáteční návrh, design, marketingovou propagaci nebo další rozvoj webu. Z těchto aspektů je patrné, že jeden člověk sám nedokáže pokrýt celý tento proces, a proto je potřeba více odborníků ve své profesi, kde každý přispěje k výsledku svým dílem.

Jednotlivé webové prezentace se mohou velmi lišit, vzhled často podléhá firemní identitě, společnosti se od sebe snaží odlišovat, čímž se mohou dostávat více do povědomí zákazníků, tudíž v této oblasti mohou UX designéři a grafici, zabývající se danou problematikou, prokázat své kreativní schopnosti. Avšak existují i jisté požadavky, které se pravidelně opakují, což může být dáno některými trendy dnešní doby. Pro firmy, které se zabývají vývojem webových prezentací a systémů na jejich správu, je proto často zbytečné pro každou zakázku vyvíjet nový systém na správu těchto stránek, ale naopak je výhodné poskytnou již své hotové řešení, které se dá lehce přizpůsobit a rozšířit o nové, specifické požadavky. Tato hotová řešení bývají součástí firemního know-how, což se dá považovat za další z možných faktorů ovlivňující úspěšnost firmy. Může tedy být velmi důležité, aby každá společnost, zabývající se danou problematikou, dokázala poskytnou své vlastní řešení. Dále je také nezbytné, aby tyto společnosti dokázaly správně a v čas reagovat na dynamické změny v okolí, zahrnout tyto změny do svých postupů a snažit se je využít ve svůj prospěch, čímž by mohly poskytovat lepší řešení.

Existují i již hotová řešení, která jsou volně dostupná k užití, avšak tato řešení nemusejí být pro spousty zákaznických firem dostačující. Dokáží sice poskytnout nezbytný základ, ale pro specifické požadavky klientů se stávají nepoužitelná, a to hlavně z důvodu špatné rozšiřitelnosti kódu a jeho budoucí nemožnosti jednoduché údržby. Naopak pro malé a teprve se rozvíjející společnosti je volně dostupné řešení systému pro správu stránek vhodným řešením, a to hlavně z důvodu úspor financí nebo možnosti jednoduché implementace. Pokud se společnosti tato řešení v budoucnu osvědčí a zjistí, že poskytují produkt, o který je mezi zákazníky zájem, můžou požadovat vytvoření specifického systému na míru.

2 Cíl práce

Hlavním cílem této práce je poskytnou sjednocené komplexní administrační řešení pro opakující se požadavky zákazníků na správu webové aplikace, které by opakovaným užitím ušetřilo čas a hlavně finanční prostředky společnosti vynaložené při budoucím vývoji. Tyto úspory by měly být v rámci této práce vyhodnoceny jak z hlediska technické, tak i ekonomické stránky a následně by měly být doporučeny i další možnosti budoucího vývoje. Výsledný systém bude použit a otestován na konkrétním případě.

Opakující se požadavky na systém budou převzaty ze sepsaného zadání společností IDEATECH s.r.o., pro kterou je tento systém určen. Tyto požadavky budou analyzovány, na základě výsledků této analýzy bude navrhnut a později implementován výsledný systém se zaměřením na univerzálnost využití napříč různými projekty společnosti. V rámci implementace systému bude zvolena optimální technologie, vycházející z potřeb společnosti a tato technologie bude pospána v rešerši nástrojů pro tvorbu systémů dané problematiky.

Jedna z klíčových myšlenek je také zajištění možnosti budoucí jednoduché rozšiřitelnosti systému za pomocí využití modulů. Tímto krokem bude zajištěna modernizace existujících firemních řešení, která mohou být již zastaralá a jejichž momentální rozšiřitelnosti je značně omezena jednak z důvodu zmíněné zastaralosti, nebo kvůli špatně udržovanému kódu. Ale i přesto by bylo vhodné tato stará řešení prozkoumat a určit, zda některé z nich bude možné zachovat a sjednotit do nové, ucelené verze.

3 Technologie pro tvorbu webových aplikací

Úplně na začátku by měl být objasněn základní rozdíl mezi dvěma podobnými pojmy, které někteří lidé často zaměňují. Těmito pojmy jsou **webová stránka** a **webová aplikace**. Webová stránka (prezentace) je formátovaný dokument, který je možné zobrazit skrze webový prohlížeč. Formátovávání je nejčastěji zajištěno pomocí hypertextu, což je kombinace značek jazyka HTML, nebo XHTML. Obsah stránek je tvořen textem, odkazy umožňujícími přechod mezi jednotlivými stránkami a multimediálními daty – obrázky, videi, zvukem. Cílem webové prezentace je ovlivnit, či změnit chování určité skupiny lidí – prezentuje určitý produkt, nebo službu a je často kanálem pro prodej.

Webová aplikace je naopak **služba,** poskytována prostřednictvím webového serveru a internetového prohlížeče, který slouží jako klient a nezná podrobnosti o vnitřní logice služby. Manipulace s aplikací a schopnost šířit ji bez nutnosti instalace softwaru mezi uživateli tvoří hlavní výhodu této služby. Řeší určitý problém prostřednictvím sebe sama. Není kanálem pro prodej produktu, ale přímo produktem. Cílem webového designéra je vytvořit nový návyk – aplikace tedy zapadne do života člověka, který ji používá. S těmito dvěma pojmy ještě souvisí jeden pojem, a to e-shop. E-shop prodává produkty a služby online – cílem e-shopu není jen prezentace produktů, ale především jejich přímý prodej (Řezáč, 2014).

3.1 Základní nástroje pro tvorbu webových aplikací

Základním nástrojem, nebo spíše jazykem pro tvorbu webové aplikace, bez kterého se neobejde žádný vývojář webových aplikací, je hypertextový značkovací jazyk, zkráceně ***HTML*****,** což je označení pro anglický výraz *Hypertext Markup Language*. Tento jazyk umožňuje propojit velké množství oddělených informací a vytvořit tak nové uspořádání pro existující informace. To má z následek, že v jednom dokumentu mohou být vedle sebe umístěny informace z oddělených zdrojů. Jejich propojení je dosáhnuto pomocí elementů, díky kterým se mohou vytvářet, formátovat nebo upravovat webové stránky. Jedná se například o nastavení vzhledu textu, vkládání obrázků a tabulek nebo používání rámů pro umístění jednotlivých informací (Písek, 2014).

Doplňkem pro tento jazyk, který se využívá pro konkrétní nastavování stylů a prvků HTML dokumentu, jsou kaskádové styly, neboli ***CSS*** (*Cascading Style Sheets*). Tento doplněk se využívá hlavně pro oddělením kódu využitého pro definování vzhledu od obsahu dokumentu, což má za následek zpřehlednění výsledného kódu. Jedná se o soubor pravidel a předpisů, které umožňují formátovat dokumenty, definují způsob prezentace v jednotlivých koncových zařízeních, určují vzhled a styl jednotlivých prvků dokumentu HTML a XHTML, ale jako takové do struktury nezasahují, což je jejich hlavní výhoda. Jedná se tedy o samostatný, doplňující jazyk k HTML, jehož vytvořené styly se aplikují na obsah dokumentu webových stránek (Druska, 2006).

Koncepce hypertextu je známa více než 60 let. V červenci 1945 napsal Vannevar Bush článek „*As We May Think*“ pro *Atlantic Monthly*, ve kterém popsal systém pro „*prohlížení a pořizování poznámek z rozsáhlých textů a grafiky*“. Avšak první verze HTML jazyka (HTML 1.0) byla objevena až v roce 1990 a neoficiální verze HTML + byla představena ve druhé polovině roku 1993. Tato verze sice obsahovala elementy pro práci s obrázky, formuláři a tabulkami, ale s formátováním textu a odstavců si zatím neporadila. V současné době se využívá verze ***HTML 5.1***, z čehož lze usoudit, že s postupem času se vydávaly nové verze, které reagovaly na požadavky uživatelů, redukovaly počty zbytečných elementů a naopak přidávaly nové elementy a funkcionalitu (Písek, 2014). Z výše uvedeného popisu je patrné, že tento jazyk má široké využití, avšak existují i různá omezení, se kterými si neporadí, a je proto potřeba si uvědomit, že pro vytvoření některých funkcí, kterými dnes běžně webové stránky disponují, si pouze s HTML nelze vystačit. Není zde možné vytvořit ověřování uživatele, HTML je určeno pro statický obsah dokumentů nebo nelze vytvořit dynamicky se měnící nabídky, vysouvací menu a nic podobného (Písek, 2014).

Předchozí zmíněné nedostatky jazyka HTML řeší programovací jazyk ***JavaScript*** od společnosti *Netscape*. Jedná se o jednu z nejrozšířenějších technologií vůbec a de facto jediný skutečný programovací jazyk používaný v rámci HTML dokumentů. Přestože začal jako jednoduchý jazyk, který nacházel uplatnění ve validaci formulářů nebo drobné manipulaci s obsahem stránky, tak se velmi vyvinul, a proto je v něm dnes možné vytvářet bohaté klientské aplikace. Během několika prvních let své existence dokázal odsunout na vedlejší kolej svého jediného konkurenta *VBSCript*, navrženého společností *Microsoft,* a kromě toho v průběhu tohoto období také začal do jisté míry nahrazovat zásuvný modul *Flash* (Pehlivanian, 2014).

Jazyk JavaScript se postupem času stal silným a pokročilým vývojářským nástrojem, avšak ne všichni návrháři, působící v oblastí daných technologií, jsou schopni v něm programovat na dostatečně vysoké úrovni, a proto se obracejí na nejrůznější knihovny, které by jim pomohly s běžnými úkony a navíc by jim ušetřily i čas. Pro ulehčení práce z těchto důvodu vznikla všestranná a volně dostupná knihovna ***jQuery***, využívající spoustu koncepcí z jazyků HTML a CSS. Tato knihovna poskytuje víceúčelovou abstraktní vrstvu pro běžné webové skriptování a je užitečná skoro ve všech situacích, v nichž je potřeba skriptovat. Jedná se například o přístup k elementům dokumentu skrze model *DOM* (*Document Object Model*), o změnu obsahu dokumentu, schopnost reagovat na akce uživatele, animovat změny v dokumentu, ale hlavně usnadňuje používání technologie ***Ajax***[1] (Chaffer, 2013).

Popularita JavaScriptu neklesala, ale naopak stále rostla, na což začaly reagovat i velké společnosti jako třeba Google, který se rozhodl pro svůj vlastní prohlížeč

[1] Ajax – asynchronní JavaScript a XML (*Asynchronous JavaScript and XML*) umožňuje aktualizovat obsah stránky bez nutnosti obnovení, zažádat a získávat data ze serveru po načtení stránky a odesílat mu jej zpět na pozadí

Chrome vytvořit zbrusu novou a výkonnou implementaci jazyka, čímž dochází k oživení myšlenky, že lze jazyk využít nejen v rámci klientského skriptování, ale lze jej využít i pro programovaní serverového kódu. Tyto úvahy v roce 2009 ústí v založení neformální organizace ***CommonJS***, která si stanovila za cíl prozkoumat a standardizovat chybějící serverová rozhraní. Ještě v tomtéž roce se ale objevuje jiný projekt - ***Node.js,*** který přichází se sadou rozhraní a knihoven obalující nově vytvořenou implementaci jazyka a jednoduše tím dovoluje spouštět JavaScriptový kód i mimo webový prohlížeč. Node*.js* si velmi rychle získal silnou uživatelskou základnu, zastínil CommonJS a stal se tak de facto standardním řešením pro serverové vykonávání JavaScriptu (Žára, 2015).

Dalším velmi důležitým jazykem je jazyk ***PHP***. Na tento jazyk nesmí být zapomenuto, jelikož se jedná o jeden z nejvíce rozšířených jazyků na straně serveru v oblasti webového vývoje a je na něm založena většina projektů, téměř 78,9% všech webových aplikací. Jeho počátky spadají do stejného roku jako u jazyku JavaScript, tedy do roku 1995, ale na rozdíl od něj bylo jeho hlavní náplní od samého počátku řešit úlohy běžící výhradně na straně serveru. PHP jazyk vytvořil Rasmus Lerdorf pod názvem „*Personal Home Page (Tools)*“, což bylo přeloženo jako „*Nástroj pro osobní domovské stránky*“, ale nyní se jedná spíše o rekurzivní zkratku „*PHP: Hypertext Preprocessor*“, česky *Hypertextový Preprocesor PHP*. Hlavním cílem jazyka PHP je tedy zmíněné zpracování dat tak, aby je bylo možné dynamicky zobrazovat do webových stránek. Konkrétně se jedná o matematické výpočty, převádění formátů dat a spolupráce s databázemi. Dále umožňuje vývojářům vylepšit statické stránky o reakce na uživatelské požadavky. Jazyk PHP se úzce specializuje na webový vývoj, a proto představuje obvyklou volbu pro tyto vývojáře (Hopkins, 2014).

3.2 Frameworky

Při vývoji jakékoliv aplikace či systému je vhodné zvážit možnost využití již existujících frameworků. Jedná se o softwarovou strukturu, která slouží jako podpora při tvorbě a vývoji dané aplikace. Může obsahovat podpůrné knihovny nebo třídy, případně jiné užitečné nástroje.

3.2.1 Typy frameworků a jejich hodnocení

Jak je možné odhadnout z podnadpisu, existuje více druhů frameworků, týkajících se webových prezentací či aplikací. Již byla zmíněná knihovna jQuery, jež sama o sobě není frameworkem, nýbrž jen zmíněnou knihovnou, a může být ve frameworcích obsažena. Frameworky se dají rozdělit na **frontendové** a **backendové**. Souvisí to s rozdělením aplikace, kde pojem frontend představuje klientskou část, která běží na straně klienta a je běžně viditelná. Naproti tomu backednová část aplikace, neboli serverová část, se stará o administraci klientské části a nachází se na straně serveru. Prostým a jednoduchým tvrzením se dá říci, že frontendové frameworky se starají o způsob zobrazování a prezentování informací, tedy o problémy uživatelského rozhraní. Backendové mají naopak za úkol se starat

o serverovou část aplikace, běžící v pozadí, jehož součástí je potom i způsob zobrazování. Samozřejmě se dají různě kombinovat a používat současně.

Asi nejznámějším **frontendovým** HTML, CSS a JavaScriptovým frameworkem, napomáhajícím při tvorbě uživatelského rozhraní, je **Bootstrap**, který byl původně vyvinut pro potřeby *Twitteru* a někteří jej nazývali „*responzivní framework*“. To však již není příliš přesné, protože zvládne pokrýt mnohem širší škálu problémů, mezi které se dá responsivita webu zařadit. Bootstrap je de facto sada nástrojů, jež má za úkol usnadnit práci s tvorbou šablony, napomáhá s úpravou typografie a celkově s tvorbou elementů uživatelského rozhraní a zároveň ošetřuje zobrazování napříč všemi platformami. Pod tímto názvem se v dnešní době mohou skrývat také technologie jako jsou *React*, *Angular*, *Vue* atp., avšak zde se pojednává o frameworku pro tvorbu *GUI* (*Graphical User Interface*). Mezi hlavní výhody těchto typů frameworků patří (Málek, 2013):

- Rychlý vývoj typizovaných webových prezentací pomocí znovupoužitelných komponent, což usnadňuje začátky projektů
- Usnadnění vývoje a úbytek starostí programátorům s frontendovou částí
- Sjednocení vizuálního jazyka napříč celým projektem, tzn. že grafik, designer, kodér a programátor využívají sjednocené zobrazování elementů

Podle výše zmíněného rozdělení frameworků bude dále v textu pojednáváno o frameworcích patřících svým zařazením do **backendové** části. Hlavní výhodou těchto frameworků je, že pokud uživatel dodrží doporučené postupy a návrhové vzory, může se oprostit od vedlejších, rušivých činností nebo problémů a může se soustředit na hlavní činnost, čímž si usnadní práci a vývoj. Dalšími neméně důležitými výhodami mohou být zajištění bezpečnosti a dodržení správných programátorských standardů, čímž se zpřehledňuje kód a předchází se opakovanému psaní duplicitního kódu. Aplikace využívající framework se stává snadno udržovatelná a rozšiřitelná nejen pro samotného vývojáře, ale i pro ostatní uživatele. Mezi další výhody použití frameworku patří (Monus, 2015):

- Rychlejší vývoj
- Dobře organizovaný, znovupoužitelný a udržovatelný kód
- Prosazování moderních webových vývojových postupů, jako například objektově orientované programování
- Oddělení prezentace od logiky (architektura MVC)
- Škálovatelnost kódu

Pokud se vývojář rozhodne pro využití frameworku, nepřináší to jenom řadu výhod, ale musí být zváženy i možné nevýhody. Jednou z nich je už samotné využití kódu třetích stran ve vlastní aplikaci. To má za následek hlavně riziko, že pokud se bude spoléhat jen na tento kód a jeho vývoj se v budoucnu zastaví, mohou být vystaveny riziku i všechny další projekty, kde se tento kód využívá. Nebylo by to poprvé, kdy se vývoj určitého frameworku úplně zastavil a tím de facto zaniknul. Dal-

ší nevýhoda může být naivní pocit bezpečí, kdy chybně napsaná aplikace je mnohdy více zranitelná, než když framework nevyužívá. Dá se předpokládat, že začínající programátor může mít s prvotním vývojem jisté problémy a udělá mnoho chyb. Měla by být také zvážena individuálnost projektu, jelikož užití frameworku může do značné míry programátora svazovat, a pokud by projekt byl příliš individuální či nestandardní, bude lepší se tomuto využití zcela vyhnout. Některé zdroje poukazují na snížení rychlosti aplikace využívající nějaký framework, avšak toto tvrzení je mnohdy velmi sporné.

3.2.2 Příklady PHP frameworků

Mezi nejznámější PHP frameworky lze zařadit například **Laravel**, **Symphony**, **Yii**, **Zend**, **CakePHP** nebo pro české vývojáře velmi dobře známý **Nette framework**.

I když je **Laravel** oproti některým frameworkům relativně mladý (2011), mezi PHP programátory se těší obrovské popularitě napříč celým světem. Mezi jeho hlavní přednosti patří velmi rozvinutý ekosystém s platformou připravenou k okamžitému hostování. Na jeho webu je dostupných velmi mnoho tutoriálů a rozsáhlá dokumentace v podobě komentovaných video záznamů obrazovky, které jsou nazývaný *Laracasty*. Další výhodou je jeho samotná implementace, jež disponuje spoustou vlastností, které urychlují samotný vývoj, nebo to, že obsahuje vlastní odlehčené šablonovací jádro *Blade*. Tento framework garantuje snadnou implementaci často se opakujících činností jako je například autentizace, relace a řazení do front nebo směrování. Navíc obsahuje vlastní lokální vývojové prostředí nazvané *Homested*. Tato kombinace předností z něj činní velmi silný vývojářský nástroj (Monus, 2015).

Pro vývojáře, kteří přechází z frontendového prostředí do backendového, je vhodné přiklonit se ke frameworku **Yii**, a to hlavně z toho důvodu, že tento nástroj je přímo integrovaný s JavaScriptovou knihovou jQuery a ve svém základu jej uživatel dostává se sadou funkci pro AJAX a mechanismem pro motivy a skiny. Jednou z dalších výhod je jeho generátor kódu zvaný *Gii*, tvorba prototypů nebo fakt, že je založen na kódovacím pojetí DRY – „*Don't Repeat Yourself*“, což v překladu zní „*neopakuj se*“ a jako už většina frameworků uživatele vede k čistě objektově orientovanému programování. Jeho hlavní dominantou je jeho údajná rychlost – honosí se titulem „*nejrychlejší PHP framework*“, která pramení z využití *Lazy Loadingu*, v překladu „*pohodového načítání*“ (Monus, 2015).

Pro rozsáhlé projekty se jeví jako velmi vhodné řešení **Zend Framework**, který byl vytvořen a optimalizován pro metodologii agilního vývoje systému. Mimo jiné je velmi robustní a obsahuje velmi velké množství konfiguračních voleb a možností nastavení. Mezi jeho hlavní partnery se řadí v oblasti IT velmi dobře známe firmy, jako jsou IBM, Adobe, Microsoft a Google. Mezi další jeho přednosti se řadí spousta kladně hodnocených funkcionalit – například nástroje pro kryptografické kódování, testovací nástroje *PHP Unit testing*, online ladicí nástroje a frontendový editor *Drag and Drop* (Monus, 2015).

Dalším velmi populárním mezi PHP vývojáři je framework **Symphony**, jehož komponenty jsou využity pro řadu známých projektů, jakými jsou *Drupal* nebo

mimo jiné i již zmíněný framework *Laravel*. To je dáno hlavně tím, že jeho komponenty jsou opětovně využitelné PHP knihovny, dodávané prostřednictvím nástroje pro správu závislostí v PHP projektech - *Composeru*, který bude popsán v podkapitole 3.4. Mezi tyto samostatné knihovny patří například tvorba formulářů, směrování, šablonování, autentizace nebo konfigurace objektů (Monus, 2015).

Jako poslední z PHP frameworků byl záměrně ponechán v České republice nejvíce rozšířený **Nette framework** od českého vývojáře Davida Grudla. Mezi jeho hlavní přednosti se řadí šablonovací systém, ladicí nástroje, rozsáhlé zabezpečení proti chybám a zranitelnostem nebo kvalitní česká dokumentace. Záměrně je navržen tak, aby co nejvíce usnadňoval a ulehčoval práci vývojáře.

Mezi jednu z hlavních předností tohoto frameworku je, již zmíněný ladicí nástroj **Tracy**, který je mimo jiné známý pod názvem *Laděnka*, jež se dá mimo jiné použít i jako jako samostatná komponenta v jakémkoliv soukromém projektu. Pomáhá rychleji odhalit chyby, logovat je nebo měřit čas – stará se tedy o *debuggování*. Jelikož PHP jazyk poskytuje dostatek volnosti, mnohdy se v projektu vyskytnou těžce odhalitelné chyby, o to větší roli hraje právě tento nástroj, který tyto chyby správně vizualizuje. Tracy je hojně využívána v takzvaném vývojovém prostředí, kdy programátorovi usnadňuje vývoj a napomáhá při odlaďování chyb. Naproti tomu v ostrém, produkčním režimu zůstává skrytá, chyby zaznamenává do textových souborů, nebo je umí odeslat prostřednictvím e-mailu příslušné osobě. Informace o chybách hrají velmi podstatnou roli, neboť uživatelé jsou de facto nejlepší testeři, avšak sami velmi málo kdy poskytnou zpětnou vazbu o aplikaci a její funkčnosti, takže vlastní testování aplikace, které je součástí vývoje, by nemělo být opomíjeno. Existuje i případ, kdy ladicí informace se neposílá přímo do okna prohlížeče, což se týká například ajaxových požadavků, generování XML výstupů či obrázků. Tyto chyby je možné odchytávat prostřednictvím doplňků do internetových prohlížečů jako jsou Google Chrome nebo Firefox (Nette Foundation, 2017a).

Nette využívá jako šablonovací systém **Latte**, který je navržen přímo pro PHP jazyk, jelikož z části využívá i jeho syntaxi a vychází z potřeb přímo webdesignerů. Použitím těchto šablon je PHP vývojářům usnadněna práce a zároveň je již na této úrovní provedeno zabezpečení proti zranitelnostem, například proti *XSS*[2], které, i když je jedno z nejtriviálnějších narušení, může vést i k odcizení identity. Obranou proti tomuto typu útoku je takzvané „*escapování*“ vypisovaných proměnných, což je převod znaků, majících v daném kontextu speciální význam, na jiné odpovídající sekvence. Pro časté zapomínání na tuto povinnost byl navržen automatický mechanismus, jenž disponuje technologií *Context-Aware Escaping*. Ta automaticky detekuje, ve které části dokumentu se makro nachází a podle toho zvolí správný způsob ošetření proměnných a tím zabraňuje chybám pramenícím z mylně zvoleného druhu ošetření vypisovaných dat (Nette Foundation, 2017d).

[2] XSS – Cross-site scripting je druh zranitelnosti webových aplikací, která využívá neošetřené vstupy a útočník dokáže podstrčit svůj vlastní JavaScriptový kód do stránky

Dalšími výhodami Latte, které jsou přímo uvedeny na stránkách Nette (Nette Foundation, 2017d), je využití maker, bloků, dědičnosti a používání filtrů, jež lze společně s makry i vytvářet. Filtry jsou speciálním druhem funkcí, jenž pomáhají upravit nebo přeformátovat data do výsledné podoby. Makra jsou fragmenty kódu, kterým je přiřazen identifikační řetězec a poté jsou preprocesorem nahrazeny obsahem daného makra. Navíc je v šablonách systém díky Latte velmi rychlý, neboť jednotlivé šablony se překládají do PHP kódu a poté se ukládají do cache paměti na disk, což má za následek, jako by byly psány přímo v PHP, ale jsou mnohem přehlednější a bezpečnější.

Velmi důležitá práce s formuláři je vyřešena díky vlastnímu mechanismu. Nette výrazně usnadňuje vytváření a zpracování přímo v aplikaci, data odeslaná na straně serveru se automaticky validují prostřednictvím JavaScriptu, je poskytnuto zabezpečení proti zranitelnostem a tento mechanismus počítá i s možností překladu webové stránky do více jazyků nebo s režimem pro vícenásobné vykreslování. Uživatel se také vyhne celé řadě rutinních úkolů, jakým je třeba psaní dvojí validace na straně serveru i klienta, minimalizuje se pravděpodobnost vzniku chyb a bezpečnostních děr. To vše zcela transparentně a automaticky. Dále je zabráněno útokům již zmíněného *XSS*, ověřuje se validita *UTF-8* kódování nebo jestli nejsou položky vybrané v *select boxech* podvržené (Nette Foundation, 2017b).

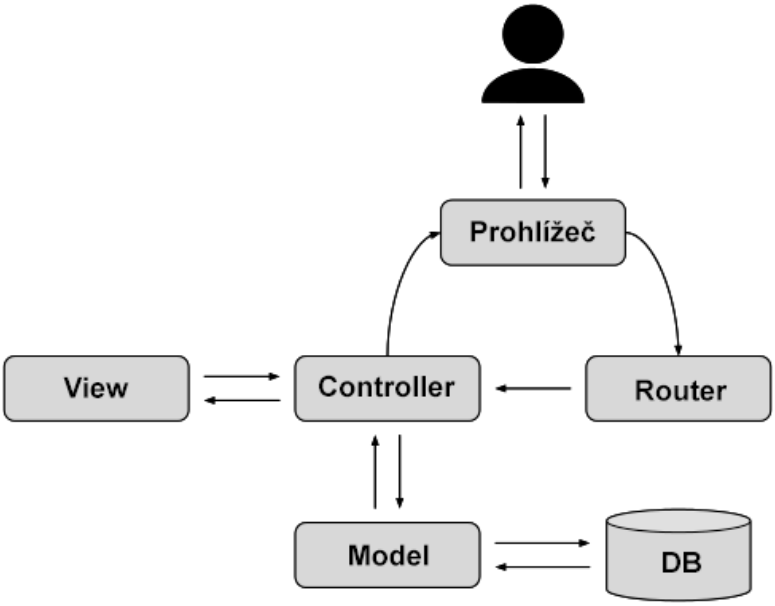
3.3 Architektura MVC

V posledních letech se začíná prosazovat vývoj založený na architektonických a návrhových vzorech, který umožňuje navrhovat software rychleji, v lepší kvalitě a znovu používat již existující zdroje. Vývojář může pracovat na vyšší úrovni abstrakce a aplikuje vzory, které popisují obecná řešení opakujících se problémů při návrhu. Vzory zapouzdřují specifickou znalost architektury a technologie, čímž pomáhají vytvářet znovupoužitelný kód. V softwarovém inženýrství se hojně vyžívá například architektura MVC. Ta byla definována již počátkem 80-tých let minulého století, avšak k rozmachu návrhových vzorů dochází až o 10 let později, a to hlavně díky zveřejnění publikace na toto téma (Bruckner, 2012).

Všechny PHP frameworky, jež byly zmíněné v předchozí podkapitole, tuto softwarovou architekturu využívají, a proto je potřeba ji zmínit také zde. MVC, celým názvem *Model-View-Controller*, nebo také *MVP*, kde *P* může znamenat Presenter (*Controller)*, se skládá ze tří základních vrstev, přičemž každá z nich je odpovědná za specifické činnosti. Architektura vznikla z potřeby oddělit u složitých aplikací s grafickým rozhraním kód obsluhy (*Controller*) od kódu aplikační logiky (*Model*) a od kódu zobrazující data (*View*, neboli pohled). Tímto se aplikace mnohem více zpřehlední, usnadňuje budoucí vývoj a týmovou spolupráci více specialistů nebo umožňuje testování jednolitých částí zvlášť.

Funkční a datový základ celé aplikace se nazývá ***Model***, jenž obsahuje celou aplikační logiku a jakákoliv akce uživatele představuje jeho akci. Vnitřní stav si spravuje sám, ven nabízí pevně dané rozhraní a voláním funkcí toho rozhraní se může zjišťovat, či měnit vnitřní stav. Model neví o existenci kontroleru ani pohle-

du. **Pohled** je naopak vrstva aplikace, která má na starost zobrazování výsledků. Obvykle využívá nějaký šablonovací systém a ví, jak se má zobrazit výsledek získaný z modelu. Celý tento průběh řídí **kontroler**, který zpracovává požadavky od uživatelů a na jejich základě volá patřičnou aplikační logiku a poté žádá pohled o vykreslení dat (Nette Foundation, 2017c). Popis procesu je znázorněn na následujícím obrázku a bude popsán na ilustrativním PHP frameworku



Obr. 1 Schéma architektury MVC

Uživatelův požadavek, který přišel formou HTTP požadavku, je zpracován skrze router a následně je předán dále do aplikace. Router je část aplikace sloužící jako obousměrný překladač mezi URL adresami a akcemi kontroleru. Dokáže tedy podle URL adresy určit, který kontroler a jaká akce se má vykonat, nebo naopak jaká URL adresa se má vygenerovat. Nyní, když je určen kontroler a akce, přichází na řadu model, jenž obstará potřebnou aplikační logiku (například dodá potřebné data nebo provede dílčí výpočty) a výsledek vrátí zpět do kontroleru. Ten tato data předá do zvoleného pohledu, který se postará o zobrazení dat a výsledek skrze okno internetového prohlížeče zobrazí uživateli.

3.4 Pokročilé vývojářské nástroje

V úvodní podkapitole byly představeny základní technologie pro vývoj webových aplikací či webových prezentací, avšak tyto technologie nemusejí být dostačující, proto existují i další technologie, které na ně navazují a vylepšují je, čímž usnadňují následnou práci při vývoji.

První z těchto pokročilejších technologií je šablonovací systém ***Jade*** (mimo jiné známý i pod starším názvem *PUG*). Vznikl původně pro *Node.js*, ale existuje

i alternativa pro PHP jazyk**.** U jakéhokoliv webu, ať už se jedná o komplikovaný, či jednoduchý, si lze usnadnit práci hned při samotném kódování. Šablony nabízejí, kromě základních věcí jako jsou proměnné, podmínky nebo cykly, hlavně také alternativní zápis HTML jazyka, který má za následek úspornější kód. Tento úsporný kód je až následně převeden do prostého HTML. Základní pravidla pro psaní v tomto šablonovacím systému jsou (Krause, 2017):

1. Ostré závorky běžné v HTML se nepoužívají. Na místo toho se píší jen názvy elementů
2. Koncové značky se vynechávají všude a zanoření je dosaženo prostřednictvím tabulátorového odsazení
3. Třídy a identifikátory se píší přímo k názvům elementu
4. Atributy se zapisují do závorek
5. Každý text na začátku je chápán jako element, připojený text k elementu je definován přes znak svislého lomítka

Existuje tedy možnost odlišného, avšak úspornějšího zápisu HTML kódu, který tento základní jazyk navíc rozšiřuje o novou funkcionalitu a následně je převeden do klasického HTML. To se hodí zejména u rozsáhlejších projektů. Co ale s rozsáhlými CSS styly? I na ně lze aplikovat podobný přístup. Aby kód zůstal srozumitelný a byla zachována struktura, případně aby mohlo dojít k obohacení o novou funkcionalitu, byly navrženy *CSS preprocesory*, které překládají vlastní syntaxi zdrojového kódu do klasického CSS, se kterým si už internetové prohlížeče dokáží poradit. Mezi nejznámější příklady se dají zařadit *LESS* a *Stylus*, které jsou psány v *Node.js,* nebo velmi populární ***SASS***, který je oproti dvěma předchozím napsán v programovacím jazyce *Ruby*. Je tedy patrné, že k jednoduchému zapisování CSS stylů je nutné přidat dílčí mezikrok, kterým je *transpilace*[3], což není vhodné pro editaci stylů přímo na serveru. Ve vývojovém prostředí je řešením sledovací režim, jenž monitoruje změny v souborech a při změně dojde k překladu a znovusestavení výsledného CSS souboru.

SASS využívá více syntaxí – klasickou *SASS* nebo *SCSS*. Od sebe se liší tím, že *SASS* nepoužívá uzávorkování do složených závorek a ani středníky, ale místo toho využívá odsazení pomocí tabulátoru. Druhou možností je *SCSS*, které má stejnou syntaxi jako *LESS,* ale využívá uzávorkování a středníky. Jaký způsob zápisu si uživatel zvolí, je už jen na něm. A teď hlavní otázka, co to tedy vývojáři přináší za výhody kromě dělby kódu a přehlednosti? Jedná se hlavně o (Documentation of Sass, 2016):

- Rozšíření jazyka o proměnné, *mixiny* (znovupoužívání opakujícího se kódu), *Nesting*(přeloženo jako hnízdění, což je zanořování CSS stylů do sebe), různé funkce či početní operace a podmínky

[3] Transpilace – proces překladu zdrojového kódu z jednoho programovacího jazyka do jiného. Na rozdíl od kompilace dochází k překladu jazyků na přibližně podobné úrovni abstrakce

- Mnoho užitečných funkcí pro manipulaci například s barvami a jinými hodnotami
- Pokročilé funkce jako například směrnice kontroly pro knihovny
- Dobře formátovaný, automaticky kontrolovaný výstup proti chybám
- Dělbu souboru na menší, znovupoužitelné části, které lze importovat

Z některých výše uvedených poznatků vyplývá, že existuje více nadstavbových technologií, které vyžadují více úkonů, než je nezbytně nutné při psaní prostého kódu. Avšak využitím kombinací těchto technologií a některých z nástrojů, které je potřeba do projektu zapojit kvůli některým z těchto technologií, se mohou lišit dobří vývojáři od těch obyčejných. Mezi další pokročilé nástroje lze zařadit ***Bower***, ***Grunt*** a ***Gulp***, což jsou moduly pro už mnohokrát zmíněný *Node.js* a jejich využití se může hodit právě pro zmíněné dodatečné úkony, spojené s využitím úspornějšího zápisu kódu. Pro upřesnění, nástroj *Bower* s těmito úkony přímo nesouvisí, ale má za úkol starat se o klientskou část aplikace a někdy je označován za „*balíčkovací systém*“ (Staněk, 2014). Tento nástroj dodává do projektu potřebné knihovny jako může být třeba jQuery nebo potřebné soubory pro frontendový framework *Bootstrap*.

Nyní konečně ke zmíněnému preprocesorovému převodu SASS do CSS, kterou může mít na starost jeden z dvojice *Gulp* nebo *Grunt*. Oba dva slouží jako takzvaný „*The JavaScript Task Runner*“, což znamená, že tyto nástroje po spuštění automaticky sledují, zachytávají a reagují na změny v určitých souborech (transpilace). Oba dva jsou si velmi podobní, ovládají se pomocí příkazové řádky, jsou *open-source* pod licencí MIT a mají širokou a aktivní komunitu uživatelů. Rozdíl je však až ten, že *Gulp* se snaží být rychlejší tím, že data se mezi jednotlivými procesy předávají přes *pipeline*[4], čímž se redukuje počet diskových operací a úlohy se dají snadno řetězit, avšak je na vývojáři, který zvolí (Ožana, 2014). Potřebné moduly pro práci s těmito nástroji jsou do projektu dodány pomocí jiného, „*balíčkovacího systému*“ ***NPM***, ale tentokrát se jedná o „*serverový balíčkovací systém*“, který do projektu automaticky přidává například moduly *gulp-pug, gulp-sass, gulp-images* atd., ale také může sloužit jako náhrada za předchozí zmíněný Bower, který oproti nově zmíněnému *NPM* ve své podstatě nic odlišného neposkytuje, takže *NPM* lze využít i pro přidání knihoven jako jsou *Bootstrap*, *jQuery* atd.

V případě využití frameworku budou k dispozici již hotové knihovny a nástroje, které se budou do projektu moci přidávat, a tím může být práce ulehčena. Přeci jen je to výhodnější, než na všechny problémy znovu vyvíjet řešení samostatně. O správu závislostí se v PHP stará nástroj zvaný ***Composer***, který podle oficiální stránky *getcomposer.org* umožňuje deklarovat tyto knihovny a má na starost jejich závislosti. Navíc se stará o aktualizace těchto knihoven a již při počáteční instalaci vybírá nejaktuálnější verzi. Nejedná se však o klasického „*správce balíčků*“, jenž by měl za úkol globální instalaci, nýbrž se stará o přidávání závislostí do

[4] Pipeline – sada instrukcí, která se uplatňuje při toku dat, kdy výstup jednoho procesu je zároveň vstup dalšího, což umožní vykonávání větší počtu instrukcí a umožňuje paralelní zpracování.

konkrétního projektu. Na první pohled je i patrné, že základní myšlenka tohoto nástroje není žádná novinka, ale je silně inspirována zmíněným *NPM*.

V posledních letech se velmi rychle do povědomí všech programátorů, bez ohledu na to, v jakém jazyce své projekty vyvíjejí, dostal pojem „***verzovací nástroje***“, nebo také jen zkráceně **GIT**. Jedná se o nástroj, který umožňuje rychlý přístup ke kompletní historii projektu, možnost vyvíjet bez dostupnosti připojení k centrálnímu repozitáři nebo schopnost vytvářet lokální větve vývoje. V neposlední řadě je každá kopie repozitáře jeho úplnou zálohou. Jeho hlavní výhodou je možnost procházet si historii vývoje, mít přehled, kdo je za jakou část vývoje zodpovědný a hlavně umožňuje snadný vývoj projektu v týmech. I když dojde k situaci, kdy jednotlivý členové týmu upraví stejné soubory, nedojde k jejich vzájemnému přepsání, nýbrž vývojáře upozorní na tyto možné problémy a přiměje je, aby svou práci sladili. Jednotlivé větve umožňují, že vývoj probíhá odděleně a ke sloučení těchto větví dojde až v případě, kdy se jedná o odladěnou část, které je připravena ke spojení s hlavní produkční verzí projektu. Tento nástroj se hodí jak pro vývoj velkých, tak i malých projektů (Vrána, 2012).

S první oficiální verzí tohoto nástroje přišel jeden z hlavních vývojářů *Linux kernelu* Linus Torvalds v roce 2005. Celý princip je založen na způsobu ukládání souborů, kdy každý soubor je uložen vždy jen jednou a poté jsou ukládány jen tzv. *snapshot* a v každém pomyslném uzlu (*commit*) jsou uloženy všechny soubory právě jako tyto *snapshoty*. Jedná se o binární soubory, které umožňují ukládat nejenom textové soubory, ale také i obrázkové, případně jiné soubory a velikost adresáře zůstává skoro totožná. Neukládá se rovnou na server, ale každá operace je nejprve lokální, takže se mezi lokální uložiště a server vloží dílčí mezičlánek, který umožňuje možnost opravy před publikováním na zmíněný server. Pro kontrolu integrity jsou využity kontrolní součty známe jako *hash* (Gajda, 2013).

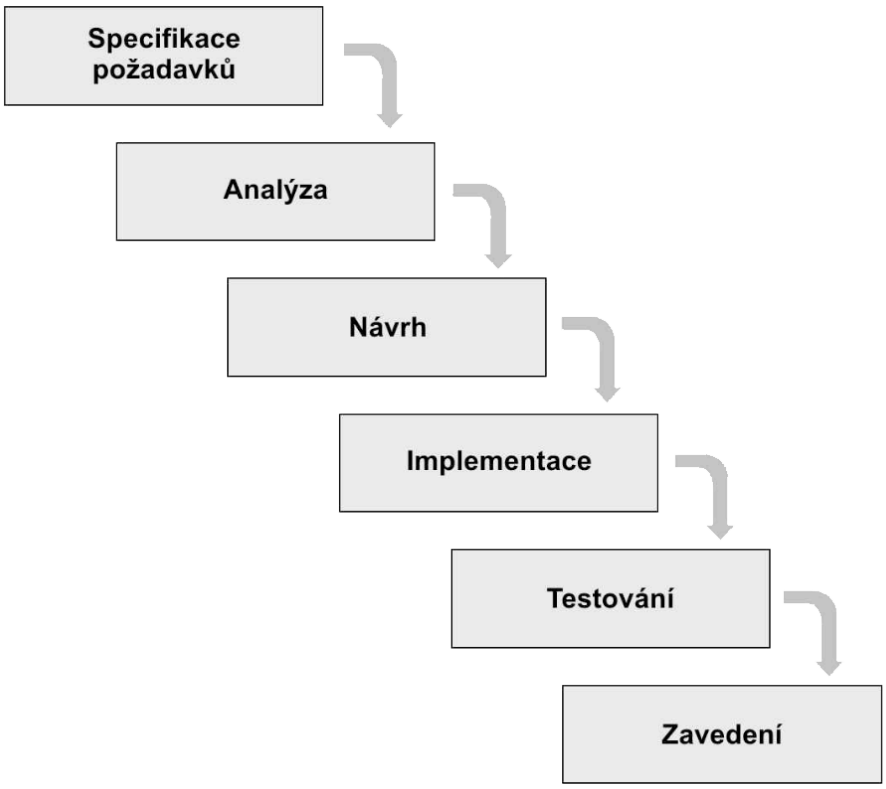
3.5 Metodiky a přístupy při vývoji systému

Již jsou známé základní a i pokročilejší nástroje pro vývoj redakčních systémů ve webovém prostředí, ale ještě je však potřeba před zahájením celého vývojového procesu si objasnit jisté fakty, jakými jsou například informace, jak při tomto vývoji postupovat, co je potřeba zjistit za informace ještě před daným vývojem a jaké existují metody vývoje systémů jako takových.

Celková úspěšnost tvorby systémů podle většiny průzkumů není uspokojivá. Podle společnosti *Standish Group* v rámci projektu *CHAOS* je úspěšnost projektu definována splněním tří kritérií – **projekt dokončen včas**, **podle rozpočtu** a **se všemi specifickými funkcemi**. S postupem času se však tato úspěšnost tvorby projektů postupně zlepšuje, a to i vlivem například aplikací různých metodik, standardů a norem. Zvýšit úspěšnost softwarových projektů se mimo jiné také snaží jak **tradiční přístupy**, tak i ty **agilní**. Každý z nich však k tomu přistupuje na základě jiných předpokladů. Tradiční přístupy považují tvorbu IS za definovaný proces, který je možné přesně popsat a podle toho popisu opakovaně realizovat a zlepšovat. Agilní přístupy naopak vycházejí z přesvědčení, že tvorba IS je empi-

rický proces, jenž nemá smysl popisovat, ale je třeba jej monitorovat a přizpůsobit realitě. V rámci tradičních přístupů se můžeme setkat s referenčními modely procesů, různými modely životního cyklu, posuzováním zralosti a způsobilosti procesů a tradičními metodikami budování IS. Agilní přístupy jsou reprezentovány agilními metodikami, jež jsou postaveny pouze na iterativním modelu životního cyklu s velmi krátkými iteracemi. V poslední době se více než kompletní agilními metodiky používají spíše jen jednotlivé agilní metody či praktiky, které se navzájem kombinují (Bruckner, 2012).

Mezi historicky neznámější modely životního cyklu patří vodopádový model a model pro iterativní vývoj. Životní cyklus systému je časový úsek, který začíná úmyslem vytvořit systém a končí, když se systém přestane používat. Model životního cyklu je rámec procesů a aktivit a je často organizován do fází. **Vodopádový model**, který se rozšířil zejména v 70. a 80. letech minulého století, se inspiroval postupy v průmyslu a rozdělil vývoj softwaru do postupně prováděných fází. Pojem vodopád se začal používat proto, že jednotlivé fáze následují po sobě a jejich grafické znázornění připomíná vodopád (Bruckner, 2012). Celý proces je zachycen na následujícím obrázku.



Obr. 2 Jednotlivé dílčí fáze životního cyklu Vodopádového modelu
Zdroj: Tvorba informačních systémů, 2012

Začíná fází specifikace požadavků, poté následuje fáze analýzy, návrhu, implementace, testování a nakonec zavedení. V době vzniku tohoto modelu představoval vý-

znamný pokrok, neboť dokázal rozdělit celý proces vývoje na dílčí fáze, které se mohou opakovat. V dnešní době je již mnohdy kritizovaný, ale v některých případech může být dostačující. To závisí hlavně na tom, zda na začátku procesu vývoje mohou být specifikovány všechny požadavky a zda se v průběhu vývoje nemění. V opačném případě nastává problém. Další podstatný vliv je, zda je potřeba zapojení zákazníka do projektu. Vodopádový model počítá se zapojením jen na začátku a na konci. Nejpodstatnější nedostatek je včasná integrace, která se provádí až na konci vývoje, kdy se mohou zjistit značné nedostatky vyžadující změny v návrhu a přeprogramování, což vede ke zpoždění celého projektu (Bruckner, 2012).

Tyto nevýhody řeší **iterativní vývoj**. Ten je postaven na skutečnosti, že člověk lépe řeší menší problémy, a proto je potřeba rozložit celý projekt do dílčích projektů – iterací. Přitom jednotlivé iterace obsahují všechny dílčí fáze od plánování až po zavedení. Výsledkem každé iterace je funkční, otestovaná část systému. To vede k podstatnému snížení rizik při vývoji, protože případné problémy se odhalí včas, a tudíž se také včas dají řešit. Iterativní vývoj rovněž poskytuje lepší přehled o projektu jako takovém (Bruckner, 2012).

Zastánci **agilních vývojových přístupů** jsou přesvědčeni, že proces vývoje nelze předem pospat, ale je nutné jej průběžně monitorovat a přizpůsobovat změnám, a proto každá z agilních metodik je svým způsobem jedinečná. I když počátky těchto technik sahají do 80. let minulého století, důležitým rokem byl rok 2011, kdy byl definován *Manifestu agilního vývoje softwaru,* ve kterém byly definovány základní principy vývoje, založeného na těchto technikách. Mezi neznámější metodiky patří ***Lean Development***, ***Extrémní programování*** nebo ***SCRUM***, která je zaměřena hlavně na řízení projektu a jejíž vývoj probíhá v krátkých iteracích nazývaný *Sprint* trvajících v rozmezí 2 až 4 týdnů. Členové týmů si vybírají úkoly, účastní se denních schůzek, monitorují a identifikují vzniklé problémy a snaží se vše v rámci *Sprintu* vyřešit (Bruckner, 2012).

4 Metodika

Při vývoji systému by měly být dodrženy jisté postupy, kterými se bude řídit i tato práce a bude se dle nich postupovat. Tyto přístupy tvorby systémů vychází z modelů životního cyklu, případně z jiných přístupů, o kterých bylo pojednáváno v předchozí podkapitole. Tato práce bude rozčleněna na následující části:

- Specifikace požadavků
- Analýza požadavků
- Návrh systému
- Implementace
- Testování
- Zavedení rozhraní
- Vyhodnocení

V úvodní fázi budou specifikovány dílčí požadavky společnosti, pro kterou je systém vyvíjen. Tyto požadavky jsou založeny na zadání společnosti, kde jsou definovány opakující se požadavky klientů na administrační systém. Na základě těchto požadavků bude poté provedena analýza, ve které se zvolí optimální technologie, zanalyzují se ostatní firemní řešení a zjistí se, jestli z daných řešení lze využití některé části pro vývoj nového systému. Na základě tohoto zjištění budou definovány funkční a nefunkční požadavky a poté se provede analýza rizik, ve které budou nalezeny možná rizika projektu, pro které se následně stanoví příslušná opatření. Součásti analýzy budou také vlastní postřehy a poznámky ke specifikaci požadavků společnosti, které budou reagovat na případné nedostatky.

Na základně specifikace požadavků a následné analýzy se vyhotoví návrh systému, který bude obsahovat dílčí návrhy funkcionality systému. Pro jednotlivé návrhy se využije jazyk UML, procesní modelování, ER model a drátěný model. Systém bude během návrhu rozložen na dílčí moduly se zaměřením na určitý druh funckionality. Po dokončení návrhu následuje implementace, jejíž výsledkem bude funkční systém, využitelný jako již hotový produkt. Implementace bude probíhat postupně po částech, kdy výstupem každé části bude funkční část systému, poté proběhne krátké zhodnocení dosavadní části s případnými návrhy na zlepšení, které budou implementovány v následující části. Vzhledem na plánované budoucí rozšiřování, nebo případné drobné úpravy dle požadavků jedlových zákazníků budou jako součást implementace také sepsány testy. Tyto testy zamezí následným chybám plynoucím právě ze zmíněných změn v systému.

Funkční řešení se následně otestuje na firemních zakázkách, kde se zjistí, jestli navržené řešení obstojí v reálných podmínkách a na praktických problémech. Po zavedení rozhraní do ostrého prostředí se provede zhodnocení, zda řešení vyhovuje, či nikoliv. Součástí celkového vyhodnocení je také ekonomické zhodnocení efektivnosti, ekonomické úspory společnosti a technické shrnutí. Společnosti budou navrženy možnosti pro zlepšení, nebo prostory pro případné rozšíření.

5 Specifikace požadavků společnosti IDEATECH s.r.o.

Již bylo uvedeno, že nově navrhované řešení bude vyvíjeno ve spolupráci se společností **IDEATECH s.r.o.** Jde o obchodní společnost se sídlem v hlavním městě Praze. Do rejstříku obchodních firem byla zapsána pod identifikačním číslem 05090521 dne 17.5.2016. Z toho lze usoudit, že se jedná o relativně mladou společnost, avšak její zakladatel a jeho společníci už předtím působili několik let v daném oboru, ale pod jinou právní formou podnikání. Výhradním vlastníkem společnosti je Adam Kysel, jenž společnost založil se základním kapitálovým vkladem 100 000 Kč, a je tedy jediným vlastníkem. Společnost se po celou dobu své existence zabývá těmito předměty činností:

- Zprostředkování obchodu a služeb
- Poskytování softwaru v oblasti informačních technologií, zpracování dat, hostingové a související činnosti s webovými portály
- Poradenská a konzultační činnost, zpracování odborných studií a posudků
- Reklamní činnost, marketing, mediální zastoupení
- Grafické služby

V současné době se společnost primárně zabývá tvorbou webových aplikací a internetových obchodů, z čehož má firma největší příjem, ale tento fakt by její řídící pracovníci chtěli změnit a mít více prostředků na tvorbu mobilních aplikací a dále více rozvíjet i tuto oblast podnikání. Pro ukázku tvorby společnost vytvořila spoustu webových aplikací s pokročilými funkcemi rezervací, s napojením na třetí strany (např. účetní systém Pohoda), různé platební brány (např. *GoPay*), propojení na *MLM systém D3Soft*, propojení s lékařskou kartou a spoustu dalších systémů. Momentálně společnost vyvíjí i specifickou webovou aplikaci zaměřenou na reporty a analýzy databázového využití klientů pro společnost, zabývající se právě optimalizací databází. V nejbližší době zahájí projekt na tvorbu specifického CRM systém. Jako vedlejší činnosti společnost vyvíjí doplňky pro systém *WordPress* a aktuálně jsou v České republice jediní, kdo vlastní propojení *WordPress WooCommerce* se službou *Balíkobot*, ze které pramení pasivní příjem společnosti. Jako cíl do následujícího roku 2018 si zvolili rozšíření služby do blízkého zahraničí, a to zejména do Rakouska a Slovenska, jelikož místo vykonání činnosti je momentálně město Břeclav, jež leží na hranici České republiky s těmito zeměmi.

5.1 Neformální specifikace požadavků

Ve 2 kapitole bylo definováno, že hlavním cílem je vytvoření sjednoceného komplexního řešení pro opakující se požadavky zákazníků na správu webové aplikace, vycházející z definovaných požadavků společnosti. Takto vytvořený systém by měl sloužit jako již hotový produkt, který bude zákazníků nabízen, nebo by měl alespoň

poskytnou dostatečně stabilní základ pro detailnější zadání a rozšiřování dle potřeb zákazníka. Společnost disponuje více hotovými projekty sloužícími jako reference pro lepší představy zákazníků, přičemž tato řešení byla vyvíjena předchozím zákazníkům přímo na míru. Velmi často se však stává, že nový zákazník má stejné potřeby, a proto by mu tato řešení také vyhovovala. Bylo by tedy vhodné tato řešení sjednotit, případně navrhnout nová, pokud by již nevyhovovala nebo byla zastaralá. Nově poskytnuté řešení by mělo být komplexní a poskytovat široké spektrum možností. Z tohoto důvodu by se mělo k nově navrhovanému řešení přistupovat zodpovědně již od samotného návrhu, a pokusit se systém správně rozdělit do dílčích částí, neboli modulů, které by měly jít jednoduše odebrat nebo naopak také přidat. Z toho vyplývá, že tyto části musí být na sobě nezávislé.

Jelikož ve společnosti IDEATECH s.r.o. nejsou jako zaměstnanci jen programátoři, kteří disponují potřebnými znalostmi kódu, ale i jiní zaměstnanci specializující se na jiné oblasti, bylo by vhodné pro nově navržený systém vytvořit přívětivé uživatelské prostředí, skrze které by se dalo se systémem snadno pracovat a v případě potřeby v systému provést modifikace bez nutnosti zásahu do kódu. Z těchto zmíněných důvodů je jednou z hlavních myšlenek navrhnout **mechanismus pro určování aktivních modulů,** ke kterému budou mít přistup výhradně zaměstnanci firmy a budou schopni zákazníkovi nakonfigurovat systém dle potřeb. Zákazník nebude mít k této části systému přístup, a tudíž nebude schopen si sám aktivovat nebo deaktivovat jednotlivé části. Pokud by vyjádřil potřebu rozšířit systém o již hotovou komponentu, mělo by stačit ji jen zaktivovat v systému, případně přizpůsobit frontendovou část a tím vše vyřešit.

Jak napovídá předchozí odstavec, ne všichni v administračním rozhraní budou mít stejné oprávnění a budou zde moci vykovávat všechny akce. Proto by v systému měl být navržen stabilní **mechanismus pro definování uživatelských práv,** založený na uživatelských skupinách a jim definovaných přístupů k akcím, které budou moci vykonávat. Jen zaměstnanci společnosti IDEATECH budou moci aktivovat a deaktivovat části administračního systému a budou nad ním mít absolutní kontrolu. Může se ale také vyskytnou situace, kdy zákazník bude vyžadovat vymezení pravomocí pro své zaměstnance. Jedná se například o situaci, kdy nebude chtít, aby někteří jeho zaměstnanci, kteří budou mít k systému přístup, si mohli zobrazit informace o objednávkách nebo o uživatelským údajích, avšak bude vyžadovat, aby se mu starali například čistě jen o přidávání příspěvků na web.

Jednou z hlavních a nedílných funkcionalit systému by měla být **správa obsahu webu**. Webovou stránku, neboli frontendovou část, budou tvořit jednotlivé stránky, jež budou zařazeny do menu. Díky němu se uživatel bude moci pohybovat na webu, a proto by v systému měl být mechanismus pro vytváření, potažmo celkovou správu tohoto menu a definování, jaké položky v něm budou zařazeny, v jaké pozici se budou nacházet – zanořování položek do sebe a definování vztahů mezi nimi. Další funkcionalitou pro správu obsahu webu, která je v systému očekávaná, je možnost práce se **SEO** položkami, jakými jsou například titulky a popisky stránek či klíčová slova webu, na která se zákazníci velmi často ptají. Ke konkrétnímu přidávání obsahu webu velmi často majitelé používají blog, nebo

spíše **aktuality**, čímž informují zákazníky o připravovaných akcích či změnách nebo poskytují informativní sdělení. Tato část se dá vesměs pojmenovat různými názvy, avšak podstatné je to, že slouží k přímému vkládání informací do stránky.

Velká část zákazníků, kteří společnost osloví, tvoří skupinu využívající internet jako prostředek pro prodej. Tento prodej může být primární činností, tvořící jediný zdroj příjmu prostřednictvím takzvaného **e-shopu**, nebo činností podpůrnou, kdy internetový prodej představuje jen část hlavního příjmu a zákazník disponuje například i „*kamennou prodejnou*“. Cílem této práce není poskytnou do detailu propracovanou platformu pro prodej, ale jak název práce napovídá, bylo by velmi vhodné zahrnout do nově vytvořeného komplexního řešení i pevný základ pro případné budoucího rozšiřování. Tento základ by bylo možné v některých případech, kdy nejsou kladeny velmi specifické požadavky, použít, případně toto řešení rozšířit prostřednictvím modulů a nových funkcionalit. Problematika internetového prodeje je poměrně komplikovanou záležitostí a vyvinout řešení pro velké společnosti by samo o sobě mohlo zabrat jednomu člověku velmi dlouhou dobu, avšak základ bývá de facto stejný. Požadované řešení této části administrace je tedy určené spíše pro menší společnosti, případně pro společnosti nacházející se na pomezí malých a středně velkých podniků. Detailní požadavky budou rozebrány v následující podkapitole 5.2.2.

Velmi důležitou částí systému, které by se měla věnovat jistá pozornost, tvoří **správa medií**. Jak již bylo zmíněno v úvodu 3. kapitoly, obsah webových stránek je tvořen kromě textu také i multimediálními daty, do nichž lze jednoznačně zařadit obrázky, videa a zvuk. Tato data jsou lidmi mnohem lépe vnímána, je dokázáno, že jim lidé věnují více pozornosti a lépe z nich získávají informace. Nově navržený systém by měl uživateli vyjít co nejvíce vstříc a poskytnout jednoduchou správu těchto důležitých dat. To znamená, že uživatel bude mít jasný přehled, jaká data se na jeho webu nacházejí, bude existovat jednoduchá možnost je přidávat, odstraňovat či editovat. K tomuto typu dat se vztahují také jisté standardy, podle kterých je nutné definovat k tomuto typu dat specifické atributy – titulek a zástupný text. Jedná se součást SEO optimalizace, a i když se může zdát, že nemusí mít velký vliv, přesto by bylo hodné tyto konvence dodržet a umožnit uživateli jednoduše pracovat s těmito specifickými daty.

Velmi účinným nástrojem, který lze využít pro práci s návštěvníkem webové stránky, je **sběr uživatelských e-mailových adres**. Pro jejich získání je nutný souhlas uživatele, jenž však může na webové stránce sám poskytnout při jeho vyplnění. Takto získanou adresu lze poté využít mnoha způsoby, jako například pro zasílání informativních e-mailových zpráv, které lze využít jako podporu prodeje, nebo pro možnosti remarketingu, tedy cílení reklamy na zákazníky. Proto by v systému měl být zahrnut mechanismus, který by umožňoval sběr e-mailových adres od uživatelů, uchovávat je a umožňovat základní operace jako například export pro jiné využití.

5.2 Formální specifikace požadavků

Specifické zadání společnosti patří hlavním a podle předpokladu nejvíce používaným položkám administrace - správě obsahu a části pro internetový prodej. K ostatním věcem jsou poznamenány jen minimalistické požadavky, hlavně na existenci určité funkcionality, a proto je zde prostor pro návrh vlastního řešení, případně pro zlepšení. Pro UX návrh administračního rozhraní je kladen požadavek na jednoduchost prostředí, se kterým by se uživatelům jednoduše pracovalo. Velmi běžně totiž v oblasti vývoje systémů dochází k situaci, že se zákazníci obracejí na poskytovatele služeb, protože nevědí, jak s některými částmi systému pracovat, některé položky nemohou najít a celkově se jim systém zdá složitý. Proto i sebelepší záměr nemusí být vždy správně pochopen a nesmí se zapomínat na hlavní myšlenku, pro koho je navrhované řešení vyvíjeno – pro zákazníky. A právě podle dosavadních zkušeností se velmi osvědčilo tabulkové UX prostředí, kterému uživatelé dobře rozumí.

Předpokládaná hlavní nabídka menu administrace v zadání se odráží od jednotlivých částí a funkcionalit systému, jež jsou specifikovány právě v zadání. Tato hlavní nabídka je zachycena na následujícím seznamu:

1. **Obsah stránek**
2. **Aktuality**
3. **E-shop**
4. **Partnerské loga a sociální sítě**
5. **Uživatelé a práva**
6. **Nastavení a SEO**
7. **Sběr uživatelských e-mailových adres**
8. **Video návody k administraci**

Základ administrace je tedy tvořen **správou obsahu** webové prezentace. Ta se skládá primárně z dílčích stránek, a samotné zařazování stránek do menu je toho součástí. Jednotlivé stránky musejí jít jednoduše vytvořit, editovat a také odstranit. U jednotlivých stránek musí existovat funkcionalita pro zařazení stránky, nebo naopak o její vyřazení z menu, případně možnost stránky zařadit do sekundárního menu, takzvaného *submenu*.

5.2.1 Formální specifikace správy obsahu a nastavení

Obsah stránky by měl obsahovat základní nastavení SEO prvků, jakými jsou titulek a popisek, jež by měly být optimalizovány pro internetové vyhledávače. Optimalizace v tomto případě znamená kontrolu, případně omezení počtu znaků. Pokud uživatel hledá specifickou stránku prostřednictvím internetového vyhledavače, zadá do vyhledavače klíčové slovo a ten mu zobrazí položky, které splňují podmínky vyhledávání. Tyto položky budou obsahovat právě zmíněný titulek a krátký popisek. Je tedy vhodné zvolit dostatečnou délku. Titulek se buď automaticky doplní

z názvu položky, nebo se definuje ručně. Kromě zmíněného titulku, popisku a názvu by stránka měla obsahovat také možnost nastavení URL adresy, pod kterou bude dostupná, její obrázek a *perex* (krátký, sekundární popisek obsahu). Textových popisů může být samozřejmě více než jen jeden. Stránka by také měla obsahovat možnost volby pořadí dané stránky v menu, jež se může nastavit pomocí čísla – čím vyšší číslo, tím bude položka zařazena výš v seznamu. Ke konkrétní stránce musí jít kromě popisků a dílčích nastavení vložit také obsah.

Pro vkládání konkrétního obsahu do webových stránek se využije některý z jednoduchých editorů. Ty umožňují vkládat kromě čistého textu také HTML elementy a tím formátovat text. Tyto editory fungují na principu rozšíření klasického textového prvku HTML formuláře o funkcionality pro vkládání právě HTML elementů. Výsledný text se poté uloží do databáze a jeho výstup se nebude ošetřovat proti výpisu HTML elementů. Některé webové stránky jsou příliš individuální, jednotlivé podstránky obsahují mnoho textových bloků a i tyto bloky musejí jít nějakým způsobem editovat skrze administraci, avšak vložit větší množství těchto editovatelných bloků do detailu stránky není správné řešení. Proto je potřeba navrhnout mechanismus editování textových bloků přímo z frontendové částí po provedení autentizace a autorizace. To se nebude týkat pevných bloků, které se dají editovat přímo z administrace a mají jasně určené své místo – aktuality a jiné podobné bloky.

Každá stránka může obsahovat svou **galerii fotek**, jež může být realizována skrze prezentaci (takzvaný *slider*), nebo maticově, avšak toto je spíše otázka návrhu webové prezentace. Nahrávání fotek by mělo jít realizovat hromadným výběrem a být omezeno velikostí jednotlivých fotek na 5MB, s možností upozornění, pokud je fotka větší než 800KB, aby uživatel nahrával fotky v kvalitě pro webové stránky. Název fotky a zástupný popisek se automaticky přiřadí, avšak bude existovat možnost jej v administraci manuálně změnit. Fotky by mělo jít hromadně nahrát, ale případně také i hromadně odstranit. Každá stránka bude kromě svojí galerie také obsahovat i možnost nahrát obrázek na pozadí, případně jeden z obrázků nastavit jako hlavní.

Již zmíněné **aktuality** by se měly skládat z názvu, titulku, URL adresy, která by se měla automaticky doplnit nebo jít i ručně zadat, obrázku (případně i z více obrázků) a konkrétního obsahu. Titulek může obsahovat základní HTML elementy, jakými jsou vložení prázdného řádku či tučného textu. Některé aktuality mohou mít vyšší prioritu sdělení než jiné, což znamená, že by měly být v pořadí nad ostatními, aby si jich uživatel snáze všimnul. Proto by měla existovat možnost nastavit tuto prioritu, jež bude před chronologickým řazením dle data vložení, a pokud datum vložení nebude aktuální, nýbrž dopředný, aktualita bude na webové stránce zobrazena až nastane toto datum.

Součástí téměř všech webových prezentací bývají například v patičce, nebo kdekoliv jinde na stránce, **partnerská loga** společnosti zobrazeny formou loga konkrétního partnera, jeho jména a odkazu na jeho webovou stránku. Jednotlivá loga by měly jít seřadit například pomocí číselného řazení a také určit, na jaké stránce se budou zobrazovat. Pro toto určení lze využít mechanismus kategorií,

přičemž jednotlivé kategorie jsou existující stránky. Pokud by pro logo nebyla vybrána žádná kategorie, logo partnera se zobrazí na všech stránkách. Na stejném principu by měly fungovat i další informace, které jsou pro web důležité. Jedná se zejména o nastavení kontaktního telefonu, e-mailových adres, odkazů na sociální sítě, ale také i o popisek a klíčová slova celé webové stránky. Internetové vyhledávače je mohou ignorovat, ale někteří zákazníci nikoliv, takže minimálně jen pro efekt by tu tato možnost měla být. Případně ponechat v této části možnost pro rozšíření o další specifické údaje podle budoucích potřeb, jež by bylo potřeba nastavovat z administrace.

Pro případ, že administrační prostředí bude moci využívat více uživatelských skupin, měl by být navržen mechanismus pro **definování oprávnění** konkrétním skupinám uživatelů, čímž se získá alespoň částečná kontrola nad uživatelským chováním. O této možnosti bylo pojednáváno už v předchozí kapitole o neformální specifikaci požadavků, avšak zde budou rozebírány konkrétní specifika společnosti. Samotná sekce pro definování uživatelských práv by měla být přístupná jen pověřeným uživatelům, a pokud by byl udělen přístup i zákazníkovi k nastavování oprávnění, musí být vyřešena otázka, jak zamezit přidělení přístupu k internímu nastavení administrace. Samotný mechanismus by měl být co nejvíce jednoduchý - uživatelé se budou moci shlukovat do skupin, přičemž těmto skupinám se bude moci povolovat přístup do dílčích částí systému. Pokud uživatel bude mít přístup do specifické části, nebude to automaticky znamenat, že v této sekci může vykonávat jednotlivé akce pro editaci, vytváření a mazání položek. Bude platit, že cokoliv není povoleno, automaticky se bere jako zakázané. V případě potřeby se bude moci jakémukoliv uživateli vygenerovat nové heslo, jež bude náhodné a bude mu odesláno na email, skrze který se bude do systému přihlašovat. Pro ještě detailnější přehled nad využíváním systému by bylo vhodné **zaznamenávat všechny uživatelské akce** a mít přehled nad tím, kdy a kým byly provedeny. Pro tento požadavek by měla být zavedena dílčí sekce, kde bude výpis těchto již provedených akcí. Díky tomu bude velmi snadné v případě potřeby jednoznačně určit, kdo je za akci zodpovědný a zjistit, proč se tak stalo.

Pro pokus o vyhnutí se často opakujícím dotazům by v administraci měla existovat malá sekce, která by obsahovala připravené videonávody, jak s administrací pracovat a upozorňovat na její klíčové informace. Je samozřejmostí, že novému klientovi bude vše vysvětleno, ale pro případ, že by něco zapomněl nebo by chtěl práci delegovat na více lidí, budou zde zmíněné návody připravené zaměstnanci společnosti. Skrze návody se společnost pokusí odpovědět na všechny klientovi případné otázky. Tato část by neměla být součástí hlavního menu, ale odkaz na ni by měl být umístěn na méně nápadné místo v patičce, nebo hlavičce systému.

5.2.2 Formální specifikace modulu pro internetový prodej

Jednoduchý základ platformy určené pro internetový prodej musí obsahovat **správu produktů**, přičemž tyto produkty mohou být jednoduché, nebo naopak variabilní. Jinými slovy, k produktu existuje možnost definovat libovolné atributy. Produkt zůstává vždy stejný, ale může se vyskytovat v různých provedeních jako na-

příklad velikosti oblečení. Další parametry produktu jsou název, cena, nastavení URL adresy pro zobrazení detailu produktu ve webové stránce, nastavení slevy a akční ceny nebo to, zda produkt lze vůbec objednat, případně status (publikovaný, skrytý atd.). Dále každý produkt může mít svou **galerii** a zařazení do **kategorií**, podle kterých se produkty budou moci na webové stránce filtrovat. Tyto kategorie se budou moct snadno vytvářet a editovat a na webu pro ně bude připravené stromové menu. Produkty se budou moci přidat do **košíku** a následně z něj i odebrat. Nákupní košík bude propojen s uživatelem a v případě nedokončení nákupu košík nezaniká, nýbrž zůstává stále naplněn produkty konkrétního uživatele. Pro dokončení nákupu bude nutné postupně v dílčích krocích vyplnit uživatelské údaje, zvolit metody dopravy a platby, které budou editovatelné z administrace a potvrdit souhrn objednávky. K objednávce se bude moci připojit uživatelský slevový kupón, jenž bude také editovatelný z administrace. Posledním krokem objednávky bude děkovná stránka, na kterou se bude moci uživatel dostat pouze po dokončení objednávky. Tato děkovná stránka bude obsahovat případné skripty pro měření konverze.

Všechny dokončené **objednávky** budou k zobrazení v administraci, kde bude možnost nastavovat jim dílčí stav podle toho, jestli se objednávka zpracovává, případně je na cestě, nebo už vyřízená. Objednávka bude také editovatelná – uživatelé si budou moci změnit jejich zadané údaje, doplnit další produkt k objednávce či ručně nastavit dobu splatnosti faktury, nebo její datum vystavení. Faktura se bude generovat automaticky a příchozí platby přes internet se spárují ručně s konkrétní objednávkou skrze variabilní symbol. Ten se automaticky vygeneruje při dokončení objednávky. Při dokončení objednávky, nebo při jakékoliv změně stavu se zákazníkovi automaticky odesílá informační e-mail na jím zadanou adresu, případně při vzniku objednávky dojde k informování i dotyčné osoby, která je za objednávky zodpovědná. Zákazníkovi navíc s e-mailem přichází i faktura ve formě PDF dokumentu. Jako se uchovávají všechny e-mailové adresy uživatelů, získané z webové stránky, tak by také měl existovat i **přehled všech zákazníků** – registrovaných i neregistrovaných – a jejich seznam se všemi jejich informacemi a přehledy jejich objednávek.

Kromě již zmíněné kupónové **slevy**, limitované datem platnosti a počtem použití, budou existovat i slevy uživatelské. Tyto slevy předpokládají registraci zákazníka do systému a budou vycházet z celkové utracené částky. V administraci se nastaví zlomové částky, při jejichž překročení dochází k přepočtu cen na webu podle procentuální slevy, jež se promítne jak do cen produktu, tak i do celkové ceny objednávky. Tyto slevy se však neprojeví do cen za poštovné a platbu. Kromě těchto slev se musí počítat i s množstevními slevami, které budou vycházet z nakoupeného množství. Navíc se nesmí zapomenout na možnost nastavení **více měn**. Tato část nemusí být vždy využita, ale v případě potřeby by měla existovat možnost nadefinovat z administrace měny, ve kterých se bude obchodovat. Správce systému bude moci vytvořit měnu, nastaví jí zkratku a určit její kurz. Zákazník na webové části jednoduše přepne měnu, ceny se přepočítají podle nastaveného

kurzu a toto uživatelovo nastavení se projeví i v administraci a e-mailových zprávách. Objednávka tedy bude i v administraci zobrazena v dané měně.

Z těchto požadavků je patrné, že pro tuto část systému jsou vyžádána **specifická nastavení**, která se týkají kromě již zmíněných položek také hlavně účetního nastavení, tedy zda provozovatel bude, či nebude plátce DPH. V případě, že plátcem bude, měl by zde nastavit i výši daně a tato hodnota se dále projeví i do výpočtu cen produktů. Toto nastavení se může po určité době změnit, a proto je vhodné vždy nastavit platnost. I když při založení internetového obchodu nemusí být provozovatel plátce DPH, po uplynutí nějaké doby může zjistit, že překročil povolenou výši obratu a automaticky se plátcem DPH stává, tudíž musí tyto změny zahrnout i do budoucího účetnictví. Kvůli vystavování faktur bude potřeba nastavit i další údaje, jakými jsou sídlo společnosti, kontaktní údaje včetně identifikačního čísla a daňového identifikačního čísla společnosti, číslo účtu a kód banky, případně spisovou značku.

6 Analýza požadavků

Po prostudování výše zmíněných požadavků společnosti je potřeba upozornit na jisté vlastní poznatky, případně na specifické nedostatky ještě před začátkem návrhu systému, které mohou pramenit z přehnané snahy o jednoduchost a již na první pohled působí poměrně zastarale, proto je potřeba do pozdějšího návrhu zavést jistou míru inovace. Některé z těchto prvků by se mohly projevit již v raných fázích návrhu, avšak jsou patrné již v tuto chvíli. Například předpokládané rozvržení systému, které se odráží do vzhledu menu již na první pohled nemůže být takové, jaké bylo zamýšleno v zadání, ale bude více sjednocené. Je to důsledkem hlavně požadavku na jednoduchost systému a přímo v původním zadání je zmíněno, že čím více možností má uživatel na výběr, tím více jej to může zmást, což by mohlo nastat hned při prvním pohledu na složíte základní menu. Dalším důvodem je plánované rozvržení systémů do samostatných modulů, tím pádem se některé zamyšlené položky sjednotí do jedné nadřazené položky.

Práce s jednotlivými údaji, jako zmíněné řazení položek, je v dnešní době nezbytný standart. Zmíněnou metodu řazení pomocí číselné hodnoty lze naopak považovat za již zastaralou, protože s příchodem nových technologií (viz. HTML 5 atd.) se práce s daty stala více intuitivní záležitostí. Tyto nové technologie poskytují řadu nových funkcionalit, jež při důmyslném využití poskytnou v budoucnu uživateli jistý komfort. Například při využití techniky *Drag and Drop* uživateli stačí položky pomocí myši poskládat tak, jak uzná za vhodné, přičemž v pozadí se ukládají již zmíněné číselné hodnoty pro určení pořadí. V zadání byly opomenuty i jiné užitečné funkce pro práci s daty, které by neměly chybět. V případě mnoha vypsaných položek by při klasickém tabulkovém zobrazení mohlo docházet k problému při vyhledávání, z tohoto důvodu by proto měly existovat kromě metody jednoduchého řazení jednotlivých položek i další operace pro práci s daty, jako jsou například filtrování, vyhledávání nebo možnost zmíněné hromadné operace mazání.

Nejvíce důležitý poznatek, který by měl být brán do úvahy ještě před samotným začátkem navrhování systému, se týká návrhu modulu pro internetový obchod. Po dokončení objednávky má systém generovat faktury a poskytovat přehled jednotlivých objednávek, které zajisté budou obsahovat informace o zákazníkovi, provozovateli či o produktech v objednávce, ale hlavně o ceně. Co se ale stane, pokud se některý ze zmíněných údajů v administraci změní? Tyto změny se v žádném případě nesmí projevit do již existujících objednávek a faktur, protože by způsobily naprostý zmatek a celý modul by se stal nepoužitelným. Ze zmíněného důvodu tedy vyplývá, že při jednotlivých změnách v produktech, údajích provozovatele, v účetnictví, nebo údajích zákazníka nesmí dojít k tomu, aby se tyto změny projevily do historie. V případě nutnosti smazání některých dat je potřeba tato data v administraci uchovat z důvodu archivace a napojení na jiné položky, jako mohou být zmíněné objednávky. Z tohoto důvodu se namísto úplného smazání položky jen skryjí, v případe editace by se systém měl pokusit starou položku odstranit, pokud by byla napojena na jiné údaje, tak je potřeba ji v systému uchovat a tudíž ji jen skrýt a na zároveň vytvořit novou položku s požadovanými změnami. Závěr je tedy

takový, že je potřeba kvalitního návrhu databáze a poté samotné funkcionality, jež využije právě návrh databáze a kontroly referencí jednotlivých odkazů z tabulek. Tyto kontroly referencí v databázi budou mít na starost výjimky v kódu.

6.1 Volba technologie

Ze specifikace požadavků společnosti je patrné, že je potřeba navrhnout komplexní řešení, přičemž hlavní zamýšlená výhoda takto navrženého systému je jeho znovupoužitelnost a rozšiřitelnost. Z předchozí podkapitoly 3.2 o frameworcích jasně vyplývají výhody využití různých typů frameworků a navíc jejich využití přispěje pro požadovanou znuvupoužitelnost nebo rozšiřitelnost a celkově i správný základ aplikace. Zmíněné závěry vycházejí hlavně ze skutečnosti, že tyto softwarové struktury jsou navrženy tak, aby dokázaly oddělit prezentaci informací od logiky aplikace, kód se stává dobře strukturovaný a znovupoužitelný a tudíž se i lépe udržuje, což je přesně to, co společnost IDEATECH požaduje. Další výhody, které frameworky do projektu přinášejí, jsou využití jejich doplňků a bezpečnostních mechanismů, pro které stačí dodržení doporučených postupů. Navíc s ohledem na velikost společnosti je i téměř nevyhnutelné využít některý z již existujících frameworků, tudíž by se případné nevyužití mohlo považovat i za chybu.

Otázka výběru bývá velmi často ryze subjektivní, jelikož každý z existujících frameworků poskytuje své specifické metodologie a své výhody. Ze zmíněných existujících frameworků však bude pro tento projekt zvolen český **NETTE framework**, což je dáno hlavně tím, že tento framework je mezi vývojáři v České republice nejvíce rozšířen, tudíž by neměl nastat problém v případě zaměstnání nových vývojářů do společnosti. Přejít z jednoho frameworku na druhý by se totiž mohl pro programátory jevit jako problém, jelikož je to otázka programátorských návyků a konvencí, ve kterých by společnost chtěla novým i stávajícím členům vyjít vstříc. Využitý framework navíc svou strukturou a rozhraním definuje do značné míry i kód aplikace.

Mimo jiné budou do projektu zapojeny i zbylé popisované technologie, bez kterých by se podobný projekt zajisté neobešel. Jedná se o jazyk **JavaScript** s knihovnou **jQuery** a frontendový framework **Bootstrap**. Pro tvorbu klientských webových prezentací, nad kterými bude systém pracovat, se využije šablonovací systém **Jade** s technologií **SASS**. Pro výsledné zpracování souborů a spojení všech dílčích prvků dohromady se postará nástroj **Gulp**. Pro přidávání potřebných dílčích prvků bude využit nástroj **Bower** a pro správu závislostí v NETTE frameworku a přidávání potřebných knihoven do projektu bude využit nástroj **Composer**. Při volbě datové základny projektu se opět vychází z potřeb společnosti, pro kterou je systém vyvíjen. Budou se využívat firemní servery, které využívají **MySQL databáze**. Pro účely zamýšleného typu projektu je to vhodný typ databáze, jelikož se nepředpokládají specifické požadavky na data.

6.2 Funkční a nefunkční požadavky

Jedná se převážně o nový projekt, který neobsahuje žádné hotové části převzaté z jiných firemních řešení, jelikož po provedení rychlého průzkumu bylo zjištěno, že by se do projektu nemohly jednoduše zakomponovat. Společnost disponuje hotovými řešeními na různých komerčních platformách, případně se řešení vyvíjelo zcela na míru klientovi, a nepočítalo se s obecností, tudíž se tato řešení nedají jednoduše zahrnout do nově navrhovaného systému, avšak v některých případech lze využít alespoň myšlenky a algoritmy. Jedná se o základ pro správu webových stránek, pro kterou byl již v minulosti navržen jednoduchý mechanismus, případně inspirace ohledně funkcionality internetového obchodu, ale žádné hotové části řešení do projektu nebudou použity.

Jiné projekty společnosti využívají systém naprogramovaný v jazyce PHP, který nevyžívá žádný framework, a i když se jedná o účinný základ, který se dá rychle využít s ohledem na budoucí vývoj či spolupráci vývojářů v týmu, je zcela nepoužitelný a zastaralý. Proto je potřeba vše navrhnout zcela od základu. V případě potřeby by se v budoucnu mohlo uvažovat v rámci rozšíření o naprogramování komunikačních rozhraní, čímž by se do projektu dala zahrnout jiná řešení, která by společnost chtěla v budoucnu nabízet, nebo je již nabízí.

Využitím frameworku lze získat výhodu možnosti zahrnutí již hotových a veřejně poskytovaných knihoven pro zvolený framework, což však nebývá podmínkou, protože existují i jiné, obecné knihovny, které se dají využít napříč všemi různými projekty. Navíc ve většině případů jsou tyto knihovny lepším řešením, než které by programátor sám vytvořil. Z těchto důvodů budou do projektu zahrnuty tyto knihovny:

- **Mpdf:** rozšířená knihovna pro generování PDF souborů nejen v Nette frameworku
- **Upload-manager:** knihovna pro práci se soubory a jejich nahráváním na server
- **Ublaboo datagrid:** knihovna pro snadnější manipulaci s daty, která obstarává práci s jednotlivými položkami jako například filtrování, řazení, selektování, vkládání a editaci položek.
- **Nella forms-datetime:** knihovna z projektu Nella pro práci s datem ve formulářích.
- **Mobile detect:** v případě potřeby detekce mobilních zařízení
- **Kdyby Translation:** knihovna pro lokalizaci textů. V případě potřeby překladů, nacházejících se na webové stránce, by měl být navržen vlastní mechanismus, ale pro tvorbu překladů ležících v administraci, je tato knihovna dostačující. Není potřeba, aby měl uživatel možnost překládat texty i pro administrační část, ale lze je nadefinovat skrze statické soubory.
- **Kdyby Facebook:** knihovna sloužící pro uživatelské přihlášení prostřednictvím údajů ze sociální sítě Facebook

- **Carrooi nette menu:** komponenta pro tvorbu menu v administrativní části
- **TinyMCE**: WYSIWYG editor pro přidávání HTML elementů při vkládání obsahu do webové prezentace.
- **Nette AJAX a Nette Forms:** rozšiřující knihovny pro zpracování ajaxových požadavků a práci s formuláři
- **Chartist.js:** JavaScriptová knihovna pro vykreslování grafů v systému

6.3 Analýza rizik

I když jde o projekt jednotlivce, je potřeba identifikovat možná rizika, pokusit se jim předejít, nebo o nich alespoň předem vědět. Pro danou analýzu se využije metoda RIPRAN, ve které bude stanovení hodnoty pravděpodobnosti, že riziko opravdu nastane, odhadnuto na základě zkušeností z předchozích firemních projektů, protože některé z uvedených rizik není ani jinak možné přesně kvantifikovat.

Metoda RIPRAN (*RIsk PRoject ANalysis*) je určena zejména pro analýzu projektových rizik a jejím autorem je B. Lacko. Metoda původně vznikla pro analýzu rizik automatizačních projektů v rámci výzkumného záměru na VUT v Brně. Praxe ukázala, že po určitých úpravách je metodu možno aplikovat pro analýzu rizik širokého spektra různých projektů a v určitých případech i pro analýzu jiných druhů rizik, než jakými jsou projektová rizika. RIPRAN™ je ochranná známka, registrovaná autorem v Úřadu průmyslového vlastnictví Praha pod reg. 283536 (RIPRAN, 2016).

Průběh metody se dělí do 4 postupných kroků, které jsou (RIPRAN, 2016):

1. **Určení nebezpečí a zpracování scénáře** – identifikace rizika
2. **Kvantifikace rizika** –výše škody a pravděpodobnost s jakou ke škodě dojde
3. **Reakce na rizika projektu**
4. **Celkové posouzení rizika projektu**

Pro kvantifikaci rizika se využívají stanovené údaje, podle kterých lze verbálně ohodnotit nepříznivé dopady na projekt (RIPRAN, 2016):

- Velký nepříznivý dopad na projekt – **VD**
    - Ohrožení cíle projektu a jeho celkového dokončení
    - Škoda více jak 20% hodnoty projektu
- Střední nepříznivý dopad na projekt – **SD**
    - Škoda 0,5%-19,5% hodnoty projektu
    - Ohrožení termínů, zásahy do plánů
- Malý nepříznivý dopad na projekt – **MD**
    - Škody do 0,5% hodnoty projektu, malé zásahy do plánu

Verbální kvantifikace hodnocení rizik se skládá ze tří části: Vysoká pravděpodobnost **VP,** kam riziko spadá, pokud je jeho pravděpodobnost výskytu nad 66%. Dále je to střední pravděpodobnost **SP** s rozmezím 33 – 66% a Nízká pravděpodobnost **NP**, kam riziko spadá, pokud je jeho pravděpodobnost pod 33%. Podle těchto údajů se staví převodní tabulka, ze které se získá poslední hodnota druhého kroku – hodnota rizika.

Tab. 1 Metoda RIPRAN - převodní tabulka pro hodnoty rizika

	VD	**SD**	**MD**
VP	Vysoká hodnota rizika **VHR**	Vysoká hodnota rizika **VHR**	Střední hodnota rizika **SHR**
SP	Vysoká hodnota rizika **VHR**	Střední hodnota rizika **SHR**	Nízká hodnota rizika **NHR**
NP	Střední hodnota rizika **SHR**	Nízká hodnota rizika **NHR**	Nízká hodnota rizika **NHR**

Zdroj: RIPRAN, 2016.

Z těchto hodnot byly sestavy následující klíčové hrozby, scénář, který nastane v případě jejich výskytu, pravděpodobnost toho výskytu, ohodnocení dopadu na projekt a celkové hodnota rizika. Jelikož se jedná o projekt jednoho člověka v rámci diplomové práce, byly do následujících hrozeb zahrnuty jen opodstatněné rizika. Následující tabulka č. 2 tedy odpovídá výsledku druhého kroku metody RIPRAN.

Tab. 2 Metoda RIPRAN - určení hrozeb a kvantifikace rizik projektu

ID	**Hrozba**	**Scénář**	**Pravděpod.**	**Dopad**	**Riziko**
1.	Chybný návrh, nepochopení zadání	Zpoždění termínů, chybná implementace	0,30	VD	**SHR**
2.	Zahrnutí kódu třetích stran	Omezení při implementaci	0,25	MD	**NHR**
3.	Zastavení vývoje zvolených frameworků	Nemožnost budoucího rozvoje	0,35	VD	**VHR**
4.	Nenávratnost implementovaného řešení	Ztráty společnosti z prodeje	0,35	VD	**VHR**
5.	Chybně zvolené technologie	Nedokončení projektu	0,15	SD	**NHR**
6.	Projekt jednotlivce	Špatné zabezpečení, chyby v projektu	0,20	VD	**SHR**

Třetí krok spočívá v sestavení konkrétních opatření, díky kterým hodnoty rizika, získané ve druhém kroku, klesnou na nižší úroveň, jež je lépe akceptovatelná. Dále se také stanoví předpokládané náklady, termíny realizace opatření a odpovědná osoba. Do nově sestavované tabulky nemusejí být zahrnuty hrozby, kterým vyšla výsledná hodnota rizika NHR, tedy nízká hodnota. Tyto typy hrozeb se akceptují, jelikož hodnotu již více snížit nelze. Naopak je potřeba klást důraz na hodnoty typu SHR a hlavně VHR, protože mají vysoký celkový dopad na projekt a jejich pravděpodobnost je také vysoká.

Tab. 3 Metoda RIPRAN – návrhy na opatření

Č. Rizika	**Návrh na opatření**	**Náklady, termín, odpovědnost**	**Nová hodnota rizika**
1.	Neustála konzultace se zadavatelem Iterativní vývoj systému v kombinaci s jinými osvědčenými přístupy	Bez nákladů 31.10.2016 Jan Vodák	**NHR**
3.	Nalezení informací ohledně budoucnosti vývoje Volba kvalitní technologie se širokou komunitou uživatelů	Bez nákladů 31.10.2016 Jan Vodák	**SHR**
4.	Předimplementační příprava Volba kvalitního obchodního zástupce Analýza požadavků zákazníků	Bez nákladů 31.3.2017 Jan Vodák	**SHR**
6.	Sebevzdělávání Navázání spolupráce s odborníky z praxe Průběžná kontrola výsledků práce	Bez nákladů 30.4.2017 Jan Vodák	**NHR**

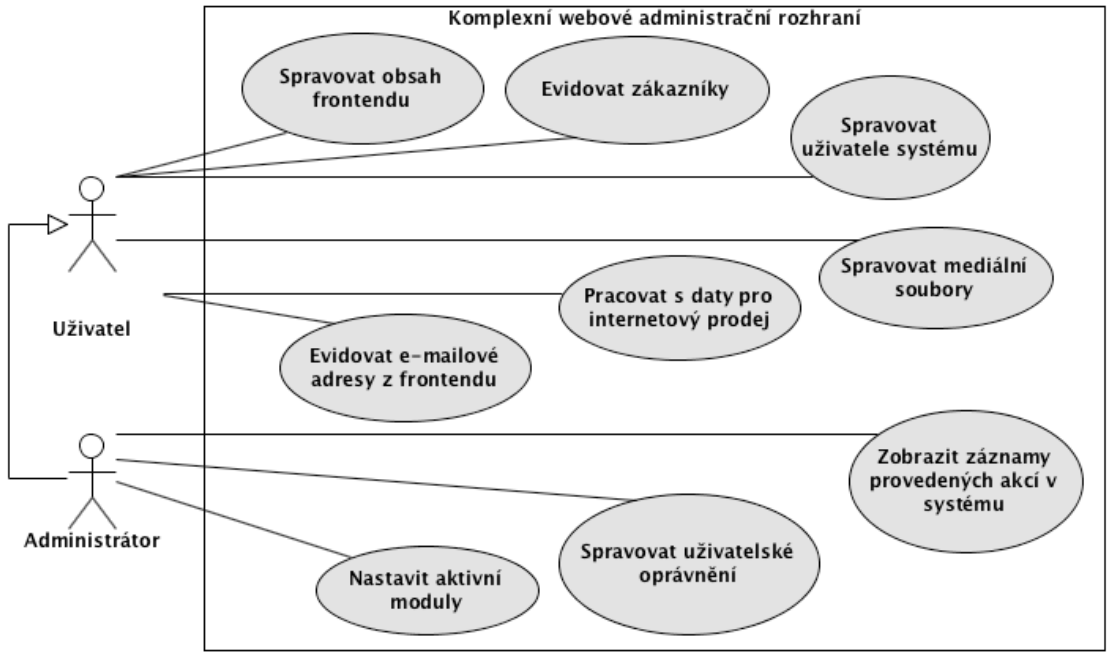
Již bylo zmíněno, že se akceptují jedině hodnoty s nízkou hodnotou rizika. Podle výsledků lze soudit, že se podařilo téměř u všech hrozeb snížit tuto hodnotu na požadovanou úroveň, avšak s přihlédnutím na míru abstrakce při odhadu se jedná pouze o ilustrační hodnoty. Jedinými hrozbami, které zůstaly pořád relativně vysoké, jsou nenávratnost navrženého řešení a ohrožení plynoucí z volby technologie (frameworku). Tyto možnosti ani nelze na požadovanou úroveň optimalizovat, protože budou existovat vždy. Nikdy není garantováno, že vývoj zvolené technologie se jednoho dne nezastaví, ale s přihlédnutím na životnost internetových aplikací, vývojářskou komunitu a množství projektů v České republice, využívající zvolený framework, je tato hrozba pro společnost přijatelná. Návratnost jakéhokoliv projektu není také nikdy garantována, avšak zkušenosti odpovědných lidí společnosti ukazují, že je tato hrozba také přijatelná, což je podpořeno také faktem, že projekt je vyvíjen v rámci akademických účelů.

7 Návrh systému

Pro základní návrh systému je potřeba převést slovní analýzu systému do standardizované podoby, která využívá při objektově orientované analýze a návrhu modelovací techniky založené na **jazyku UML** – *Unified Modeling Language*. UML byl vytvořen v polovině 90. let a v současnosti je v této oblasti standardem. Jeho velkou výhodou je nezávislost na procesu vývoje, protože není svázán s žádnou konkrétní metodikou. UML je v podstatě jedinou používanou objektovou notací, která je podporována naprostou většinou CASE nástrojů a na UML je postavena většina objektově orientovaných metodik. Tento jazyk umožňuje prostřednictvím různých typů diagramu zachytit systém z různých pohledů a na různé úrovni abstrakce (Bruckner, 2012).

Pro popsání chování systému z hlediska uživatele lze využít diagram případů užití, neboli ***Use Case Diagram***. V tomto typu diagramu lze zachytit, jaké typy uživatelů používají systém a jaké činnosti vykonávají. Prvky diagramu jsou aktér (*Actor*), případ užití (*Use case*) a vztah (*Relation*). Aktér představuje prvek okolí systému, který komunikuje se systémem a nemusí se nutně jednat jen o živé osoby, ale také o externí systém, se kterým modelovaný systém komunikuje. Aktér hlavně vymezuje základní roli, kterou hraje člověk, hardwarové zařízení nebo externí systém ve vztahu k modelovanému systému (Bruckner, 2012).

Z předchozích kapitol o specifikaci požadavků na systém a následné vlastní analýzy byl sestaven následující Use Case Diagram, jenž zachycuje základní požadavky na funkcionalitu z pohledu uživatele. Dílčí případy užití, zachycené na následujícím obrázku, představují základní uživatelské operace, které můžou definovaní uživatelé se systémem vykonat. Jednotlivé případy užití vyjadřují základní úroveň, přičemž další vnořené interakce se dají zobrazit při dílčím rozkladu činnosti do nižších vrstev. Jednotlivými aktéry jsou Administrátor, tedy uživatel s nejvyšším oprávněním, a Prostý uživatel. Role Administrátor je ve vztahu typu generalizace k Prostému uživateli, což znamená, že přebírá všechny případy užití od tohoto typu role. Samozřejmě v systému může být nadefinováno více typů rolí, které budou mít oprávnění pouze pro specifické případy užití, avšak pro počáteční návrh je podstatné rozlišit alespoň tyto dvě role, kde Administrátor by měl být typ role pro tvůrce systému a Uživatel zákazník, pro kterého je systém přizpůsoben. Z toho vyplývá, že N*astavení modulů*, *Správa uživatelského oprávnění a Zobrazení záznamů o uživatelských akcích* budou přístupné jen pro tvůrce systému, nikoliv pro zákazníky. Vyplývá to z předpokladu, že zákazník nemůže mít oprávnění pro aktivaci nezaplacených modulů, kontrola uživatelských akcí může sloužit jako takzvaná „*zadní vrátka*“, která například v případě stížnosti klienta na chybějící data mohou dokázat, že případné smazání provedl konkrétní uživatel. Avšak v případě potřeby se dá tato sekce zpřístupnit i zákazníkovi.



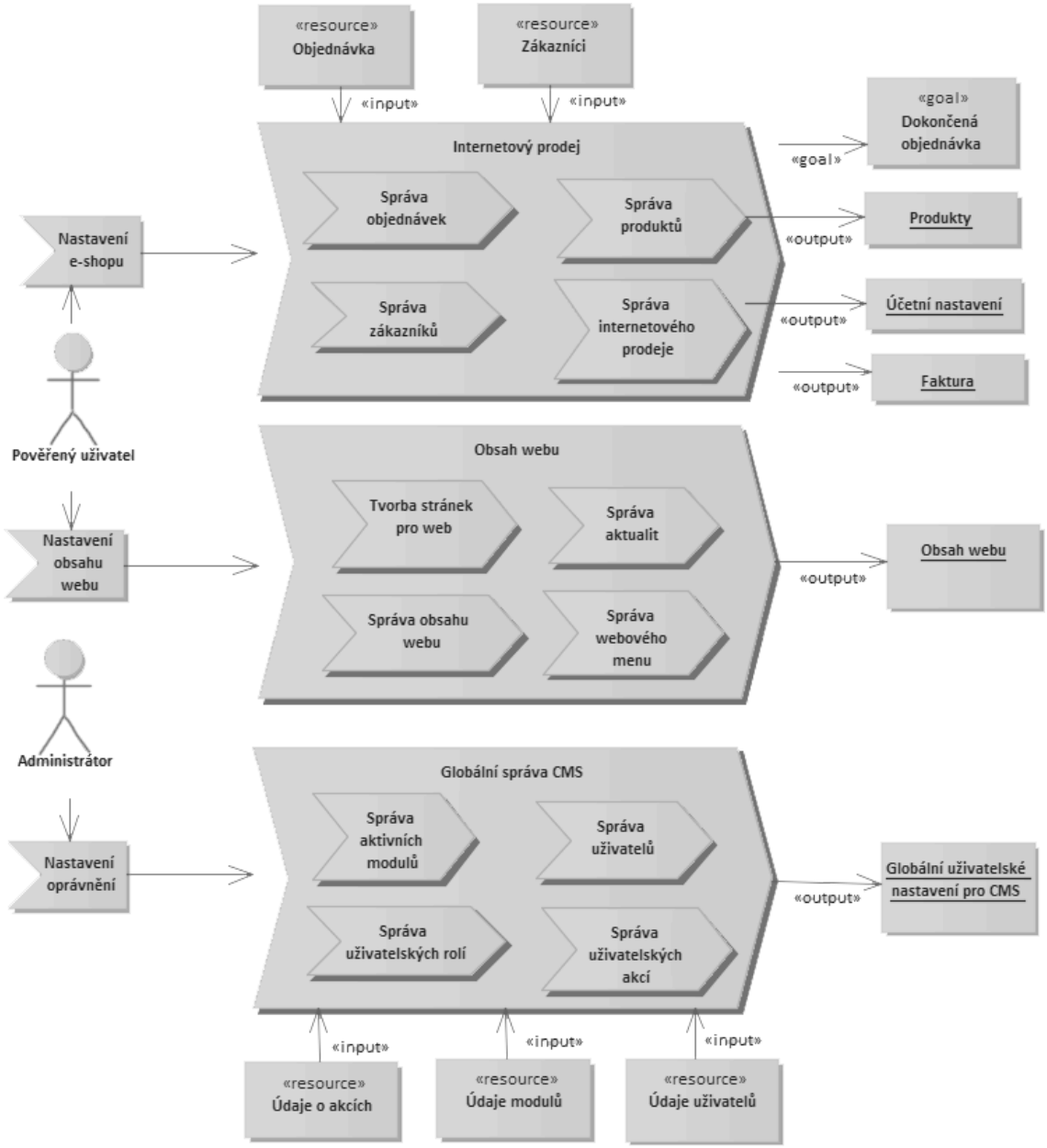
Obr. 3 Základní Use Case Diagram systému

Další ze zástupců z řady technik pro modelování je **procesní modelování**, které využívá mnoho různých notací, obvykle závislých na konvenci v daném byznysu, na konvenci dodavatele softwaru a na použitém modelovacím nástroji. Standardizace pro procesní modelování není tak silná jako v případě UML, zejména proto že se jedná o modelování mnohem měkčího systému než programového. Striktní konvence modelu v takovém případě často velmi svazuje, což vede k tomu, že se různé důležité nuance v byznysu zachycují nestandardizovaně. Na druhou stranu je striktní konvence nutná, aby byl zachován jednotný smysl procesního modelu a jeho použitelnost (Bruckner, 2012).

Ze zástupců procesního modelování byla zvolena **Eriksson-Penkerova notace**, která slouží jako rozšíření UML a pokouší se vyřešit nevhodnost standardu UML pro procesní modelování. Některé z UML diagramů, například diagram aktivit, se hodí pro modelování některých aspektů procesu, jako je modelování návaznosti činností, avšak neumožňuje vhodně modelovat některé byznys aspekty procesů (například kvantitativní cíle procesu). Eriksson a Penker navrhují modelovat proces jako celek a sledovat u něj vstupní a výstupní objekty, kontrolní a podpůrné zdroje a hlavně cíl procesu. Tento přístup pak lze kombinovat se standardním diagramem aktivit UML pro zobrazení toku činností v rámci procesu, případně pro předávání událostí mezi procesy (Bruckner, 2012).

Diagram základních procesů první vrstvy navrhovaného systému je znázorněny na následujícím obrázku. Mezi základní procesy patří *Internetový prodej*, *Správa obsahu webu* a *Globální správa CMS*. Tyto procesy opět obsahují řadu dalších, menších procesů, takže se opět dají rozložit na nižší úrovně. Opět jsou zde zachyceny základní role, které tyto procesy budou v systému vykonávat prostřed-

nictvím událostí. Pro jednotlivé procesy jsou použity zdroje, informace a navíc tyto procesy mají svůj výstup, nebo pravidlo.



Obr. 4 Eriksson-Penkerův diagram základních procesů systému

Výstupem procesu pro správu produktů je samotný produkt, správy internetového prodeje je účetní nastavení a výstupem správy objednávek je faktura. Zdrojem tohoto procesu jsou údaje o zákaznících a objednávky a podmínkou je dokončení objednávky z frontendové části zákazníkem.

7.1 Rozvržení do modulů

Systém bude rozdělen na veřejně přístupnou část (*frontend modul*), který je přístupný všem uživatelským rolím, počínaje základní rolí typu *guest* (prostý návštěvník webu) a privátní část (*backend),* do které budou mít naopak přístup pouze uživatelé s vybranými rolemi. Privátní část se dále rozdělí do dílčích modulů, které vyplývají ze základního návrhu systému, znázorněném prostřednictvím Use Case diagramu na Obr. 3. Tyto moduly jsou:

- Blog modul
- CMS modul
- Core modul
- Email modul
- Eshop modul
- Media modul

Blog modul bude sloužit pro vkládání příspěvků do klientské částí systému a jedná se vlastně o modul pro vkládání aktualit. Tento modul by mohl být klidně součástí CMS modulu, avšak bude lepší jej osamostatnit, což usnadní budoucí možnost deaktivace v případě jeho nevyužití, avšak návrh tohoto modulu bude součástí CMS modulu. Media modul slouží pro správu všech nahraných obrázků na server, email modul naopak obstará práci se sesbíranými emailovými adresami zákazníků. Složitost těchto modulů je poměrně jednoduchá a jejich velikost není příliš velká, což je dáno tím, že plní úzký okruh činností. Více složitější jsou moduly CMS, Eshop, Front a Core, protože jejich okruh činností je mnohem více rozsáhlejší.

Označit media modul za jednoduchý se může jevit poněkud sporně, protože jeho složitost může být naopak velmi komplikovaná, avšak v základním návrhu bude vykonávat pouze jednoduchou správu médií o které bylo pojednáváno v podkapitolách 5.1 – správa médií a 5.2 – nahrávání fotek pro produkty, stránky a aktuality případně pro nahrání obrázku jako pozadí stránky. Základní správa bude tedy obsahovat přehled nahraných souborů (obrázků), editaci SEO elementů jako je popisek a zástupný text obrázku, nastavování hlavního obrázku z již nahraných obrázků, případě se bude kontrolovat velikost nahrávaných souborů při jejich nahrávání. Kromě zmíněného výčtu prvků, které budou obsahovat své obrázky, mohou být v rámci rozšiřování systému vytvořený nové prvky, které budou mít také své vlastní obrázky. Z tohoto důvodu musí být pozdější implementace dostatečně obecná a počítat s tímto faktorem.

7.1.1 Core modul

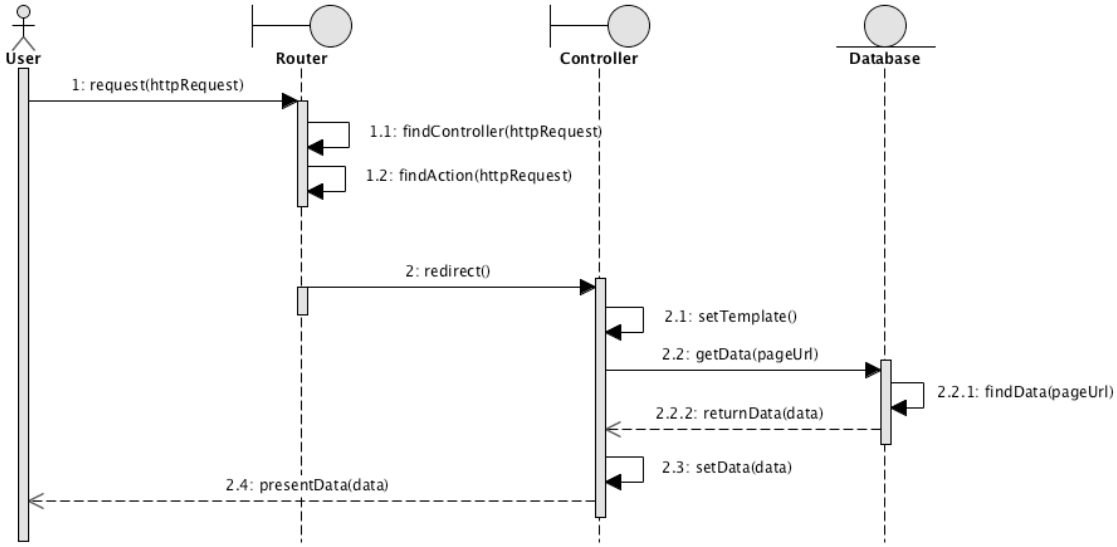
Tento modul bude sloužit, jak název napovídá, jako jádro - bude se přes něj přistupovat do privátní části systému, a navíc bude obsahovat základní funkcionalitu, která musí být obsažena v jakémkoliv projektu, který bude na tomto systému postaven. Případy užití tohoto modulu jsou *Nastavit aktivní moduly*, *Spravovat uživa-*

telské oprávnění a *Zobrazit záznamy provedených akcí v systému*, které jsou zobrazeny na Obr. 3. Tyto případy užití byly tedy sjednoceny do tohoto moduly a dále budou obohaceny o další funkcionalitu systému.

Z předpokládaného rozdělení systému na veřejně přístupnou a privátní část, nebo dále také na sekce, které budou přístupné jen vybraným uživatelským rolím, vyplývá, že bude potřeba kontrolovat, zda uživatel je přihlášený, a pokud bude přihlášen, jestli má potřebné oprávnění pro vstup a zobrazení požadované části systému. Proces pro ověření totožnosti uživatele se odborně nazývá **autentizace** a má za úkol ověřit, že uživatel je skutečně ten, za koho se vydává. Proces autentizace bude založen na databázi, ve které budou uloženy uživatelské údaje (přihlašovací jméno a heslo), jejichž kombinaci bude znát jen daný uživatel. Proces ověření uživatelského oprávnění se nazývá **autorizace** a navazuje na autentizaci. Přihlášenému uživateli bude přidělena konfigurovatelná role, které je přiděleno oprávnění pro přístup do sekce.

Při návrhu a pozdější implementaci se musí brát do úvahy zvolená technologie - Nette framework, který využívá architekturu MVC. Daná funkcionalita byla popsána v podkapitole 3.3, avšak pro navržení mechanismu autentizace a autorizace je potřeba správného pochopení. Pro následné popsání požadavku od uživatele na zobrazení stránky bude využit UML sekvenční diagram. Tento typ diagramu graficky znázorňuje průběh zpracování procesu v podobě zasílání zprav. Zprávy si posílají většinou objekty, ale komunikovat mohou i třídy nebo aktéři. Základní charakteristikou objektu je jeho schopnost přijmout zprávu a reagovat na ni – spuštění příslušné metody. V rámci této metody objekt může poslat zpráva dalšímu objektu, a tak vzniká sekvence zpráv, jež je právě zachycena pomocí tohoto typu diagramu (Bruckner, 2012).

Na obrázku jsou znázorněni aktér *Uživatel* a objekty *Router*, *Kontroler* a *Databáze*. Pro dodržení autentičnosti jsou pojmenovány i se zprávami anglicky, jelikož i v pozdější implementaci bude vše pojmenováno anglicky. Diagram obsahuje dva typy zpráv – plná čára s plnou šipka na konci značí synchronní zprávy, kdy vysílající objekt čeká na ukončení zpracování zprávy. Šrafovaná čára značí asynchronní návratovou zprávu, kdy vysílající objekt naopak na návratovou zprávu čekat nemusí. Na obrázku lze tedy vidět, že počáteční uživatelský požadavek přichází na *Router*, který jej zachytí a zpracuje. Podle zpracovaného požadavku se najde příslušný *Kontroler*, ve kterém se případně naleznou požadovaná data v *Databázi* a těmito daty se naplní šablona, která se na závěr zobrazí uživateli skrze okno webového prohlížeče.

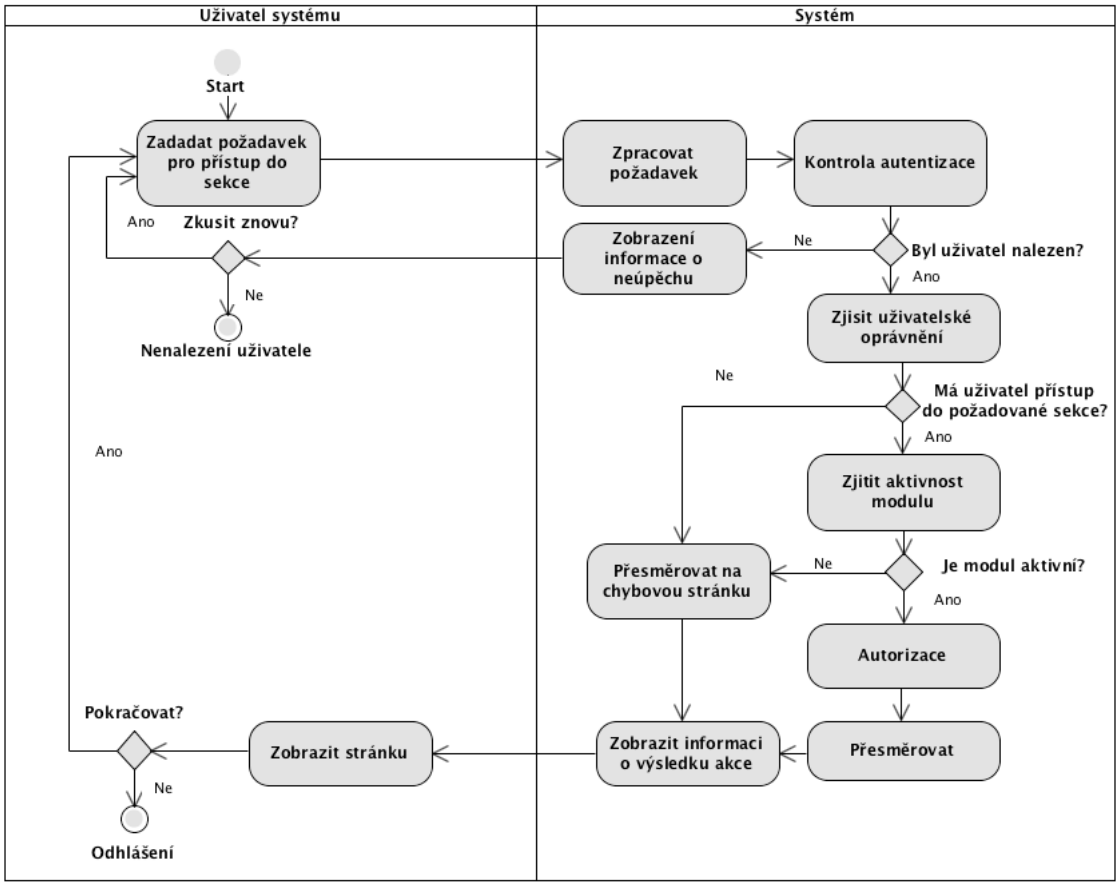


Obr. 5 Sekvenční diagram pro požadavek na zobrazení stránky

Proces autentizace a autorizace je tedy využit i v procesu zobrazení stránky systémem a bude následně znázorněn v diagramu aktivit, který rovněž patří do UML. Tento typ diagramu umožňuje zachytit posloupnost aktivit, které mohou probíhat jak sekvenčně, tak paralelně. Lze jej využít pro modelování byznys procesů, modelování logiky scénáře případu užití či modelování logiky byznys pravidel. Jde o objektově orientovanou alternativu vývojových diagramů. Prvky diagramu jsou: zahájení, ukončení, jednotlivé aktivity, tok, rozhodování a větvení. Pokud je potřeba zachytit, kdo danou aktivitu provede, lze k tomu využít prvek takzvané plavecké dráhy, čímž se aktivity seskupí do souvislých pruhů podle nositele aktivity (Bruckner, 2012).

Již v tomto bodě lze vymezit takzvané „*plavecké dráhy*“ pro aktéry, kteří budou vykonávat jednotlivé aktivity a budou jimi *Uživatel* a *Systém*. *Uživatel* celý proces zahájí svým požadavkem na *Systém* o zobrazení určité sekce, *Systém* mu ji zobrazí, ale ještě předtím provede ověření zmíněné autentizace a autorizace. Tyto činnosti se budou ověřovat při každém požadavku ještě před načtením požadovaných dat z databáze v každém kontroleru. Bylo by proto vhodné využít základní prvek objektového programování – **dědičnost** a nadefinovat pro kontrolery společného předka. Konkrétní návrh funkčnosti je zobrazen na následující straně.

Z následujícího obrázku lze jednoduše vyčíst posloupnost kroků, které přechází zobrazení konkrétní stránky. V případě neúspěchu při autentizaci či autorizaci bude následovat zobrazení chybové hlášky a přesměrování na předem připravenou stránku pro tento případ, nebo naopak v případě úspěchu se uživateli zobrazí požadovaná stránka. Nejprve se provádí kontrola oprávnění a až poté se provádí kontrola, zda je modul vůbec aktivní. Je to dáno převážně tím, že i když bude sekce aktivována, uživatel do ní vůbec nemusí mít přístup – proto se kontrola oprávnění provádí dřív.

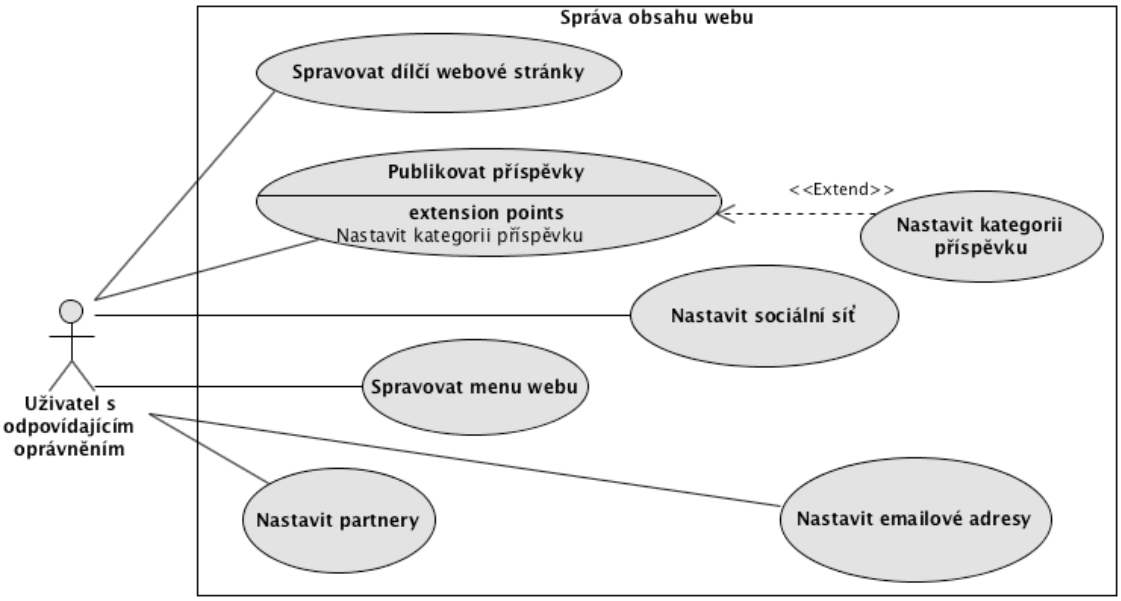


Obr. 6 Diagram aktivit pro kontrolu autentizace a autorizace

Kromě těchto zmíněných kontrol bude *Core modul* obsahovat další, již zmíněnou základní funkcionalitu. Do této funkcionality bude patřit práce s uživateli, tedy možnost uživatele vytvořit a ručně mu nastavit jeho roli. Role se bude dát také manuálně vytvořit a nastavit jí přístupy do konkrétních sekcí. Při tomto nastavování se opět musí brát v potaz zvolená technologie. V Nette frameworku existuje možnost role definovat spolu se zdroji a akcemi v konfiguračních souborech a pokaždé, když se v systému vytvoří nový zdroj nebo akce, musí se ručně nadefinovat právě v těchto souborech. Této možnosti by však bylo dobré se vyvarovat, zdroje s akcemi automaticky ukládat do databáze a uživatel by poté jen ručně v administraci nastavil dílčí oprávnění pro dané role. Pro automatické načtení zdrojů a akcí se vytvoří takzvaný *autoloader*, který projde všechny soubory a podle zvláštního případu anotace je uloží do databáze. Uložení proběhne jen jednou, uloží se vždy název akce/zdroje a k němu se také uloží jeho popisek pro usnadnění pozdější manipulace s těmito daty. Bude se vycházet již ze zmíněného pravidla, že cokoliv není povoleno, je zakázáno. Uživateli vystačí vždy jen jedna role, čímž se situace značně usnadní, jelikož v případě více rolí by mohlo docházet ke konfliktům, kdy nějaký zdroj by v jedné roli byl povolen a v jiné zase ne, avšak vzhledem na rozsah navrhovaného systému je to přijatelné zjednodušení.

7.1.2 CMS modul

Tento modul je de facto hlavní prvek systému a jeho primární úlohou bude obstarat správu obsahu webu - zde se budou vytvářet a spravovat jednotlivé stránky frontendového modulu a z těchto stránek se bude definovat i menu. Základní funkcionalitu tohoto modulu znázorňuje následující Use Case diagram.



Obr. 7 Use Case Diagram pro správu obsahu webu

Kromě zmíněné tvorby stránek pro frontendovou část se zde také budou vytvářet a modifikovat i další položky, které s tvorbou obsahu souvisejí. Jedné se o tvorbu aktualit, jež budou vyčleněny do samostatného modulu, ale co se týká zařazení, patří sem a z uživatelského hlediska toto začlenění nebude rozpoznatelné. Dále se pro web budou vytvářet odkazy na sociální sítě a partnery, které budou ve formě obrázku a odkazu na příslušnou webovou stránku, jak bylo specifikováno v požadavcích společnosti. Posledním z daných nastavení je práce s e-mailovými adresami, které určí, kam se budou odesílat e-maily z kontaktního formuláře, nebo v případě vytvoření nové objednávky atp. Vazba typu *extend* značí rozšíření daného případu užití o novou funkcionalitu. Uživatel sice přistupuje a vyvolá původní akci, která ovšem vyvolá akci rozšiřující. Jedná se samostatný případ užití, který na původní akci nemá žádný vliv a v případě nefunkčnosti či neexistence by se původní volaný případ užití tak či tak provedl. O existenci rozšíření původní případ užití nemusí ani vědět.

Při tvorbě jednotlivých stránek pro webovou prezentaci by nemělo být zapomínáno, že ve velmi mnoha případech existují dílčí podstránky jednotlivých stránek. Jinými slovy, podstránka může být potomek jiné stránky, která je naopak její rodič, tedy nadřazená položka, což má za následek definování vztahu rodič-potomek. Ve výpisu menu na frontendové částí se potom využije takzvané *stromo-*

vé menu, jež je obdobné jako pro výpis kategorie produktů, kde se tento vztah musí také navrhnout. Jednotlivé kategorie by také měly být navrženy i pro modul aktualit/blogu, čímž se usnadní pozdější filtrování ve veřejně přístupné části. Dále může nastat situace, že dílčí stránka nemusí být dynamická a editovatelná přímo z administrace, nýbrž odkazuje na statickou stránku, která je již předdefinována. Zmíněný příklad se může týkat třeba výpisu produktů, nebo stránka může také odkazovat na produkt, který by mohl být ve specifických případech zařazen do menu jako jeho položka.

Z těchto zmíněných důvodů je vhodné tvorbě menu věnovat jistou pozornost, jelikož prosté zatrhnutí, zda je stránka položkou menu by nemuselo být dostačující. Navíc se velmi často objevují případy, ve kterých je potřeba na webové stránce vytvořit více typů menu, kdy položky hlavního menu, umístěného například v hlavičce webové stránky, nemusejí odpovídat položkám jednoduššího menu umístněného v patičce. Proto je vhodné vytvářet jednotlivé menu jako objekty, do kterých budou zařazovány jednotlivé stránky, příspěvky blogu, produkty nebo statické odkazy na akce presenterů. Tyto menu se poté budou uživatelům snáze editovat.

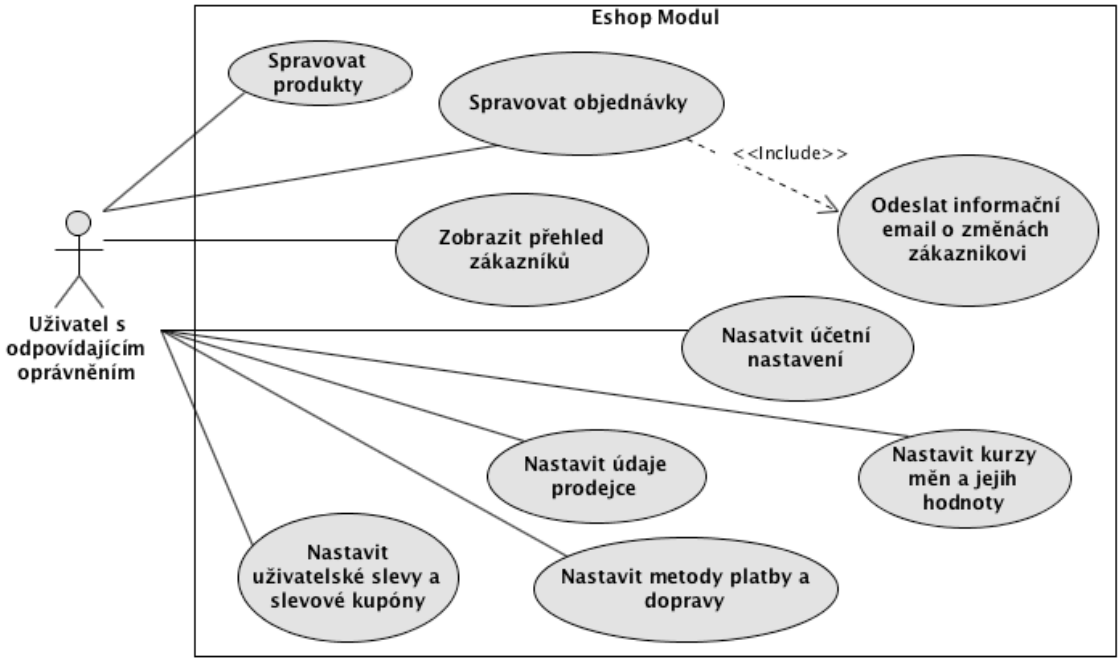
Součástí tvorby jednotlivých stránek bude také práce se SEO elementy, konkrétně s titulkem, popiskem a klíčovými slovy. Z tohoto důvodu by se měly vytvořit všechny dostupné stránky z webové prezentace v příslušné sekci a to i v tom případě, že budou odkazovat na již připravenou statickou šablonu. Pro každou ze stránek se zmíněné hodnoty jednoduše nastaví a poté v šabloně vypíší. V případě, že se hodnoty v administraci nevyplní, využijí se předem připravené, obecné textové řetězce, které budou staticky definovány v příslušných metodách pro předávání dat do jednotlivých šablon. Ve výchozí metodě pro nastavení těchto hodnot se tedy nejprve pokusí najít příslušná stránka v administraci, v případě nalezení se doplní hodnoty z administrace, jinak se doplní obecné popisky.

7.1.3 E-shop modul

Obsahem tohoto modulu budou veškeré činnosti, jež spadají pod položku internetového prodeje. Jedná se o volitelný typ modulu, jelikož ne všichni, kteří osloví společnost IDEATECH, budou chtít prostřednictvím svého webu prodávat. Proto by měl jít tento model snadno deaktivovat. Hlavní činnosti jsou znázorněny na následujícím Use Case diagramu. Diagram zachycuje základní případy užití. Některé lze opět hierarchicky rozložit na nižší úrovně. Například *Spravovat produkty* obsahuje další případy užití, jakými jsou *Vytvoření produktu*, *Vytvoření variant produktu, Vytvoření galerií produktu*, *Vytvoření atributů produktu* a *Zařazení produktu do kategorií*.

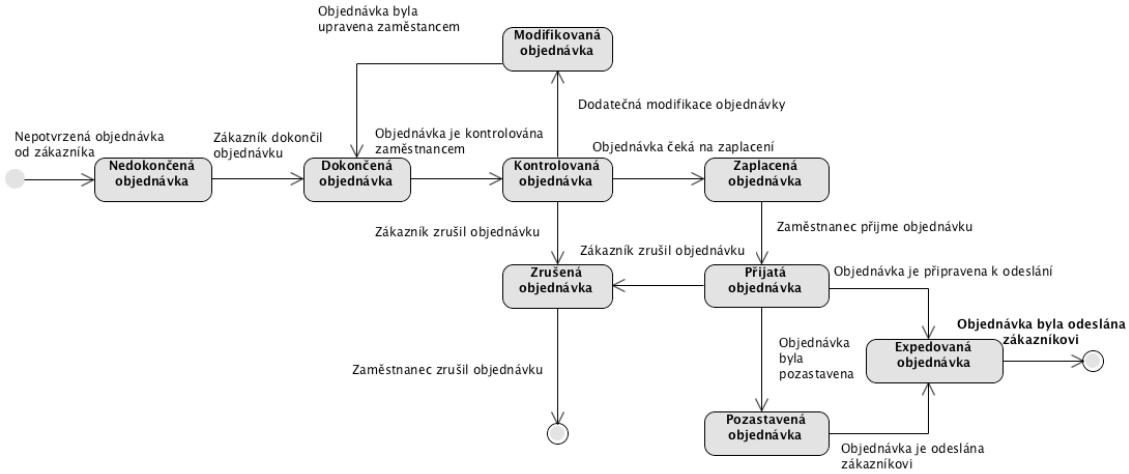
Již bylo zmíněné ve specifikaci požadavků, že některé produkty nemusejí být jen jednoduché, ale například jeden produkt se může vyskytovat ve více variantách a zákazník si z těchto variant může vybrat. Bylo by velmi nevhodné, kdyby při výpisu produktů byl tentýž produkt vypsán vícekrát, pokaždé v jiné variantě, protože se de facto jedná o tentýž produkt. Každá varianta však může mít jiné vlastnosti, různou cenu či obrázky. Naopak hodnocení by mělo zůstat vždy stejné, jelikož se

jedná vždy o jeden a tentýž produkt. Proto je potřeba k produktům vytvářet atributy, které k jednotlivým produktům bude možné později přiřadit, a tím vytvořit jejich varianty. Z jednoduchého produktu se tedy může stát produkt s variantami podle toho, zda mu bude některý z atributů přidělen. Z jednotlivých atributů se tedy vytvoří další produkty, jež jsou k původnímu produktu ve vztahu rodič-potomek a budou vytvořeny na základě těchto atributů.



Obr. 8 Use Case Diagram pro správu internetového obchodu

Případ *užití Spravovat objednávky* je rozšířen o funkcionalitu *Odeslat informační e-mail* o *změnách zákazníkovi*. Vztah mezi těmito případy užití je *include*, což znamená, že pokud by toto rozšíření nefungovalo, nefungoval by i původní případ užití. Z toho vyplývá, že jsou na sobě přímo závislé a případ *užití Spravovat objednávky* by bez tohoto rozšíření nebyl funkční. Je to dáno hlavně tím, že pokud se objednávka jakkoliv změní, musí o tom být uživatel informován skrze e-mail. Dále lze tento případ užití opět rozložit na nižší úroveň, jelikož ve správě objednávky se budou moci *Upravit údaje zákazníka*, které se nepromítnou nikam jinam než do údajů objednávky, bude zde možnost *Přidat další produkty do objednávky*, nastavit manuálně nebo automaticky datum vystavení a splatnosti faktury. Mimo tyto možnosti zde bude velmi důležitý proces *Nastavení stavu objednávky*. Těch může mít objednávka více, avšak vždy bude pouze jeden proces, který nastaví systém buďto automaticky, nebo jej nastaví uživatel podle toho, zda zákazník zaplatil, zda je objednávka hotová, nebo naopak zrušena. Všechny tyto stavy a přechod mezi nimi je znázorněn na následujícím stavovém diagramu.



Obr. 9 Stavový Diagram pro objednávku

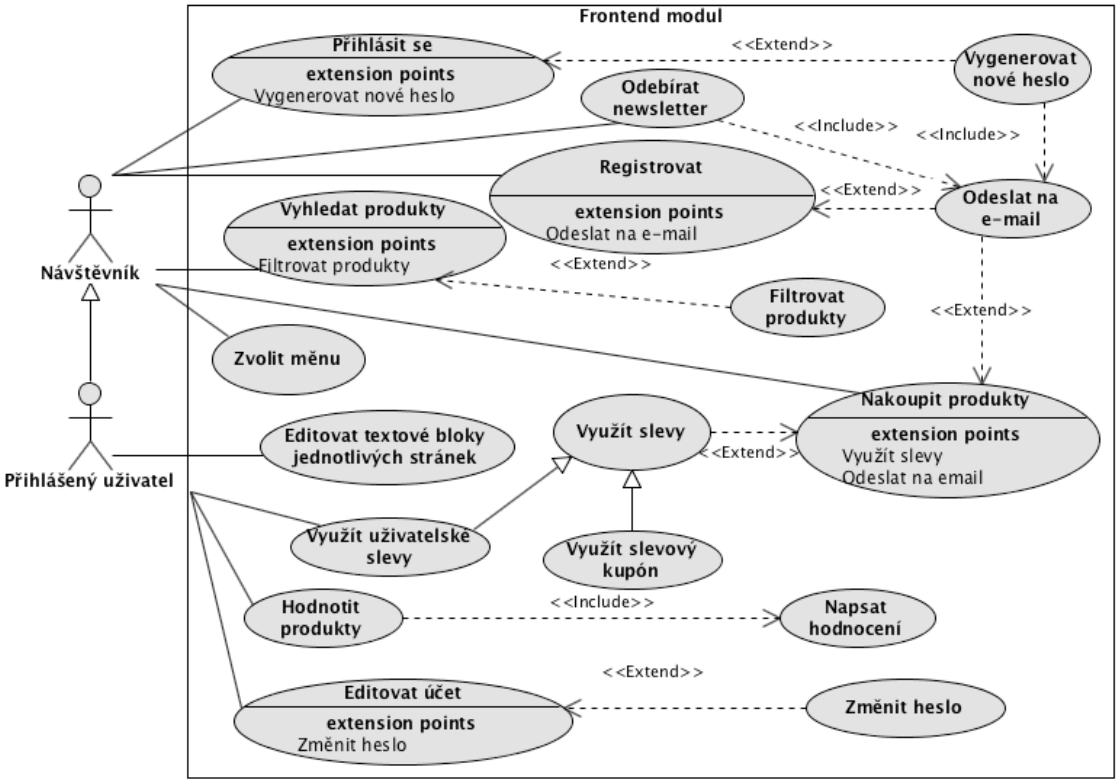
Z předchozího diagramu vyplývá velké množství stavů, ve kterých se objednávka může vyskytovat. Tyto stavy lze zredukovat jen na pět modifikovatelných stavů: **Nová**, **Přijatá**, **Expedovaná**, **Pozastavená**, **Zrušená**. Existuje však ještě dodatečný šestý stav, což je stav pro košík, v něm se objednávka nachází pokud není dokončena, a proto tyto objednávky nejsou zahrnuty do přehledu objednávek v administraci. Do tohoto přehledu se objednávka dostane až v momentě, kdy ji uživatel potvrdí a objednávka se stala dokončenou. Tento stav objednávce automaticky nastaví systém. V databázi tedy může existovat mnoho položek, které nebudou do administrace zahrnuty a mohou se využít pro pozdější analytické účely, například pro zjištění, které produkty byly přidány do košíku nejčastěji, avšak objednávka nebyla dokončena, nebo jestli se uživatel dostal alespoň k procesu volby metod dopravy a platby atp. Z tohoto tedy jasně plyne, že košík je vlastně jen nedokončená objednávka, která je určitým způsobem spárována se zákazníkem.

7.1.4 Front modul

Posledním ze zmíněných modulů je ten, pro který není nastaveno žádné omezení přístupu a je tedy veřejně přístupný komukoliv. Do této části se propisují data, která jsou vytvářena a modifikována z privátní sekce, a proto odráží možnosti, jež poskytuje kompletní administrace. Na rozdíl od ostatních modulů se tato část bude velmi často měnit a přizpůsobovat podle potřeb a požadavků klienta, nehledě na to, jestli se bude jednat o potřeby týkající se vzhledu nebo jiné modifikace. Velmi výhodné by však bylo tuto část navrhnout co nejvíce obecně, aby i tady docházelo ke změnám co nejméně a například pro vzhled by mohla být navržena výchozí šablona, která by se jen minimálně upravovala v případech, kdy klient nebude vyžadovat originální vzhled této části. Návrh případů užití a jeho aktérů pro tento modul je znázorněn na následujícím use case diagramu.

Na první pohled se může zdát, že proti předešlým use case diagramům je tento návrh poměrně složitější, avšak to je dáno převážně tím, že ostatní diagramy vyjadřují první úrovně návrhu a dají se dále rozkládat na nižší vrstvy. Tento návrh

se už poměrně moc rozkládat nedá a zahrnuje téměř kompletní přehled uživatelských případů užití. Hlavními aktéry jsou *Návštěvník*, tedy uživatel, který není přihlášen, a naopak *Přihlášený uživatel*, který má stejné možnosti, ale navíc získává možnost *Hodnotit produkty*, *Využívat uživatelské slevy* a v případě potřeba také *Editovat svůj profil.* Nepřihlášený i přihlášený uživatel budou mít možnost *Vyhledávat produkty*, *Nakupovat produkty*, mít možnost *Volby peněžní měny*, ve které budou chtít nakupovat a tím pádem se i ceny automaticky přepočítají dle nastaveného kurzu, *Odebírat novinky* nebo se *Přihlásit*. Pokud uživatel zapomene heslo, existuje zde možnost *Vygenerovat heslo nov*é a odeslat jej na zadaný e-mail. Tento rozšiřující případ užití je zde využit pro více možností v různých vazbách (*include* i *extend*). V návrhu je možné vidět i vazbu *generalizace* mezi případy užití. Jedná o možnost využít slevy jednak uživatelské, nebo prostřednictvím slevových kupónů. Vždy se jedná o slevy, ale pokaždé se budou v některých detailech lišit.

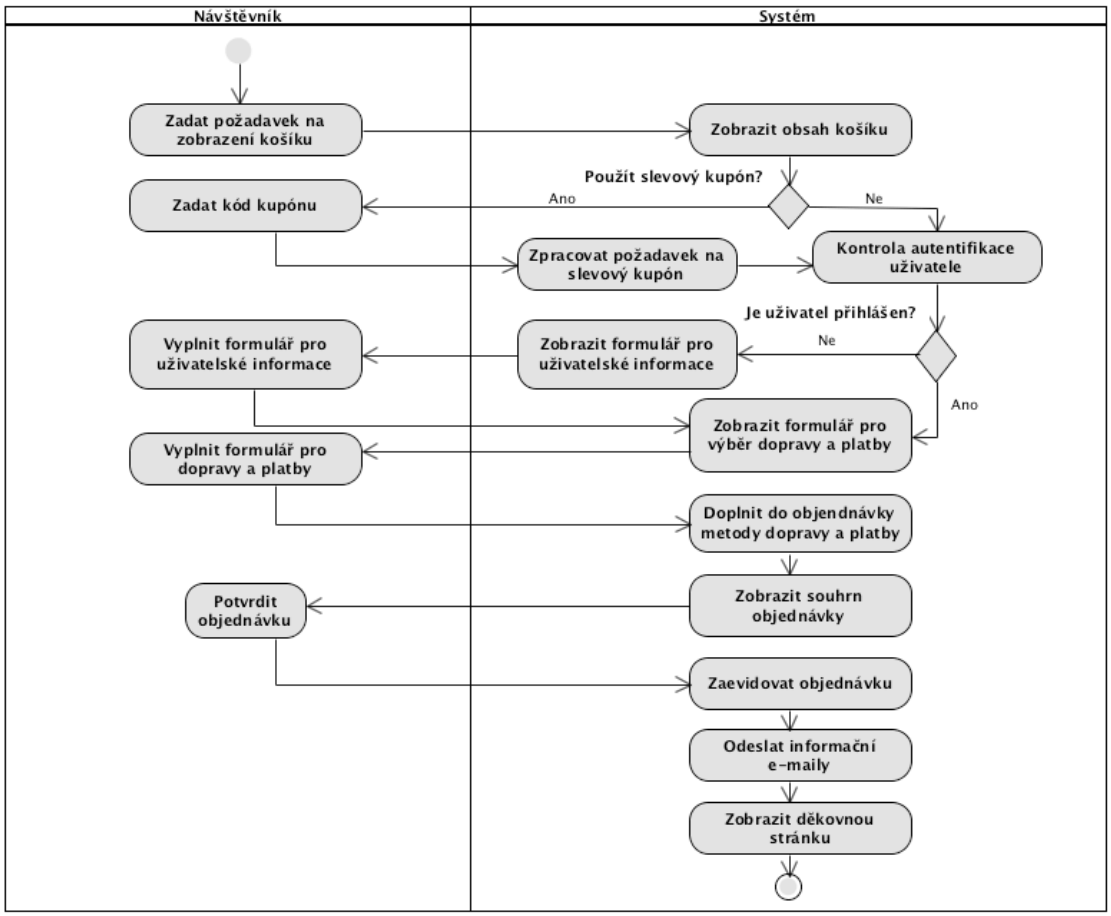


Obr. 10 Use Case Diagram pro veřejně přístupnou část

Ve specifikacích společnosti v podkapitole 5.2 bylo pojednávalo o problému, kdy jednotlivé webové stránky obsahovaly větší počet editovatelných bloků a vytvoření patřičného počtu formulářových polí v editaci jednotlivých stránek a podstránek v CMS modulu bylo již v dané podkapitole označeno za neoptimální řešení. Tento problém se dá vyřešit využitím *WYSIWYG* knihovny *TinyMCE*, kterou pomocí JavaScriptového nastavení lze využít pro takzvanou „*inline editaci*“. Lze poté tedy dosáhnout požadovaného efektu, že přihlášený uživatel s příslušným oprávněním

má možnost přímo ve webové stránce tyto bloky editovat. Předpokládaná funkcionalita potom je taková, že v jednotlivých šablonách budou vypsány výchozí textové hodnoty, které po kliknutí na ně spustí editaci a jakmile proběhne zmíněná editace, nové hodnoty se uloží do databáze, odkud se vždy od tohoto okamžiku budou vypisovat do šablony namísto výchozích hodnot. Nemusí se jednat pouze o textové hodnoty, jelikož knihovna umožňuje vkládání HTML elementů do prostého textu. Na první pohled by se mohlo zdát, že tato funckionalita by měla spadat pod CMS modul, avšak jedná se o funkcionalitu Front modulu, protože právě zde se daná funkcionalita bude vykonávat.

Jak funguje košík již bylo nastíněno v návrhu modulu pro internetový obchod (*Eshop modul*), avšak samotné provedení nákupu je samo o sobě také poměrně komplikovaným procesem a bylo by vhodné jej trochu více rozebrat. Tento proces se odehrává v uživatelské části systému, tudíž jeho návrh patří sem. Tento proces je zachycen na následujícím diagramu aktivit.



Obr. 11 Diagram Aktivit pro proces uživatelské objednávky

Celý tento proces se odráží od požadavků, které byly specifikovány v předchozí kapitole 5.2.2, a bude se skládat z pěti dílčích kroků – **zobrazení obsahu košíku**,

kde bude možnost využít slevový kupón, **vyplnění údajů o zákazníkovi** - v případě, že je přihlášen, se tyto údaje do objednávky doplní automaticky a tento krok se přeskočí na **volbu metody dopravy a platby, zobrazení souhrnu objednávky** a při následném potvrzení objednávky se zobrazí **děkovná stránka**, kde celý tento proces jednoho nákupu končí. První čtyři kroky se skládají primárně ze čtyř samostatných formulářů, kdy se automaticky přechází na následující při potvrzení toho předchozího. Poslední krok – děkovná stránka je jen poděkování na závěr. Při potvrzení objednávky se uživateli odesílá informativní e-mail s výzvou k zaplacení a zároveň se jeden e-mail odesílá oprávněné osobě s informací o nové objednávce. Do souhrnu objednávky se zákazník bude moct dostat jedině v případě, že nemá prázdný košík, vyplnil zákaznické informace a zvolil si způsob dopravy a platby. Obdobně to bude i s děkovnou stránkou, na kterou se bude moct dostat jen po potvrzení objednávky.

Při volbě dopravy a platby se navíc bude kontrolovat, jakou celkovou částku zákazník hodlá utratit, neboli jaká je suma košíku. Podle této sumy se následně uživateli zobrazí příslušné volby dopravy a platby, což má za následek vyřešení problému ceny daných metod podle určité utracené částky. Například si provozovatel internetového obchodu bude přát, že od utracené částky 1 000 Kč budou tyto metody zdarma. V administraci v příslušném nastavené pro internetový obchod se ještě předtím nastaví právě tato zlomová částka, od které se dané metody zobrazí a pomocí databázového dotazu se tyto hodnoty automaticky vyfiltrují pro daný výpis.

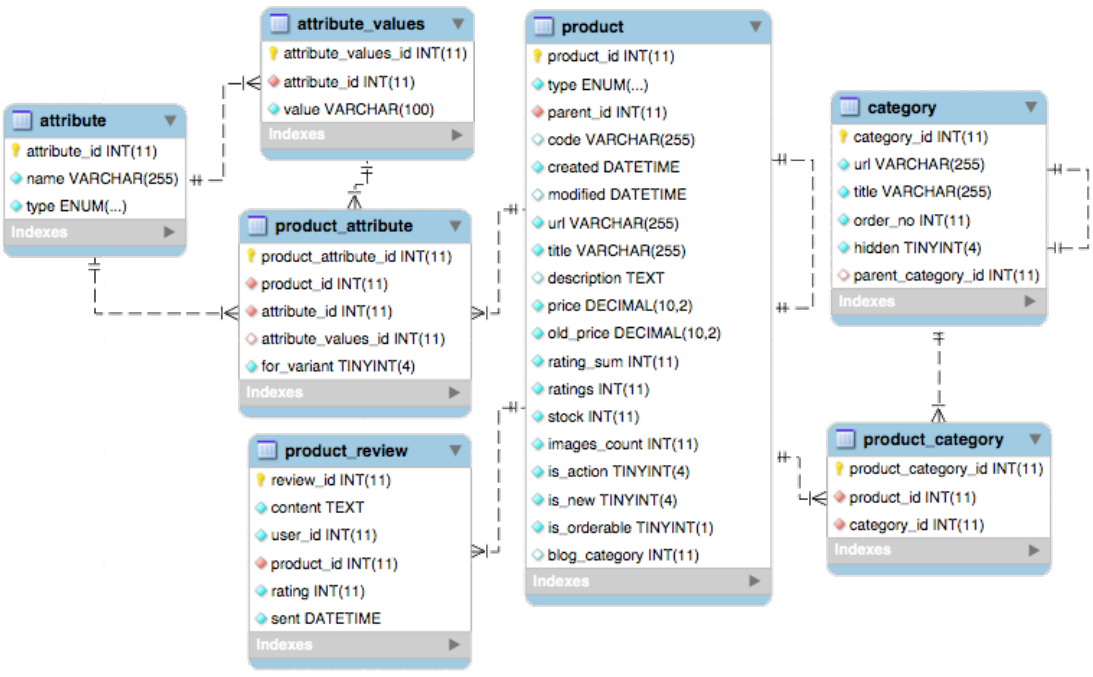
7.2 Návrh databáze

Pro návrh datové základny se využívají modely patřící do ER modelů, jež slouží přímo pro modelování dat. Základními elementy datového modelu na konceptuální úrovni jsou entita (*Entity*) a vztah (*Relationship*). Příkladem entit může být konkrétní prvek reálného světa. Vztah je nějaké spojení mezi těmito entitami. U vztahů lze rozlišovat jejich kardinalitu a volitelnost. Kardinalita zkoumá, zda vztah jedné entity k výskytu jiné entity je výjimečný (exkluzivní), a tedy může nastat pouze jednou, nebo zda může nastat vícekrát. U volitelnosti vztahu se sleduje, zda vztah při výskytu entity vždy nastane a je tedy povinný, nebo zda nastat nemusí a je tedy volitelný (Bruckner, 2012).

Již bylo stanoveno, že bude využita MySQL relační databáze a z tohoto důvodu bude také pro návrh využit speciální nástroj **MySQL Workbench**, který se využívá pro správu, potažmo modelování tohoto typu databáze a byl vytvořen přímo společnosti MySQL. Výstupem tohoto modelování je potom EER diagram, v originálním znění *Enhanced Entity-Relationship model*, který byl navržen pro tvorbu „*přesnějších*“ databázových schémat a lépe odráží datové vlastnosti nebo omezení. V porovnáním s klasickým ER modelem obsahuje všechny jeho modelovací pojetí, avšak navíc obsahuje například vztahy typu *specializace* a *generalizace* (Bruckner, 2012). Vzhledem k funkcionalitě navrhovaného systému bude celkový návrh rozložen do menších částí, které budou složit pro popis dílčích částí. Celkový

návrh by zde nebylo možné jednoduše popsat, proto bude zaměřeno jen na vybrané stěžejní části, které byly navrhovány v předchozích podkapitolách.

První z těchto částí je **návrh datové základny pro produkty**. Z požadavků společnosti a samotné funkcionality platformy pro internetový obchod vyplývá, že produkty budou existovat jednoduché a variabilní. Vše je zachyceno na následujícím návrhu. Návrh obsahuje tabulky *produkt*, *kategorie*, *hodnocení produktu*, *atributy* a *hodnoty atributů*. Kvůli přednosti byly některé vazby, které vedly na tabulky mimo tento návrh, odstraněny, ale v těchto tabulkách zůstaly odkazující indexy.



Obr. 12 EER diagram pro návrh produktů

Z vazeb, které zde zůstaly, jsou patrné vztahy typu **1:1**, kdy například produkt může mít svého rodiče, nebo naopak svého potomka. To stejné platí i pro kategorie, které budou tvořit ukázkovou stromovou strukturu v případě složitějších zanoření. Ze znázorněných vazeb je dále patrné, že jeden produkt může mít více atributů s více hodnotami a atribut může být přiřazen k více produktům. Z toho vyplývá, že je potřeba takzvaná **vazební tabulka**, které pomáhá řešit tento typ vztahu, jenž je označený **M:N**. Tento vztah lze potom rozložit prostřednictvím této tabulky na jednodušší – **1:N,** kdy prvek z jedné tabulky může být ve vazební tabulce obsažen vícekrát v různých kombinacích s prvkem z jiné tabulky.

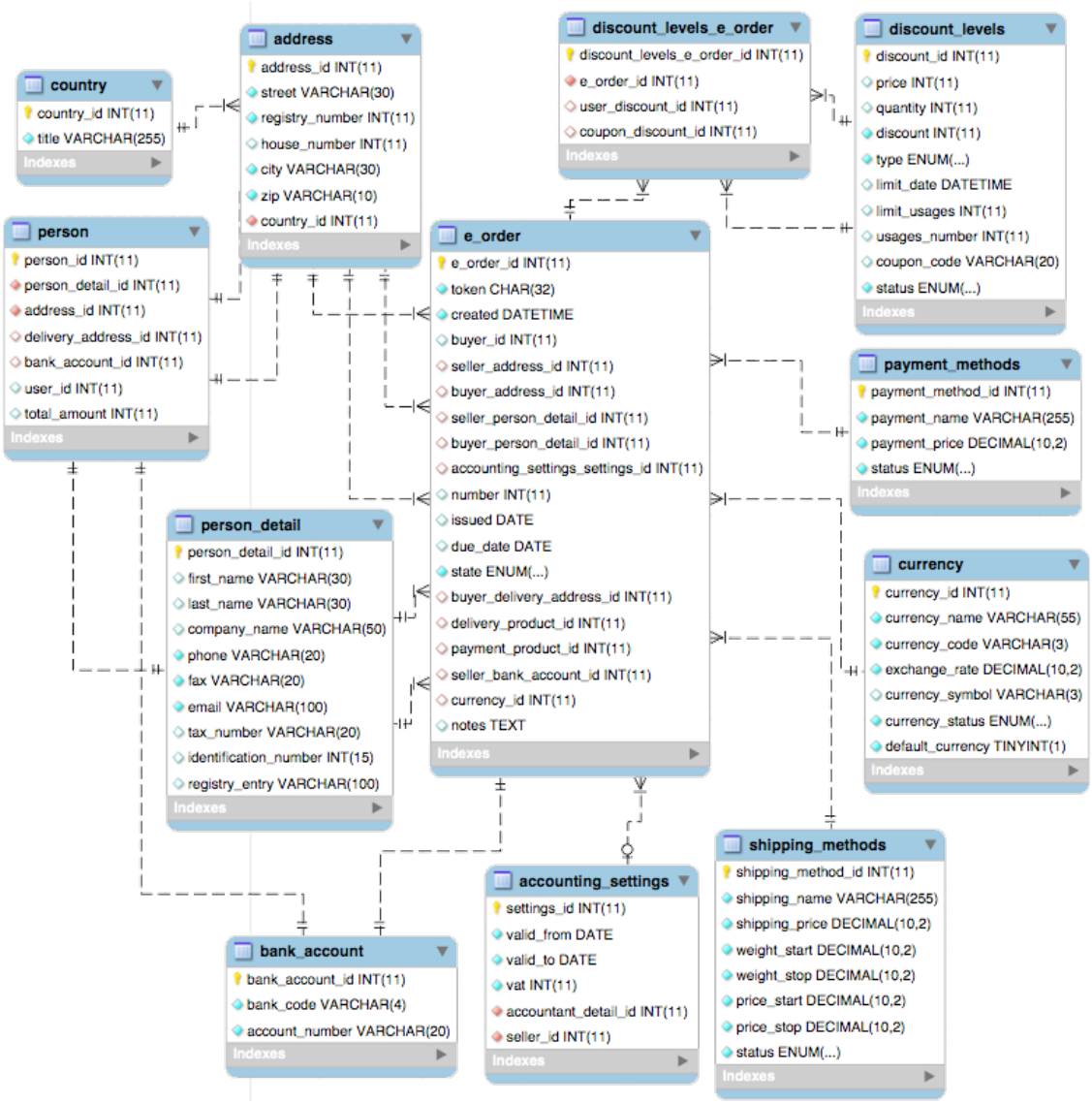
Tento typ tabulek se v návrhu vyskytoval velmi často. Obdobná tabulka je například potřeba pro přiřazení produktů k objednávce, jelikož objednávka může zahrnovat více produktů, ale navíc tyto produkty mohou patřit do více objednávek. Stejný vztah platí i pro produkty a kategorie – produkt může být ve více kategoriích a tyto kategorie mohou mít více produktů. Tyto vazební tabulky potom obsahují jen indexy s přímou vazbou na záznamy do daných tabulek, nebo mohou obsaho-

vat i primární klíč pro případ, že se k tomuto danému záznamu připojí další atributy, jak je možné vidět ve vazební tabulce pro produkty a atributy. Zde se přiřadí atribut a varianta pro produkt, navíc se však uchovává i hodnota, jestli je daný produkt ten původní (variabilní produkt), nebo je již varianta tohoto produktu s danou vlastností.

Funkcionalita je potom taková, že existuje možnost vytvořit jednoduchý produkt, atributy a pro daný atribut i dílčí hodnoty. Tyto atributy se přiřadí produktu, který se tím pádem stává variabilní, a zvolí se, které hodnoty atributu se budou aplikovat pro tento produkt. V pozadí se potom tento produkt zkopíruje a vytváří se podle těchto hodnot atributů ve více variantách. Tyto kopie produktu se potom mohou od sebe lišit v některých vlastnostech, jakými je např. cena. Ve veřejné části se potom vypisují produkty a podle typu (jednoduchý, variabilní) se v detailu produktu dále hledají tyto kopie, které si uživatel může skrze volbu, realizovanou formulářem, zobrazit a přidat do košíku.

Další, poměrně rozsáhlejší návrh se týká **objednávky** včetně tabulek pro ukládání údajů k osobám a všech ostatních souvisejících údajů. Konkrétní tabulka pro produkty byla k vidění v předchozím diagramu a s objednávkou je spojena skrze indexy v propojovací tabulce, která zde pro zjednodušení není zachycena. Zde by bylo vhodné objasnit předchozí poznatek o provázanosti dat, která se při dodatečné modifikaci nesmí projevit do jiných uchovávaných údajů. Velmi často nastane situace, že zákaznické údaje a údaje měnových kurzů budou modifikovány, způsoby platby a dopravy mohou být smazány, ale žádná z těchto modifikací se nesmí projevit do faktur, které budou vycházet právě z údajů objednávek.

Řešení tohoto problému vyplývá ze skutečnosti, že objednávky si musí všechny potřebné údaje uchovávat skrze reference do příslušných tabulek a v případě modifikace některých z těchto údajů musí být ošetřena jejich neměnnost. Této stálosti dat se dosáhne tak, že v pozadí se místo modifikace konkrétních dat vytvářejí data nová, jak již bylo naznačeno v kapitole 6. Například z následujícího diagramu vyplývá, že tabulka osoby (*person*) je jen propojovací tabulkou, která spojuje údaje pro adresu, bankovní účet, detail osoby, potažmo právnické osoby a všech další údajů do jednoho celku. V případě modifikace detailních informací se vytváří v příslušené tabulce nový záznam, starý se ponechává neaktivní a nový záznam se propojí v příslušné tabulce pro osoby. Požadovaná změna se tedy skutečně vykoná, ale staré údaje, na které se odkazuje v objednávce, zůstávají stále v databázi uloženy. Takto si objednávka uchovává údaje pro dopravu, platbu, měnové kurzy, údaje prodejce a kupujícího, ale co je hlavní, údaje pro účetnictví atp. Pokud by uživatel požadoval smazání některé z položky, na kterou je odkazováno skrze cizí klíč, položka se neodstraní, ale nýbrž se jen skryje tím způsobem, že se jí nastaví hodnota statusu na skrytý.



Obr. 13 EER diagram pro objednávku

Nastavení účetnictví odpovídá tabulka „*accounting_settings*“, která uchovává vymezenou platnost, velikost DPH, odkaz na prodejce a také odkaz na jeho detail. Detail z toho důvodu, že pokud se změní údaje pro prodejce, tyto změny se sice projeví do nově vzniklých objednávek, ale nesmí dojít ke změně globálního nastavení velikosti DPH. Tento údaj je vymezen svou platností a toto pravidlo nesmí být porušeno. V údajích prodejce může dojít ke změně adresy, případně názvu, tyto změny se neprojeví do starých faktur, jelikož si každá uchovává potřebné informace, ale DPH být změněno nemůže.

Detail účetního nastavení obsahuje mimo jiné i položku „*tax number*,“ která odpovídá daňovému identifikačnímu číslu prodejce (účetního). Účetní nastavení bude globální pro celý systém a vždy musí byt nastaveno a to i v případě, že provozovatel není plátce DPH. V tomto případě nastaví hodnotu na 0 a nevyplňuje po-

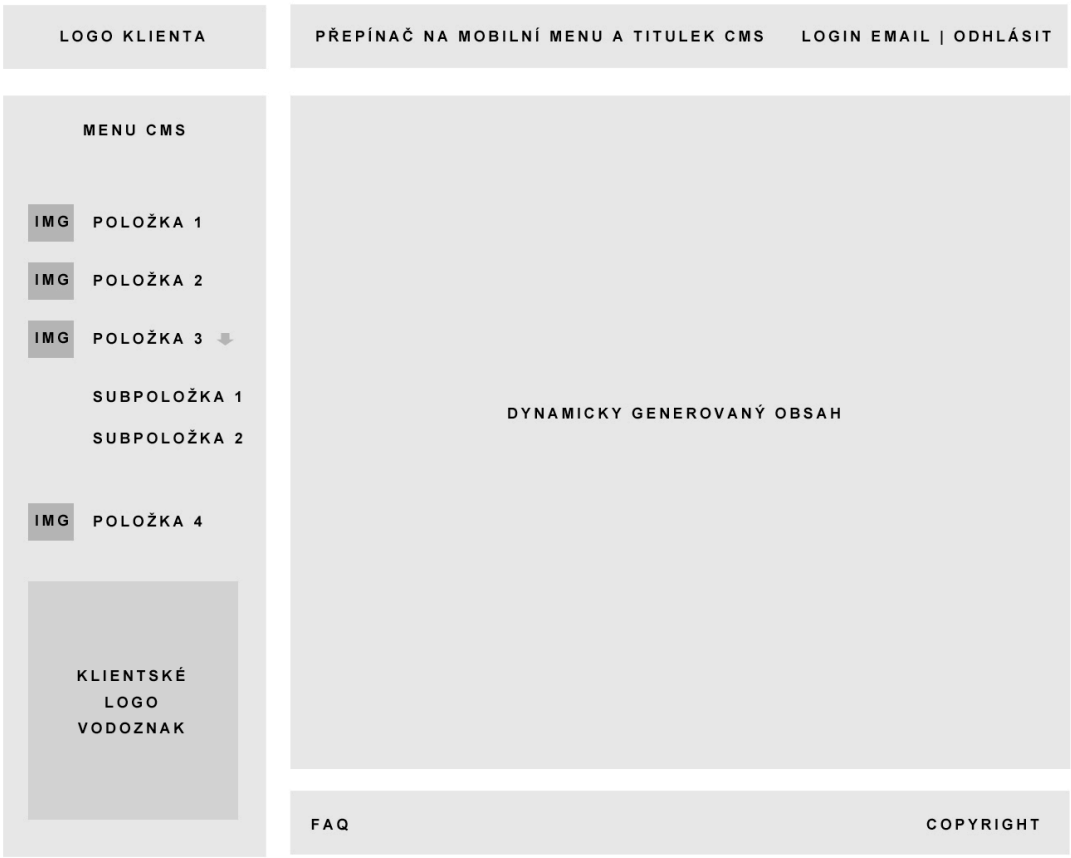
ložku pro DIČ (*tax number*). Před každou budoucí prací s DPH se vždy budou kontrolovat tyto dva údaje, jinak se s DPH počítat nebude. Ceny produktů a služeb v systému potom budou podléhat tomu účetnímu nastavení, objednávky budou cenu přepočítávat vždy podle svého nastavení, jehož hodnotu si vždy uchovávají.

7.3 Návrh vzhledu administrační sekce

Již bylo zmíněno, že veřejně přístupná část se bude projekt od projektu měnit, takže návrh vzhledu, potažmo základní funkcionality bude proveden pro de facto neměnnou část systému – pro administrační, privátní sekci, jež obsahuje všechny moduly až na jednu výjimku, kterou tvoří právě zmíněný frontendový modul. Podle těchto modulů dojde ke sjednocení položek menu, které bude obsahovat **Nástěnku** sloužící pro základní přehled, globální **Natavení** a **Správu uživatelů**. Tyto tři položky vzniknou rozdělením *Core modulu* na více částí, jelikož se předpokládá, že tyto položky zde budou obsaženy vždy. Dále zde bude položka pro správu obsahu – **CMS** (vycházející ze zkratky *Content Management System*), **E-shop** a na závěr **Správa médií**.

Pro návrh základního rozložení prvků na stránce bude využit návrh prostřednictvím **wireframu**, neboli **drátěného modelu** a jedná se o poměrně důležitý krok při přípravě obsahu webové stránky. Jedná se o skicu prezentace obsahu (např. podoba webové stránky), neboli návrh, který definuje funkci a obsah stránky. Wireframe obvykle připravuje UX specialista, jinými slovy odborník na uživatelský zážitek a použitelnost webu. Prostřednictvím toho návrhu je rozhodnuto o budoucím rozložení jednotlivých prvků tak, aby webová stránka efektivně plnila svůj cíl. K tomu patří nejen rozvržení jednotlivých prvků na webu, ale také vhodné rozložení textů, jejich množství a účel (Sálová, et al, 2015).

Protože se jedná o administrační část systému, není potřeba experimentovat se vzhledem, se snahou oslovit co nejširší spektrum uživatelů, ale naopak by zde měl být kladen důraz na jednoduchost a měly by být dodrženy jisté standardy, využívající se při tvorbě vzhledu systému. Zejména pak, že největší část prostoru by měla být věnována právě pro práci s daty, menu by mělo být jednoduché, většinou bývá umístěno na levé straně, a hlavička s patička nemusejí být příliš výrazné, ale naopak by měly obsahovat jen nutné údaje. Následující obrázek odpovídá tomuto rozvržení prostřednictvím drátěného modelu.



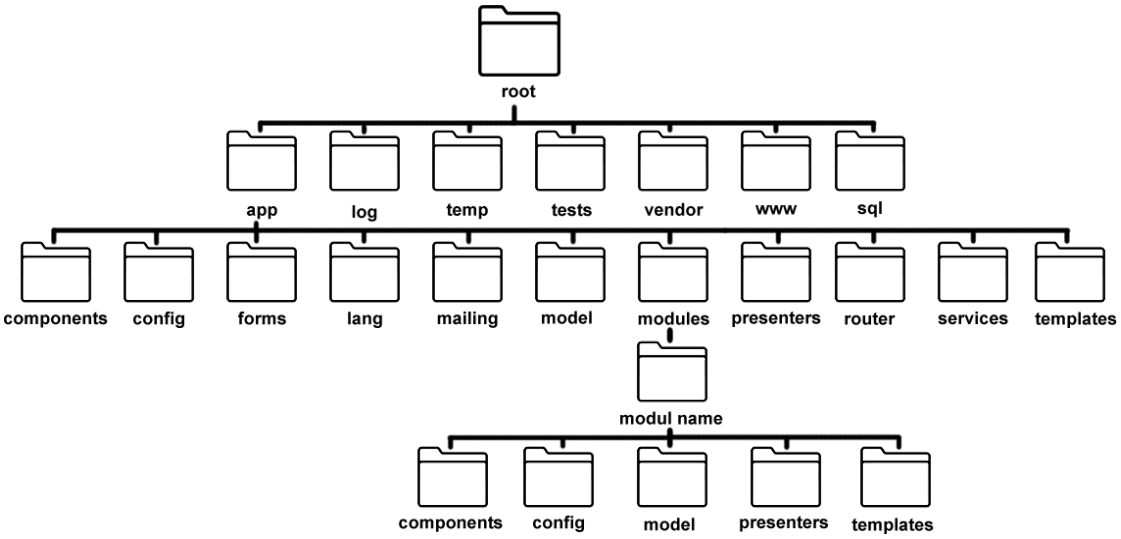
Obr. 14 Drátěný model pro administrační sekci

Vzhled se bude měnit co nejméně, změnami mohou projít barvy administrace a obě klientské loga, která jsou umístěna vlevo nahoře a pod menu vlevo dole, kde je umístěn vodoznak klientského loga. Menu je umístěno přesně tak, jak bylo popsáno vlevo a je fixováno, takže nebude mizet při posunu obrazovky dolů. Tento posun se tedy bude týkat jen pravé části od menu, tedy té, která je určena pro obsah, což je dáno hlavně důrazem na použitelnost. Vpravo dole je umístěn copyright s logem společnosti IDEATECH, vlevo dole v patičce bude odkaz na časté dotazy, kde společnost umístí případně i materiály s videem pro nastavení a práci s administrací. Hlavička obsahové části bude obsahovat ikonu pro přepnutí menu do ještě tenčí lišty, místo názvů položek menu zůstanou jen ikonky a obsahová část se tím pádem stává ještě větší. Dále zde bude hlavní titulek pro jednotlivé stránky, vpravo pak bude informace o právě přihlášeném uživateli s odkazem na jeho detailní informace a tlačítko pro odhlášení.

8 Implementace a zavedení rozhraní

Z předchozích kapitol, ve kterých byly rozebírány specifikace a následně návrh funkcionality systému, lze vydedukovat, že nově navrhovaný systém by měl být založen na co nejvíce samostatných a nezávislých modulech, což bylo patrné už v době specifikace požadavků a následný návrh to jen potvrdil. Výchozí adresářová struktura Nette frameworku tedy již není dostačující, avšak i s touto možností se již počítá a o případnou modifikaci a rozdělení souborů do vlastní struktury se postará takzvaný *autoloader*, který při dodržení jmenných prostorů tříd zabezpečí správné nalezení potřebných souborů.

Výsledná adresářová struktura je zachycena na následujícím diagramu, kde rozložení aplikace do modulů je vyjádřeno jedním adresářem pojmenovaným „*modul name*“, podstatné je však obsah tohoto adresáře, který odpovídá rozložením každému z navrhovaných modulů. Tímto rozložením je zabezpečena dostatečná nezávislost modulů, jelikož všechny jejich potřebné soubory jsou obsaženy uvnitř jejich výchozího adresáře. Vyskytuje se však i situace, že se volají prvky jiného modulu, k čemuž velmi běžně dochází ve frontendovém modulu, jenž se skládá výhradně z prvků jiných modulů. Třídy, které jsou společné pro více modulů, jsou obsaženy v nadřazeném adresáři pro moduly, tedy v adresáři pojmenovaném *app*. Propojení se zbytkem aplikace je také zajištěno skrze konfigurační soubory, přičemž každý z modulů má vlastní a všechny jsou spojeny do jednoho hlavního nacházejícího se v adresáři nazvaném *app*.



Obr. 15 Výsledné rozvržení adresářové struktury

Tyto konfigurační soubory slouží pro výchozí nastavení, které má na starost třída *Configurator* spouštěná při počáteční inicializaci. Výchozím nastavením může být například nastavení produkčního a vývojového režimu, nastavení session, databáze atp. Mimo jiné se zde umísťují definice vlastních služeb, které jsou velmi často

využívány, čímž se kód stává přehlednější a je tím zajištěna znovupoužitelnost opakujícího se kódu. Jako vlastní služby jsou zde konfigurovány mimo jiné komponenty, jež tvoří základní stavební kámen celého systému.

8.1 Implementace

První ukázkovým příkladem, který plynule navazuje na rozložení adresářové struktury a základnímu popisu konfiguračních souborů, potažmo využití komponent, je právě možnosti tvorby komponent skrze ***autowiring***.

8.1.1 Autowiring a tvorba komponent

Tento proces začíná definicí takzvaných *továrních služeb* ve zmíněných konfiguračních souborech, skrze které jsou tyto komponenty vytvářeny jako instance tříd daných komponent. Výhodou takto vytvářených komponent je mimo jiné také možnost jejich opakovaného vytváření skrze Nette nástroj *Multiplier*. Následné redukce množství kódu je dosaženo prostřednictvím služby pro tovární třídy, která je postavena na funkcionalitě zvané *Interface* a *DI kontejneru*, což je ve své podstatě jednoduše řečeno delegování zodpovědnosti o dodávání potřebných závislostí do vyšších vrstev, tedy tam, kde se potom třídy opravdu vytvářejí. Většinou se jedná o systémový kontejner, který je na počátku tvorby téměř všech objektů (Dobeš, 2014). Následující ukázka kódu odpovídá takto navržené komponentě.

```
1.  class PriceCounterControl extends Control {
2.
3.    const TEMPLATE_PATH = '/../templates/components/PriceCounter/';
4.
5.    private $currencyManager;
6.    private $currency;
7.
8.    public function __construct(CurrencyManager $currencyManager) {
9.      parent::__construct();
10.     $this->currencyManager = $currencyManager;
11.     $this->currency = $this->currencyManager->getActualCurrency();
12.   }
13.   // Class content = render function and others functions ...
14. }
15.
16. interface IPriceCounterControlFactory {
17.   /** @return PriceCounterControl */
18.   public function create();
19. }
```

V příslušném presenteru se poté tato komponenta jednoduše vytvoří za pomoci vytvořeného rozhraní, které je zachyceno v předchozím ukázkovém kódu. Vytvoření pak vypadá následovně:

```
1.  /** @return PriceCounterControl */
2.  protected function createComponentPrice() {
3.    return $this->priceCounterControlFactory->create();
4.  }
```

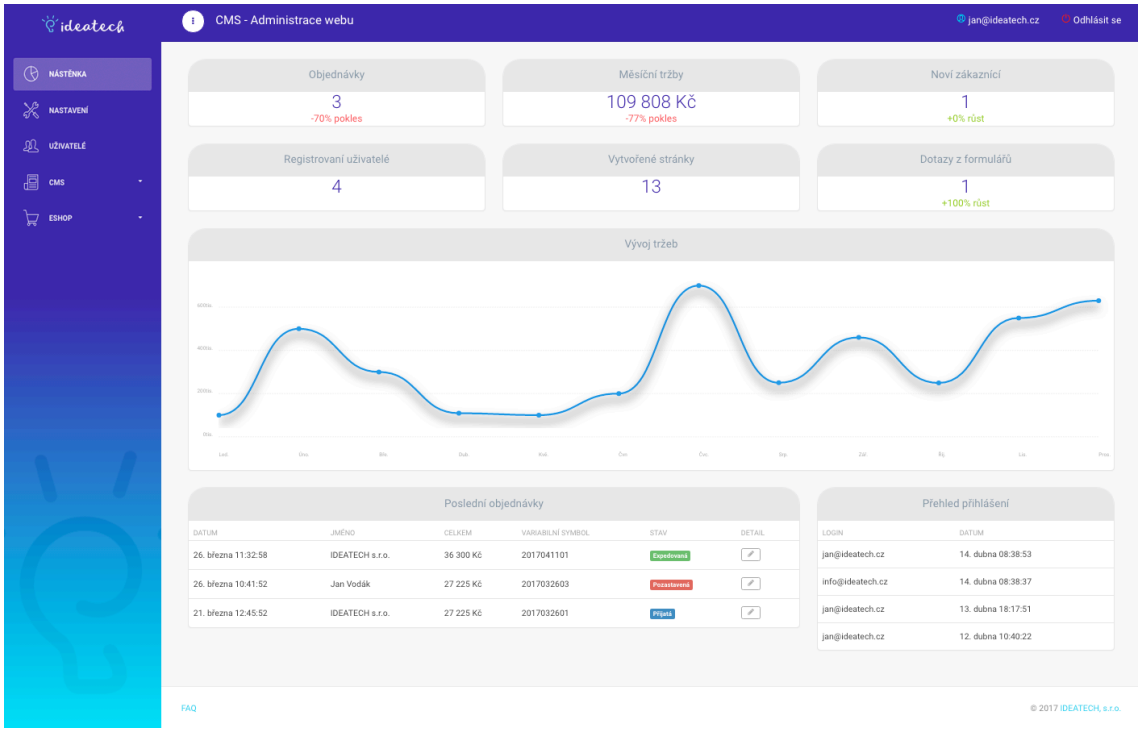
Ukázkovou komponentou, vytvořenou tímto způsobem, je například komponenta pro výpočet výsledné ceny, které se volá pokaždé, když se v systému vypisuje jakákoliv položka týkající se ceny. Tato komponenta je zachycena právě v předchozích ukázkách kódu na předchozí straně. Ceny produktů jsou uloženy v databázi, ale výsledná částka se dopočítává podle položek, jež obsahuje. Výsledná cena se může skládat z více faktorů, které mají na výslednou hodnotu velký vliv - jedná se o DPH, slevy a měnový kurz. Dále do tohoto výpočtu vstupuje činitel, že pokud se budou dopočítávat slevy z objednávky, tak tyto slevy by se neměly týkat cen za dopravu nebo platby. V poslední řadě se zde vyskytuje možnost zaokrouhlení a nastavení počtu desetinných míst při výpisu výsledné částky.

Pro ukázku zde bude uvedena hlavní funkce, sloužící pro výpočet výsledné částky, což je také výstupem této funkce. Funkce spadá pod příslušný model a je volána ve zmíněné komponentě pro výpočet ceny. Jako parametry se této funkci předávají hodnoty ceny, měnový kurz, účetní nastavení, identifikační číslo uživatele a přidaná částka. Nejdříve se testuje, zda je prodejce plátce DPH a pokud ano, částka se zvýší o hodnotu DPH. Totéž se provede i pro dodatečnou částku, které odpovídá například ceně za dopravu a platbu. Pokud je zadáno uživatelovo identifikační číslo, zjistí se jeho celková hodnota útraty a prozkoumá se, jestli pro tuto hodnotu není nastavena nějaká sleva. Tento druh slevy slouží jako věrnostní klub, kdy od určité částky se ceny snižují o zadanou procentuální hodnotu. Kupónové slevy se řeší již při výpočtu celkové částky pro objednávku, takže v tomto bodě je tato sleva již započtena v proměnné pro hodnotu ceny. Na závěr se přičte k ceně přidána částka, které se netýkaly slevy z věrnostního klubu, a provede se přepočet podle měnového kurzu. Vše je zachyceno na následující ukázce kódu.

```
1.  public function calculatePrice($price, $cur, $settings, $uid, $adPrice) {
2.    $vatPayer = isset($settings[self::TAX_NUMBER]);
3.
4.    if ($settings && $vatPayer) {
5.      $price = $price * (1 + ($settings[self::VAT] / 100));
6.      if ($adPrice) {
7.        $vat = 1 + ($settings[self::VAT] / 100);
8.        $adPrice = $adPrice * $vat;
9.      }
10.   }
11.
12.   if ($uid) {
13.     $userAmount = $this->personManager->getUserTotalAmount($uid);
14.     $discount = $this->getActualUserDiscountLevel($userAmount);
15.     $price = $discount ? $price * (1 - ($discount / 100)) : $price;
16.   }
17.
18.   if ($adPrice) {
19.     $price += $adPrice;
20.   }
21.
22.   return $price / $cur[CurrencyManager::EXCHANGE_RATE];
23. }
```

8.1.2 Dashboard

Nyní k ukázce systému, který vychází z předchozího wireframe návrhu. Po úspěšném přihlášení do administrace a kontrole oprávnění následuje přesměrování na uvítací stranu, jež slouží jako nástěnka (anglicky *dashboard)*. Tato nástěnka má za úkol integrovat informace z více složek do jednoho zobrazení a využívá se jako přehled klíčových informací a upozornění. Tato uvítací stránka je zachycena na následujícím obrázku:



Obr. 16 Domovská stránka administrace – dashboard

Na obrázku lze vidět výchozí rozvržení položek v administraci. To se skládá ze základních prvků jako jsou menu, hlavička, patička a obsahová část. V obsahové části jsou v boxech umístěny základní číselné statistiky, jakými jsou počet objednávek v daném měsíci, celkové měsíční tržby a počet dotazů z kontaktních formulářů. Tyto položky zároveň obsahují procentuální porovnání oproti předchozímu měsíci. V dolní části je rychlý přehled posledních objednávek s odkazy na jednotlivé detaily a také je zde znázorněn přehled posledních uživatelských přihlášení do administrace se jménem uživatele a časem jeho přihlášení. Nejvíce dominantní část domovské stránky však tvoří grafické vyjádření měsíčních tržeb za poslední rok. Uživatel si tedy dokáže jednoduše představit, jaké tržby momentálně dosahuje v porovnání s předchozími měsíci i rok pozpátku. V případě že bude modul pro internetový obchod neaktivní, příslušné položky nebudou vůbec použity a budou skryty.

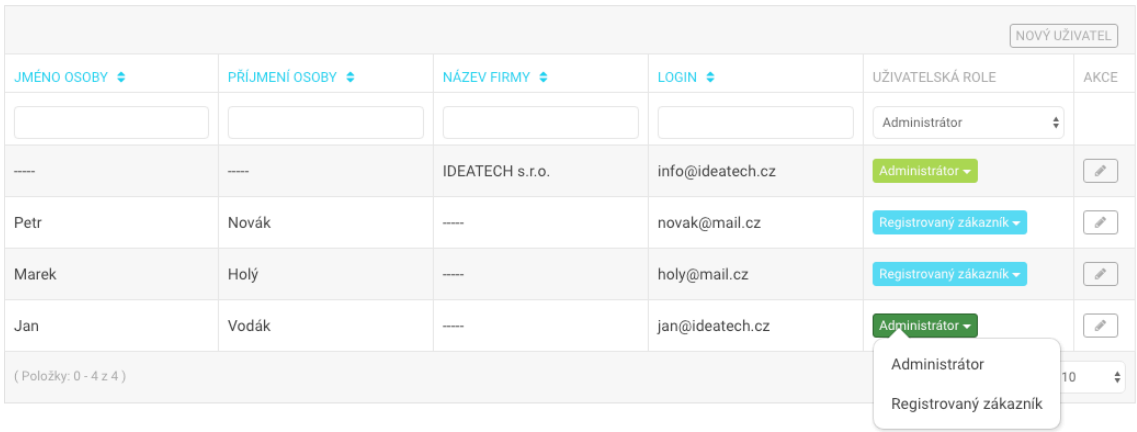
V podkapitole 6.2 o funkčních a nefunkčních požadavcích bylo vysvětleno, že pro grafické vyjadřování informací je využito JavaScriptové knihovny *Chartist.js*. Toto využití spočívá v přidání potřebných JavaScriptových souborů do projektu a v následné předáním dat pro vykreslení. Mimo jiné jednotlivé vytvoření grafu, získání dat pro graf a jeho vypsání do šablony je realizováno také skrze komponentu, o kterých bylo psáno v předchozí podkapitole. Předání příslušných dat mezi PHP soubory a JavaScriptem je zajištěno skrze příslušný typ Nette funkce - ***handle***, které jsou určeny pro zpracování signálů, neboli pro *subrequesty*. Hlavní využití těchto funkcí spočívá ve zpracování ajaxových požadavků, kterých je mimo jiné využíváno například u formulářů nebo pro zmíněné předání dat do JavaScriptových souborů. V této funkci je potřeba si získat příslušná data, která jsou poté odeslána v Json formátu a následně získána v JavaScriptovém souboru. Pro ukázku je zde uvedena příslušná funkce:

```
1.  public function handleGetMonthData() {
2.    $months = array();
3.    $sales = array();
4.    $monthsNames = $this->formatHelperService->getChartMonths();
5.
6.    for ($i = 0; $i < 12; $i++) {
7.      $monthNum = date('n', strtotime("-$i month"));
8.
9.      $monthSales = $this->getTotalMonthSales($monthNum);
10.
11.     if($monthSales > 0){
12.       array_push($months, $monthsNames[$monthNum - 1]);
13.       array_push($sales, $this->getTotalMonthSales($monthNum));
14.     }
15.   }
16.
17.   $this->presenter->sendJson([
18.     'labels' => array_reverse($months),
19.     'series' => [
20.       ['name' => 'monthSales', 'data' => array_reverse($sales)],
21.   ]]);
22. }
```

V této funkci je vidět zmíněné získání dat pro jednotlivé měsíce rok dozadu od aktuálního měsíce. Data jsou předána jedině v tom případě, když jsou tržby pro daný měsíc větší než 0, což zabrání zbytečnému vykreslení nulových hodnot. Ověření může být vhodné například pokud nějaká firma začíná využívat toto řešení a nemá historická data pro zobrazení. Jméno měsíce se získává z pomocné služby, ve které jsou definovány mimo jiné zkratky českých měsíců, nebo se zde nachází funkce pro formátování jmen osob a jejich adres do podoby vhodné pro výpis do šablony. Poté následuje v šabloně pro výpis zavolání této funkce skrze speciální makro *plink*, které slouží pro odkazování se na metody presenteru, čímž se data načtou do šablony a v příslušeném skriptu, který slouží pro vytvoření grafu, jsou data získána prostřednictvím jQuery metody *getJson* a následně jsou dosazeny do grafu.

8.1.3 Uživatelé, nastavení oprávnění a aktivnosti modulů

Přehledy údajů uživatelů a jejich případná správa v administraci využívá komponentu ***datagrid***, *o* které bylo psáno v podkapitole 6.2 o funkčních a nefunkčních požadavcích. Tato komponenta usnadňuje práci s daty a umožňuje mimo jiné například vyhledávání, řazení, filtrování, ale také možnost editace nebo vkládání údajů přímo v daném výpise. Ukázkový datagrid pro přehled uživatelů a jejich rolí je zachycen na následujícím obrázku:



Obr. 17 Ukázkový datagrid pro správu uživatelů

Na obrázku je znázorněn výpis uživatelů a základních informací o nich. Tento výpis je realizován skrze tabulku, avšak ta je obohacena právě o zmíněnou interaktivitu. Tabulka obsahuje klasicky hlavičku, kde jsou vypsány názvy sloupců, které lze řadit vzestupně či sestupně, a pod názvy sloupců jsou umístěny vyhledávací pole, případně filtry, přičemž práce s jednotlivými prvky je realizována ajaxově. Pod těmito řádky následuje výpis daných informací. Ukázkový výpis na obrázku obsahuje mimo jiné i informaci o přihlašovacím jméně do systému, které je realizováno skrze unikátní e-mailovou adresu a uživatelskou roli. Tato role se dá jednoduše při kliknutí na ni změnit, jak je patrné z předchozího obrázku, a lze ji vybrat z předem připraveného seznamu všech povolených rolí.

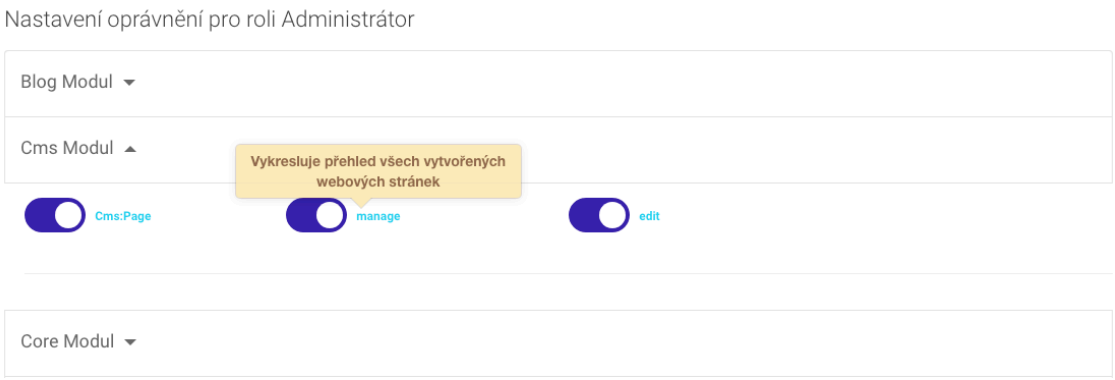
Pro detail nebo vytvoření nové položky jsou zde umístěny příslušná tlačítka, která odkazují na konkrétní podstranu. Možnost tvorby a editace položky přímo ve výpise zde není možné vykonávat, jelikož se jedná o rozsáhlejší proces, ale například v datagridu pro uživatelské role je tato možnost tvorby a editace přímo ve výpise možná, jelikož role obsahují pouze základní informace, jakými jsou její jméno, popisek a následný unikátní textový řetězec, podle kterého lze roli jednoduše identifikovat. Tyto akce následně zpracovávají příslušné *handle* metody, o kterých bylo psáno v předchozí podkapitole.

Autentizaci uživatele zajišťuje speciální třída, která se využívá při uživatelském přihlášení do systému, ale provedení **autorizace** předchází poměrně složitější proces. Jednotlivým rolím musí být umožněno manuální nastavení příslušeného oprávnění pro jednotlivé sekce, případně udělení povolení vykonávat jednot-

livé akce. Z tohoto důvodu odpadá možnost využití nastavení oprávnění skrze konfigurační soubory, kde si lze předem nadefinovat uživatelské role a poté jim nastavovat oprávnění, a nelze využít ani možnost nastavení oprávnění skrze anotace, protože i v tomto případě by administrátor v systému nemohl jednoduše provádět modifikace. Proto je potřeba vytvoření vlastního způsobu, který je založen na anotacích, jak již bylo nastíněno v podkapitole 7.1.1, a navíc je potřeba také zapojit databázi.

Anotace jsou speciální druh komentářů, které dokáží rozšířit programovací jazyk o novou funkcionalitu. Celý mechanismus je tedy založen na vytváření speciálních komentářů, ve kterých bude nastaveno jméno a popis pro jednotlivé zdroje (presentery) a jejich akce, na které uživatel může být přesměrován a které by mohl případně vykonat. Tyto akce se týkají i jednotlivých komponent, jež jsou pro specifické případy vytvářeny. Pro všechny tyto anotace se následně provádí kontrola, zda jsou uloženy v databázi, a v případě, že tam nejsou, provede se jejich uložení. Tato činnost je mimo jiné podmínky v kontrole také založena na testování unikátního složeného klíče v databázi, který má na starost právě doplněk kontroly, zda je položka již v databázi uložena. Tento klíč se skládá ze jména akce a indexu na zdroj, tedy na presenter. Jméno zdroje také navíc obsahuje jméno modulu, což umožňuje hned na první pohled jednoznačně určit, pod který modul tento zdroj patří, a to i když kromě zmíněných hodnot položka pro akci obsahuje i přímo referenci na daný modul. Proces načtení všech anotací zdrojů a akcí, uložení do databáze i se zmíněnou kontrolou, se provádí automaticky vždy při vstupu uživatele do příslušné sekce pro nastavení oprávnění a obstarává ji speciální služba.

Z toho mechanismu vyplývá, že je potřeba uchovávat zvlášť uživatele, role, zdroje a akce pro tyto zdroje. Pro propojení rolí a jejich povolených zdrojů s akcemi je vytvořena v databázi speciální vazební tabulka, která obsahuje tři sloupce s indexy rolí, zdrojů a akcí. Proces manipulace s daty této tabulky je zabezpečen skrze komponentu formuláře. Tento formulář obsahuje všechny akce dostupné v daném systému a uživatel má možnost v něm zatrhnout, které akce budou pro danou roli povoleny. Při zpracování se poté z této tabulky nejprve smažou všechny aktuální záznamy určené pro danou roli a uloží se všechny nově povolené akce, které byly uživatelem povoleny i s odkazy na jejich zdroj a roli. Ukázka tohoto formuláře je znázorněna na následujícím obrázku, kde je nastavení přístupů pro roli Administrátor. Tato role je zároveň nejvyšší zákaznickou rolí a je přiřazena uživateli, kterému bude systém následně prodán. Jedná se o v pořadí druhou nejvyšší roli hned za rolí Super Administrátor, která je naopak přiřazena vývojáři, případně lidem ze společnosti IDEATECH a liší se hlavně v tom, že se nedá editovat. Tato role má permanentně povolen přístup kamkoliv a možnost vykonat všechny akce, zejména se pak jedná o možnost přístupu k logovaným uživatelským akcím, případně možnost určovat aktivnost modulů.



Obr. 18 Formulář pro nastavení oprávnění pro vybranou uživatelskou roli

Na obrázku jsou vidět moduly, které lze po kliknutí na jejich název rozbalit. Jejich obsahem jsou vlevo možnosti nastavení aktivnosti zdroje a vpravo jsou zmíněné akce. Po najetí na každý formulářový prvek se objeví nápověda sloužící pro zvýšení přehlednosti. Prosté názvy zdrojů a akcí většině uživatelů, s výjimkou programátora, nic neřeknou, od toho jsou zde zmíněné nápovědy.

Výsledné sestavení má na starost speciální třída, jež slouží jako takzvaný „*autorizátor*“. Nette framework disponuje již hotovou implementací takovéto třídy, která poskytuje programátorovi lehkou a flexibilní vrstvu pro řízení uživatelských práv a přístupů. Nově navržená třída využívá právě tento předpřipravený mechanismus, jehož práce spočívá v definici rolí a jednotlivých zdrojů. Z tohoto důvodu se tedy uchovávají právě zmíněné informace, jakými jsou zdroje a akce, jelikož mechanismus je vybudován na již existující funkcionalitě Nette frameworku. Tato třída bude využívat návrhový vzor „*factory*“ a bude vracet při svém vytvoření kompletní nastavení uživatelského oprávnění, seznam rolí, zdrojů a povolených akcí pro jednotlivé role. Následující kód zachycuje výslednou metodu, která zabezpečí vrácení kompletního nastavení.

```
1.  public function getPermission() {
2.    $permission = new Permission();
3.    $permission = $this->gerRoles($permission);
4.    $permission = $this->getResources($permission);
5.    $rolesActions = $this->roleManager->getAllResourceActions();
6.
7.    foreach ($rolesActions as $action){
8.      $role = $this->roleManager->getRole($action[self::ROLE_ID]);
9.      $res = $this->roleManager->getResource($action[self::RESOURCE_ID]);
10.     $action = $this->roleManager->getAction($action[self::ACTION_ID]);
11.     $permission->allow($role, $res, $action);
12.   }
13.
14.   $permission->allow(self::SUPER_ADMIN, Permission::ALL, Permission::ALL);
15.
16.   return $permission;
17. }
```

Tato metoda vytváří původní připravenou třídu pro práci s oprávněním a poté získá všechny povolené akce i zdroje a předá je do tohoto nastavení včetně příslušných uživatelských rolí. Na závěr se automaticky nastaví oprávnění pro roli „Super Administrátor“, která není konfigurovatelná z administrace. Po nakonfigurování této tovární třídy do konfiguračního souboru se může přistoupit ke konkrétnímu hlídání oprávnění, jež se bude provádět vždy před jakýmkoliv přesměrováním, nebo spíše před načtením stránky. Toho lze dosáhnout skrze umístění následujícího kódu do startovací metody rodičovského presenteru, od kterého jej dědí všechny ostatní presentery.

```
1.  protected function startup() {
2.    parent::startup();
3.
4.    $moduleSlug = strstr($this->getName(), ':', true);
5.    $resource = $this->getName();
6.    $action = $this->getAction();
7.
8.    if (!$this->getUser()->isAllowed($resource, $action)) {
9.      $logMsg = $this->translator->translate('core.common.permission');
10.     $this->flashMessage($logMsg, MessageType::ERROR);
11.     if ($this->loginPresenter) {
12.       $this->redirect($this->loginPresenter);
13.     }
14.   }else if (!$this->moduleManager->checkModulActivity($moduleSlug)) {
15.     $modulMsg = $this->translator->translate('core.common.noActiveMod');
16.     $this->flashMessage($modulMsg, MessageType::ERROR);
17.     if ($this->loginPresenter) {
18.       $this->redirect($this->loginPresenter);
19.     }
20.   }
21. }
```

V předchozím kódu je zachycena podmínka, zda uživatel má oprávnění pro přístup do požadovaného presenteru a konkrétní akce, ale je zde ji jiný test. Pokud uživatel s danou rolí má požadované oprávnění, ještě se neprovádí požadované přesměrování, ale místo toho se spouští tento další test – zda je modul, kam uživatel žádá o přístup, aktivní, či nikoliv. Tato aktivnost se také nastavuje v administraci skrze *datagrid* a je velmi prostá. Modul má své jméno, unikátní textový identifikátor a aktivnost. Tento unikátní identifikátor je vlastní první část jeho názvu - například *„CoreModul”* bude mít svůj identifikátor *„Core”*. Tento identifikátor je obsažen v názvu zdroje (presenteru), který tento zdroj začleňuje do modulu. Aktivnost modulu se nastavuje skrze tento datagrid, kde obdobně jako se pro uživatele volí role, tak zde se volí ze daných možností – aktivní, nebo neaktivní.

Z názvu akce v presenteru se tedy vezme i identifikátor (název) modulu, najde se příslušný záznam v databázi a provede se test, zda je modul nastaven jako aktivní. Pokud ne, provede se přesměrování na presenter, který slouží pro přihlášení. Uživatel už přihlášený je, jelikož splnil první část testu, takže bude přesměrován na domovskou stránku administrace – tedy na nástěnku. Mimo jiné je v kódu také vidět využití překladače realizovaného pro administraci skrze knihovnu Kdyby. Jednotlivé řetězce jsou umístěny v příslušných souborech a zde jsou jen jednoduše

volány. Přístup do jednotlivých sekcí je nyní kontrolován jednak na základě příslušného uživatelského oprávnění, tak i na základě aktivnosti modulu. To ale není vše, protože je navíc potřeba neaktivní položky v administračním menu skrýt a toto skrytí zamezí zbytečným pokusům o přístup. Administrační menu využívá knihovnu, která byla také zmíněna v podkapitole 6.2 o funkčních a nefunkčních požadavcích. Využitím této knihovny lze potom jednoduše konfigurovat menu prostřednictvím konfiguračním souborů. Tyto soubory jsou později přidány do konfiguračních souborů pro dílčí moduly, které jsou spojeny v jeden výsledný soubor.

Vykreslení menu zajišťuje speciální komponenta, v níž se ověřuje, zda modul pro odkazovanou položku je aktivní. Název modulu se určí ve zmíněných konfiguračních souborech. Využitím této knihovny pro tvorbu menu se dá také ověřovat, jaké role do jednotlivých odkazů budou mít přístup, a v případě, že uživatel tuto roli nemá, položka se automaticky skryje. Obdobně se dá kontrolovat také to, zda je uživatel přihlášen, zda je předán požadovaný parametr pro požadovanou akci nebo je zde možnost vytvořit vlastní metodu, která se o tuto kontrolu postará a z konfigurace je na ni odkázáno. Kontrola aktivnosti modulů se netýká veřejně přístupné části – webové prezentace. Přesněji řečeno se kontroly provádějí před úplně každou akcí, takže i zde, ale všechny akce obstarávají presentery v tomto modulu, takže tato kontrola bude vždy provedena s kladným výsledkem. Je tedy potřeba již před tvorbou veřejně přístupné části vědět dopředu, co zákazník očekává, a v případě budoucí potřeby se požadovaná funkcionalita dodělá a v administraci zaktivuje.

8.1.4 Nastavení měn a nákupní proces na frontendu

Nákupní proces ve veřejně přístupné části se skládá z 5 kroků, kde první krok je zobrazení obsahu košíku, za tímto krokem následuje zadání uživatelských údajů, volba dopravy a platby, rekapitulace objednávky a děkovná stránka. Z návrhu vyplývá, že uživatelský košík je vlastně položka ve stejné databázové tabulce, jako jsou objednávky, jelikož se jedná o stejné informace, které se liší jen stavem. Tento stav určí, zda se jedná o uživatelský košík, případně již dokončený nákup, ze kterého se stává objednávka.

Jak bylo zmíněno v návrhu, jednotlivé kroky při dokončení nákupu zpracovávají formuláře, které postupně doplňují dílčí zákaznické údaje do databáze k příslušnému záznamu a podle těchto informací se následně dá i řídit přístup k těmto jednotlivých kroků objednávky. Pokud má uživatel prázdný košík, nebo nevyplnil své osobní údaje, nebo nezvolil možnosti dopravy a platby, nebude moci přejít k rekapitulaci objednávky a pokud nedokončil objednávku, nezobrazí se mu děkovná stránka. V zobrazení obsahu košíku je formulář pro editaci množství kusů jednotlivých produktů v košíku (objednávce) a pro přidání slevového kupónu do objednávky. Nejdůležitější formulář je však v rekapitulaci, kde zákazník musí souhlasit s obchodními podmínkami a navíc se zde doplní nejvíc informací, jako jsou kompletní údaje prodejce, informace o použité měně a účetním nastavení, nebo se zde generuje variabilní symbol pro spárování platby. Při dokončení objednávky je nastaven příslušný stav, který určí, že je objednávka dokončena. Od této chvíle se

s daným záznamem nebude pracovat jako s košíkem ve frontendové části, ale nýbrž jako s dokončenou objednávkou, které je evidována v systému. Fyzicky se však stále jedná o jeden a tentýž záznam. Stav objednávky je konfigurovatelný a objednávka může přecházet mezi jednotlivými stavy přesně tak, jak bylo navrženo v příslušném diagramu na Obr. 9, kde jsou tyto změny zachyceny skrze stavový diagram.

Jednotlivým krokům nákupního procesu však předchází proces jiný, který je toho všeho de facto součástí. Jedná se o **proces přidání produktu do košíku**. Již bylo vysvětleno, že košík se od objednávky liší pouze stavem, ve kterém se daný záznam nachází, ale ještě je potřeba uvést, kdy a jak se tento záznam vytváří. V tento moment je již jasné, že je využit mechanismus zapojení databáze a problém při párování zákazníka s vytvořeným košíkem dokáží vyřešit cookies, které uchovávají identifikátor příslušného košíku na určenou dobu, jež je například delší než u sessions, a proto je lepší je využít na jejich úkor. Jako identifikátor daného košíku, který se bude uchovávat v cookies, se využije identifikační číslo objednávky (košíku) z databáze. Pro zvýšení bezpečnosti se bude k identifikátoru košíku také ukládat i 48 znaků dlouhý textový řetězec, který se náhodně vygeneruje a také se uloží do databáze. Tato kombinace zajistí, že uživatel nebude schopný jednoduše ovlivňovat objednávky jiných uživatelů, jelikož by bylo velmi obtížné uhádnout náhodně vygenerovaný textový řetězec a v kombinaci těchto dvou hodnot se to jeví jako takřka nemožné.

S cookies se manipule prostřednictvím HTTP požadavků, přicházejících od klienta. Server je dokáže zpracovat a poslat na ně odpověď, která také obsahuje tento typ hodnot. Pro práci s tímto typem dat připravil Nette framework speciální typ objektů, nabízející uživatelům bezpečné API, zapouzdřující HTTP požadavek a odpověď do zmíněných objektů, čímž dokáže ošetřit nedostatky prostých PHP funkcí pro práci s cookies, jako je například ošetření neplatných znaků nebo poskytnutí zabezpečení před krádeží skrze JavaScript (Nette Foundation, 2017e).

Objednávka, přesněji řečeno košík, se vytváří v momentě prvního přidání produktu do košíku, tedy v momentě, pokud si uživatel přeje vložit produkt do košíku a ještě žádný nemá. Avšak ani tento mechanismus nezabrání tomu, že v databázi bude zůstávat množství nevyužitých košíků, což je způsobeno tím, že někteří z návštěvníků nedokončí započatou objednávku. Vytvoření se provede vždy jen jednou a záznam v cookies je platný pro každého uživatele 1 rok. Tedy pokud si nic neobjedná, čímž by dokončil objednávku, v tom případě se vytváří nový záznam při zmíněném prvním vložení produktu. Při přidávání jednotlivých produktů se tedy kontroluje, zda má zákazník svůj košík, a pokud jej nemá, vytvoří se mu nový. Jako nevýhoda se může jevit fakt, že pokud uživatel smaže cache/cookies, nebo se přihlásí z jiného počítače, případně z anonymního okna prohlížeče, nebude mít k dispozici svůj košík, avšak i s tímto nedostatkem se toto řešení jeví jako dostatečné.

Kontrola existence košíku je postavena na zmíněných HTTP požadavcích, jejichž součástí jsou cookies, ve kterých je uloženo identifikační číslo položky

v databázi a identifikační textový řetězec. Vytvoření uživatelského košíku je zachyceno v následujícím ukázkovém kódu.

```
1.  private function createBasket() {
2.    $token = Random::generate(48);
3.
4.    $basketId = $this->database->table(self::TABLE)->insert(array(
5.          self::TOKEN => $token
6.        ))[self::ID];
7.
8.    $this->response->setCookie(
9.        self::NAME, http_build_query(
10.           array(
11.             self::ID => $basketId,
12.             self::TOKEN => $token,
13.           )
14.       ), time() + (self::VALIDITY)
15.   );
16.
17.   return $basketId;
18. }
```

Je zde vidět i dříve popsána práce s HTTP požadavky, kdy součástí odpovědi na tento požadavek je i právě zmíněná položka cookies. HTTP odpověď je obalena do objektu *response*, který slouží pro zmíněnou práci s odpovědí. Položce v odpovědi je nastaveno jméno, a obsah tvoří pole, jež je převedeno na řetězec, ve kterém jsou hodnoty ve tvaru „klíč=hodnota“ a jednotlivé položky jsou odděleny znakem „&“. Dále je také určena platnost těchto dat, která jsou nastavena na již zmíněny 1 rok. Platnost může být libovolná, klidně i kratší, ale z preventivních důvodů je nastavena zvolená délka.

Téměř totožný mechanismu se využívá i při práci s **měnovými kurzy**, jež jsou modifikovatelné z administrace. Mimo jiné se zde nastavuje výchozí měnový kurz. Při dokončení objednávky ve veřejně přístupné části se dosazuje aktuálně využívaný měnový kurz, který si uživatel může zvolit. Tato volba se při zmíněním dokončení objednávky kontroluje a zjišťuje se, zda uživatel využívá výchozí měnový kurz, nebo provedl změnu. Volba měny se zaznamenává taktéž do cookies a zde se uchovává i pro následující uživatelovy návštěvy s případnými nákupy. Přesněji řečeno se testuje cookies, zda obsahuje záznamy pro měnové kurzy. V případě že ano, použije se tento zvolený měnový kurz, v opačném případě se využívá výchozí měnový kurz, který musí být vždy nastaven. Je to z důvodu, že pokud uživatel žádný měnový kurz nevybere, jedna z měn se vybrat musí a bere se právě ta, která je nastavena jako výchozí. Tuto výchozí měnu také využívá celý systém, ceny se podle ní přepočítávají a využívá se jednotka této výchozí měny.

8.2 Testování

Při vývoji jakéhokoliv systému je potřeba klást velký důraz také na správnost a funkčnost navrhovaného řešení a k tomu slouží testování. O to víc jsou testy potřeba v daném případě plánovaného budoucího rozšiřování nebo pro případné

úpravy dle potřeb zákazníka. Samotný vývoj probíhá v několika fázích a proto existuje i více druhů testů. Specifické jsou pak právě při vývoji projektu, fungujícím ve webové prostředí.

Ještě než dojde k nasazení vytvořeného řešení do ostrých provozních podmínek, musí se vše řádně zkontrolovat, k čemuž pomohou právě testy. V jednotlivých fázích vývoje byly během implementace psány kromě dané funkcionality také tyto testy. Pro základní funkce, například pro výpočet celkové sumy objednávky (cena a počet kusů), případně pro výpočet výsledné ceny, které patří do tříd modelů dle MVC architektury, jsou využity základní **UNIT testy**. Pomocí těchto testů se budou ověřovat výstupy daných funkcí pro rozdílné vstupy. Dané funkce pro rozdílné vstupy musí vždy vrátit očekávaný výsledek a velmi důležitým činitelem při tomto procesu je uvědomění si, že testy se nepíšou proto, aby vždy vyšly, nýbrž aby odhalily případné chyby a nedostatky, takže pokud na začátku testování testy nevychází, je tento proces na dobré cestě. Pro tento druh testování je využit **Nette Tester**, který je vytvořen přímo pro potřeby daného frameworku a slouží nejen pro tento typ testů. Mimo jiné se v něm dají testovat jednotlivé presentery, nebo pomocí vytvořených doplňku se dají snadno otestovat také komponenty, databáze, případně základní nastavení prostředí. Testy jsou zautomatizované a spouštějí se přes příkazový řádek, kde se po spuštění všech napsaných testů vypíše výsledek testování – počet vykonaných testů, počet chybných testů, počet úspěšné provedených testů atp. Testuje se pomocí speciálních testovacích, předem připravených tříd *Assert*, které vyhodnocují vstup s očekávaným výsledkem.

Dalším druhem testů, které jsou pro daný systém napsány jsou **akceptační testy,** konkrétně *Acceptance test*. Správně by se mělo jednat o uživatelské testování, které by měl provádět sám zákazník, avšak toto chování se dá pomocí jistých knihoven simulovat a předejít budoucím chybám. Ukázka kódu, který slouží pro simulování uživatelského chování je znázorněn na následující ukázce:

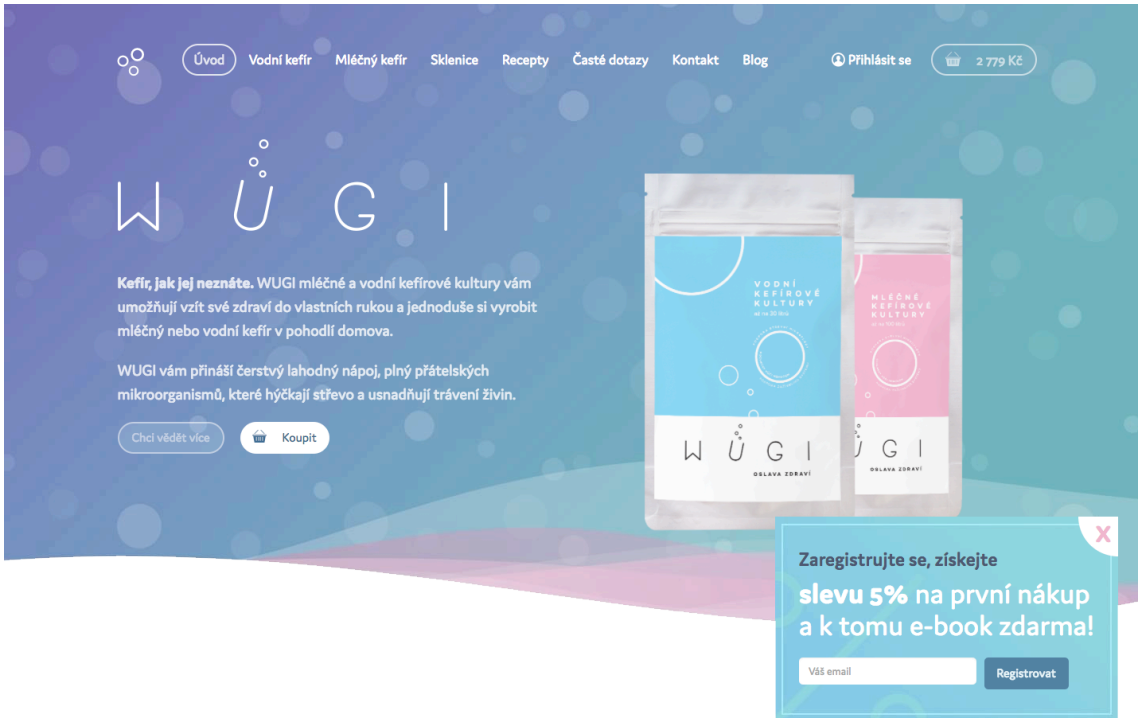
```
1.  public function contactForm(AcceptanceTester $I) {
2.      $I->wantTo('try if contact form is working');
3.      $I->amOnPage('/kontakty');
4.      $I->fillField('#contactForm-name', 'Test');
5.      $I->fillField('#contactForm-email', 'jan@ideatech.cz');
6.      $I->fillField('#contactForm-phone', '12345678');
7.      $I->fillField('#contactForm-message', 'Automaticke testovani.');
8.      $I->click('input[type=submit]');
9.      $I->wait(2);
10.     $I->see('Email byl úspěšně odeslán.', '.alert-success');
11. }
```

V ukázkovém kódu lze vidět testování kontaktního formuláře, který se vyplní a poté se čeká, zda se zobrazí zpráva o úspěchu, která je identifikována pomocí CSS selektoru. Pro simulování chování uživatele se využívá knihovna *Codeception*, která toto chování simuluje ve webovém prohlížeči. Testuje se celková funkčnost – webová prezentace tak i systém, který slouží pro její správu. Testování není omezeno na jednotlivé třídy, nýbrž se testují celé scénáře, jako mohou být registrace, dokončení objednávky, přidání produktu do košíku atd. Pro testování v plnohodnotném

webovém prohlížeči je využit ovladač v daném prohlížeči a server *Selenium*. Při daném testování není ani potřeba mít přístup ke zdrojovému kódu, nýbrž stačí zadat adresu, kde se webové prezentace/aplikace nachází. Testy lze psát objektově za využití třídy *Cest* a výhodou je potom lépe rozčleněný kód. Toho lze dosáhnout, že třídy ve svém jménu obsahují klíčové slovo *Cest a* dědí od příslušné třídy.

8.3 Praktické využití systému

Jedním z cílů této práce je kromě implantace i následné nasazení a otestování systému na reálné problematice. Postupným vývojem se systém rozrůstal o požadovanou funkcionalitu a po dokončení určité části funkcionality byla tato část zavedena do praxe. Momentálně je systém využit v praxi na třech webových stránkách, lépe řečeno u dvou internetových obchodů a jedné zahraniční webové stránce, která pro správu obsahu využívá základní moduly jako Core, CMS, Media a Front modul. Tyto tři webové stránky jsou na sobě plně nezávislé, na každé z nich je nasazen systém samostatně. Každý ze tří klientů požadoval jistá specifika, která odlišují jeden web od druhého, avšak základ zůstal vždy stejný. Nejvíce odlišností se nachází samozřejmě ve veřejně přístupném modulu (frontend), který byl pro každého klienta, co se týká vzhledu a požadované funkcionality, odlišný. Na následujícím obrázku je ukázka hlavičky domovské stránky jedno z e-shopů:



Obr. 19 Ukázka vzhledu frontendového modulu

Na předchozím obrázku je například možnost vidět, že prvkem menu se staly kromě klasických položek také odkazy na produkt. Při tvorbě výsledných stránek je

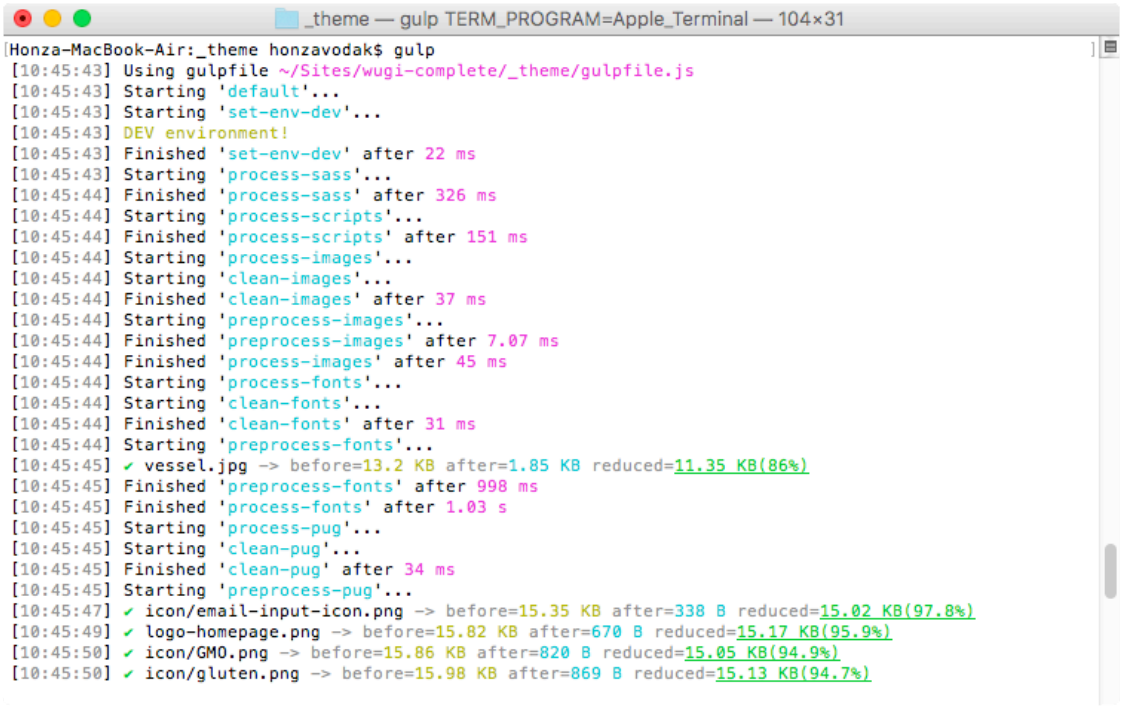
tedy potřeba mít možnost volby odkazu například na daný produkt. O tomto problému bylo pojednáváno v předchozí podkapitole 7.1.2, která se zabývala návrhem CMS modulu. Některé položky menu nejsou, jak již bylo také zmíněno v dané podkapitole, dynamické, nýbrž odkazují na staticky vytvořené stránky. Takto předem vytvořenou, statickou stránkou jsou například stránky pro kontakty, nebo blog, kde jsou vypsány všechny příspěvky.

V případě obou e-shopů se specifika příliš nelišila. Základy zmíněných e-shopů jsou stejné, avšak vzhledem k rozdílnosti nabízených typů produktů se vyskytují jisté odlišnosti. Jeden z e-shopů nabízí k prodeji oblečení, druhý naopak probiotické doplňky stravy. U jednoho bylo potřeba vyřešit varianty produktů pro oblečení a hlavně počty kusů na skladě, doplňky stravy naopak varianty produktů nevyžadovaly a dokonce nebylo potřeba řešit ani stav skladů. Nepotřebná funkcionalita se v kódu jednoduše skryla, avšak pořád zde zůstává, kdyby bylo potřeba v budoucnu některou z těchto funkcionalit dodělat. Tato metoda s neaktivním mechanismem, který je jen skryt, se osvědčila, protože na počátku tvorby internetového obchodu s doplňky stravy například nebyl potřeba mechanismus hodnocení produktů a dokonce ani kompletní mechanismus pro práci s uživateli s možností registrace do systému. Později si klient požadavky rozmyslel a dodatečná funkcionalita byla do projektu jednoduše dodána. V budoucnu by však bylo vhodné se tomu vyvarovat a nabízet řešení jako kompletní celek. Případné skrývání funkcionality by se mělo týkat jen administrační sekce, nikoliv webové prezentace.

V případě čistého CMS, určeného pro zahraničního klienta, došlo k poměrně výrazné modifikaci CMS modulu, jelikož si klient navrhnul uspořádání obsahu veřejně přístupné části do jistých layoutů, které si z administrace žádal editovat. Tyto layouty uspořádávaly obsah do bloků a výsledný vzhled stránek, které tvoří zmíněné bloky, vytváří různorodý obsah. Z administrace lze potom celé tyto bloky přidávat do jednotlivých stránek za využití technologie *Drag and Drop*. Někdy byl na jednom řádku sudý počet bloků, jindy zase lichý. Celý jejich návrh byl poměrně nesourodý, proto bylo potřeba vytvořit specifický mechanismus tvorby takto požadovaného obsahu. Na tomto projektu však pracovalo více firemních programátorů a vzhledem k obtížnosti vymezit činnosti jednotlivých programátorů a kvůli požadavkům klienta zde nemůže být tento projekt více rozebírán.

V každém ze tří případů byl navržen originální vzhled, který byl následně převeden do HTML kódu a poté byl tento kód dosazen do navrhovaného řešení systému, jenž zabezpečuje kompletní funkcionalitu. Na rozdíl od administrační sekce, byly pro převedení návrhu vzhledu z grafické podoby do HTML kódu využity navíc některé z nástrojů, které byly rozebírány v podkapitole 3.4 o vývojářských nástrojích. To je zapříčiněné převážně tím, že administrace využívá jen minimum kódu, určeného pro práci se vzhledem, spíše se vychází z knihoven frameworků, které již obsahují vlastní styly pro definici vzhledu stránky. To však pro veřejnou část nebylo dostačující, jelikož zde je kladen, na rozdíl od administrační sekce, požadavek na upoutání pozornosti návštěvníka se snahou o odlišení se od konkurence, jak vyplývá z Obr. 19, kde je vidět ukázka hlavičky frontendového modulu jednoho z internetových obchodů.

Již bylo zmíněno, že pro kódování frontendové šablony těchto webových stránek je využit šablonovací systém Jade, pro zápis syntaxe CSS stylů byla pro tyto projekty zvolena syntaxe SASS a jako preprocesor, který se postará o převedení syntaxe na klasické CSS a mnohem více, byl zvolen nástroj Gulp. Příkaz gulp je zachycen na následujícím obrázku:



Obr. 20 Proces sestavení souborů skrze nástroj Gulp v konzole

Na obrázku je vidět, že příkaz využívá gulpfile, ve kterém jsou zapsány jednotlivé úkoly, jež tento příkaz vykonává. Příkaz poté do konzole vypisuje jednotlivý průběh daných úkoly – kdy byly započaty a kdy byly dokončeny. Mezi tyto úkoly mimo jiné patří zmíněné zpracování SASS syntaxe nebo JavaScriptových souborů, pročištění fontů a poslední řádky na obrázku patří zmenšení obrázku na velikosti vhodné pro webové prostředí. Pod těmito řádky, které bohužel již na obrázku nejsou zachyceny, je také informace o spuštění režimu pro automatické sledování změn, které poté sleduje změny v jednotlivých souborech, případně celých adresářích a pokud dojde ke změně některých ze souborů, příkaz automaticky provede požadované zpracování kódu a znovu načte stránku, kde se změny ihned projeví. Dále je zde také obsažena informace, na kterém portu je daná stránka k zobrazení.

8.4 Ekonomické zhodnocení projektu a doba návratnosti

Jednou z hrozeb, obsaženou v analýze rizik v podkapitole 6.3, je nenávratnost implementovaného řešení. Dále jedním z dílčích cílů je ekonomické zhodnocení nově navrhovaného řešení, proto je tedy potřeba pokusit se o ekonomické zhodnocení

nově vytvořeného řešení. Pro výpočet zisku lze využít základní ekonomický vzorec, který slouží pro výpočet zisku (Z) z tržeb (T) a nákladů (N). Tento vzorec je zachycen v následující rovnici:

$$\mathbf{Z = T - N} \tag{1}$$

Při určité míře abstrakce a daném předpokladu, že v tomto případě společnost nové řešení nekupuje, ale nýbrž jej vyvíjí, by se dal čistý zisk vyjádřit z tržeb za prodej systému zákazníkům. Náklady by v tomto případě tvořily mzdy vyplacené zaměstnancům, kteří se podíleli na vývoji. Lépe řečeno by se vynásobily časy programátorů strávených při vývoji s jejich hodinovou sazbou. Do tohoto výpočtu by však musel být brán do úvahy také fakt, že systém nebyl prodán jako hotové řešení, ale byl k němu vytvářen originální vzhled frontendového modulu a navíce v každém projektu se vyskytovaly jistá zákaznická specifika, která byla do výsledné částky zahrnuta. Avšak hlavním vylučovacím faktem, který zamezuje tomuto jednoduchému výpočtu zisku je, že systém byl pro společnost vyvíjen v rámci akademických účelů zdarma, a proto zde není žádný náklad společnosti. Jediným, ale podstavným nákladem tedy zůstává čas zhotovitele, který lze vyjádřit jako náklad ušlé příležitosti. Tato hodnota by odpovídala zisku, který by zhotovitel mohl získat, kdyby se věnoval odlišným činnostem než vývoji daného řešení.

Vlivem těchto okolností tedy alespoň lze odhadnout, kolik prostředků bylo společnosti ušetřeno tímto vývojem, a pokusit se stanovit, po kolika případech užití by se tento potenciální náklad vrátil. Odhadovaný čas, strávený při vývoji systému, činí asi 500 hodin čisté práce. V případě, že by tento systém byl vyvíjen v rámci firemního projektu, by hodinový náklad v podobě mzdy PHP programátora na pozici junior mohl činit asi 210 Kč na hodinu. Tato částka zahrnuje mzdu i všechny potřebné částky, které zaměstnavatel musí za zaměstnance skutečně zaplatit. Vynásobením těchto dvou čísel vychází **částka 105 000 Kč**, která odpovídá ušetřeným nákladům společnosti při dané míře abstrakce a odhadů. Pokud by společnost nabízela toto řešení za poplatek **15 000** korun, který by tvořil marži v konečné sumě za poskytnutí systému, musela by společnost výsledné řešení prodat alespoň **7krát**, aby pokryla potenciální náklady. Výsledná cena by se skládala kromě dané marže také z položek, zahrnujících případnou cenu originálního vzhledu (grafický návrh a následné kódování), nasazení kódu na systém, vícepráce a otestování funkčnosti.

Protože byl systém využit již ve třech firemních projektech, lze z toho usoudit, že vývoj tohoto řešení nebyl zbytečný. Navíc má společnost v plánu toto řešení dále využívat. Z hlediska ekonomického hodnocení byl pro společnost projekt velice výhodný a vytvořený systém nezůstane nevyužit, naopak je plánováno jeho budoucí hojné rozšiřování a využívání. V případě, že by společnost tímto řešením nedisponovala, musela by vždy ten stejný základ vytvářet od nuly, případně kopírovat z jiných projektů a mohly by se zde vyskytovat chyby plynoucí z chybného odstranění nepotřebných souborů a zákaznických specifikací. S narůstajícím počtem projektů bude růst i úspora času a financí, která se ještě navýší v případě prodeje systému jako hotového řešení.

9 Závěr

Hlavním cílem práce bylo navrhnutí komplexního webového administračního rozhraní pro společnost IDEATECH s.r.o. a podle tohoto návrhu také následné naimplementování tohoto sytému. Tyto dva hlavní cíle byly úspěšně splněny a systém byl po úspěšné implementaci také otestován v praxi na několika stěžejních případech. Systém je zaměřen na univerzálnost použití napříč různými projekty, což dokazuje jeho užití ve třech odlišných projektech a toto číslo bude dále narůstat, jelikož proces vývoje není u konce a poskytnuté řešení se bude dále rozvíjet a využívat.

Největší přínos nově vytvořeného řešení je sjednocení nejčastěji se vyskytujících zákaznických požadavků, které byly po společnosti doposud vyžadovány. Vychází se ze zkušeností odpovědných pracovníků společnosti, kteří sepsali tyto opakující se požadavky do výchozího zadání. Mnoho z firemních řešení bylo založeno na komerčních, veřejně poskytovaných hotových řešeních, čímž se šetřil čas a prostředky na vývoj. Jiná řešení byla vytvářena přímo na míru podle požadavků zákazníků, avšak v mnoha případech se tyto požadavky opakovaly, a proto bylo potřeba vytvořit ucelené firemní řešení, čímž se navíc může zvýšit prestiž firmy. Jiným z důvodů byl přímo zákaznický požadavek na vlastní řešení, které by nebylo vytvořeno na zmíněných komerčních řešeních. Toto byly hlavní důvody pro vývoj nového firemního řešení a následným využíváním tohoto řešení opět dojde k úspoře prostředků a času při vývoji.

Univerzálnost systému je založena na rozvržení dílčích částí systému do na sobě nezávislých funkčních modulů a tyto moduly lze podle potřeby deaktivovat, čímž se stávají v systému skryté a uživatel k nim nemá přístup. V případě potřeby se tyto moduly dají opět aktivovat. V případě nutnosti rozšíření systému o další funkcionalitu lze velmi jednoduše doplnit systém o moduly nové a tyto moduly také začlenit do funkcionality nastavení aktivnosti modulů. Nezávislost modulů vychází z dobře navržené adresářové struktury, která je založena na výchozí adresářové struktuře Nette frameworku. Každý z modulů obsahuje všechny potřebné soubory pro svoji funkcionalitu, tudíž tvoří komplexní celek.

Na úplném počátku byl proveden průzkum v oblasti technologií pro tvorbu webových administračních rozhraní. Výsledek průzkumu byl shrnut do samostatné kapitoly a získané poznatky byly později využity při implementaci navrhovaného systému. Využitím správných vývojových nástrojů a správnou volbou technologie je zabezpečena možnost požadované rozšiřitelnosti a je poskytnut stabilní základ vyvíjeného řešení pro jeho budoucí použitelnost.

Návrh a následná implementace systému vycházejí z již zmíněného poskytnutého zadání společnosti, pro kterou je systém vyvíjen. Tyto požadavky byly důkladně prostudovány a poté z této studie byly vytvořeny formální a neformální specifikace požadavků. Následně byla provedena vlastní analýza požadavků a v případě jistých nedostatků, které byly zjištěny ze zformulovaných specifik systému, byly navrženy i případné návrhy na zlepšení. Tyto návrhy byly konzultovány s odpovídajícími zástupci společnosti, kteří byli neustále zapojování do všech pro-

cesů od úplného počátku a měli nad těmito procesy dostatečný dohled. V rámci vlastní analýzy byla také provedena analýza rizik spojených s vývojem systému, která byla založena na analýze rizik RIPRAN. Rizika byla úspěšně zhodnocena a byly pro ně nalezeny odpovídající návrhy na opatření, čímž se výsledná hodnota rizika snížila.

Bylo zmíněno, že nově poskytnuté řešení se bude neustále rozvíjet, jelikož tento vývoj není ani zdaleka u konce. V systému existují jisté prostory pro zlepšení, které by se do systému měly doplnit. Jedná se například o mechanismus překladů jednotlivých částí frontendového modulu, kdy stránky, produkty a všechny další potřební prvky by měly mít opět více variant pro různé jazykové mutace, text v šablonách by měl být převeden do databáze a odtud být dále uživatelsky modifikovatelný. Poté, podle zvoleného jazyka ve veřejné části, by se z databáze vybraly data pro daný jazyk a ta byla dosazena do šablon. Nelze využít stejný mechanismus, který byl použit pro překlad administrační sekce, jelikož takto překládaný obsah nelze jednoduše editovat z administrace.

Nejen k jednoduché manipulaci s překlady by navíc mohl přispět i plánovaný přechod z klasické Nette databáze na jeden z ORM (*Object Relational Mapping*) framework, konkrétně na *Doctrine*. Tento přechod byl navržen a následně také i schválen odpovědnými osobami ze společnosti, které si od tohoto přechodu slibují zlepšení funkcionality a zpřehlednění celkové činnosti Modelové vrstvy MVC architektury. Tento přechod se již začal pomalu realizovat s postupným rozšiřováním funkcionality.

Další z možných rozšíření se týká účetního nastavení, protože se může vyskytnout situace, že v některých případech je potřeba nastavit rozdílné sazby DPH pro vybranou skupinu produktů, a proto mechanismus centrálního nastavení pro všechny produkty nemusí být dostačující. Z tohoto důvodu by bylo vhodné stávající mechanismus ponechat a rozšířit jej o novou funkcionalitu, kdy by šlo vytvářet více DPH hodnot, které by poté šly v detailu jednotlivých produktů nastavovat. V případě, že by se DPH u produktu nezvolilo, byla by nastavena výchozí centrální hodnota.

Mnoho firem využívá pro správu účetnictví některý z vybraných softwarů, určených pro danou problematiku, a proto také požadují, aby data z prodeje byla propojena na daný účetní software. Z tohoto důvodu je potřeba doplnit nově vytvořený systém o rozhraní, které by toto propojení dokázalo poskytnout. Toto rozhraní by mělo za úkol naformátovat potřebné data do požadovaného tvaru a tato data následně poslat do účetního softwaru. Navíc pokud bude zákazníkovi nabídnut přechod na nově vytvořené řešení, je potřeba ošetřit při tomto přechodu také zachování stávajícího tvaru URL adres. Zákazník mohl investovat nemalé finanční prostředky do vytvoření stabilní pozice v internetových vyhledávačích a zajisté nebude stát o to, aby při tomto přechodu došlo ke ztrátě. V některých případech se může jednat o velké množství URL adres, jejichž změny by nemuselo být možné provést manuálním zpracováním.

10 Literatura

10.1 Knižní zdroje

BRUCKNER, T. *Tvorba informačních systémů: principy, metodiky, architektury*. Praha: Grada, 2012. Management v informační společnosti. ISBN 9788024741536.

CAMERON, D. *A Software Engineer Learns HTML5, JavaScript and jQuery: A guide to standards-based web applications.* U.S.: Cisdal Publishing, 2013. 256 s.

CHAFFER, J.,SWEDBERG, K. *Mistrovství v jQuery: [kompletní průvodce vývojáře]*. Brno: Computer Press, 2013. Mistrovství. ISBN 9788025141038.

DARIE, C. , BRINZAREA, B., HENDRIX, A. *AJAX and PHP: Building Modern Web Applications 2nd Edition.* Birmingham: Packt Publishing, 2009. 308 s.

DRUSKA, P. *CSS a XHTML: tvorba dokonalých webových stránek krok za krokem*. Praha: Grada, 2006. Průvodce (Grada). ISBN 8024713829.

GAJDA, W. *Git recipes*. 2013. New York, NY: Apress, 2013. Expert's voice in open source. ISBN 9781430261032.

HOPKINS, C. *PHP okamžitě*. Brno: Computer Press, 2014. ISBN 9788025141960.

KRAUSE, J. *Programming web applications with node, express and pug*. Berlín. Apres, 2017. ISBN 9781484225103.

PEHLIVANIAN, A., NGUYEN, D. *JavaScript okamžitě*. Brno: Computer Press, 2014. ISBN 9788025141632.

PÍSEK, S. *HTML: začínáme programovat. 4.*, aktualiz. vyd. Praha: Grada, 2014. Průvodce (Grada). ISBN 9788024750590.

PRETTYMAN, S. *Learn PHP 7: Object Oriented Modular Programming using HTML5, CSS3, JavaScript, XML, JSON, and MySQL.* New York: Apress Media, 2016. 308 s. ISBN 978-1-484217-29-0

ŘEZÁČ, J. *Web ostrý jako břitva: návrh fungujícího webu pro webdesignery a zadavatele projektů*. Jihlava: Baroque Partners, 2014. ISBN 9788087923016.

SÁLOVÁ, A., VESELÁ, Z., ŠUPOLÍKOVÁ, J., JEBAVÁ, L., VIKTORA, J. *Copywriting: pište texty, které prodávají.* Brno: Computer Press, 2015. ISBN 9788025145890.

SIROVICH, J., DARIE, C. *SEO v PHP : programujeme profesionálně.* 1. vyd. Brno: Computer Press, 2008. 380 s. Programmer to programmer. ISBN 978-80-251-2083-5.

VERENS, K. *CMS Design Using PHP and jQuery.* Birmingham: Packt Publishing, 2010. 340 s. ISBN 978-1-849512-52-7.

VERHOEF CH., EVELEENS, J., L. *The Rise and Fall of the Chaos Report Figures*, *IEEE Software*, vol. 27 , s. 30-36, 2010, DOI:10.1109/MS.2009.154

VRÁNA, J. *1001 tipů a triků pro PHP*. Brno: Computer Press, 2012. ISBN 9788025129401.

ZAKAS, N. *The Principles of object-oriented JavaScript.* San Francisco, CA: No Starch Press, 2014. 97 s. ISBN 978-1-59327-540-2.

ŽÁRA, O. *JavaScript: programátorské techniky a webové technologie*. Brno: Computer Press, 2015. ISBN 9788025145739.

10.2 Internetové zdroje

Acceptance Testing. CODECEPTION*: Elegant and Efficient Testing for PHP* [online]. 2016 [cit. 2017-03-15]. Dostupné z: http://codeception.com/docs/03-AcceptanceTests

Chartist - API Ddocumentation. CHARTIST.JS: SIMPLE RESPONSIVE CHARTS [online]. [cit. 2017-04-13]. Dostupné z: http://gionkunz.github.io/chartist-js/api-documentation.html

Debugování a zpracování chyb NETTE FOUNDATION [online]. 2017a [cit. 2017-02-21]. Dostupné z: https://tracy.nette.org/cs/

Dependency Manager for PHP. COMPOSER [online]. 2016 [cit. 2017-02-23]. Dostupné z: https://getcomposer.org

DOBEŠ, V. *Tvorba komponent s využitím autowiringu*. *Planette* [online]. 2014 [cit. 2017-04-15]. Dostupné z: https://pla.nette.org/cs/create-components-with-autowiring

Documentation of Sass: (Syntactically Awesome StyleSheets). SASS [online]. 2016 [cit. 2017-02-22]. Dostupné z: http://sass-lang.com/documentation/file.SASS_REFERENCE.html

Documentation of UBLABOO Datagrid. UBLABOO [online]. [cit. 2017-04-13]. Dostupné z: https://ublaboo.org/datagrid/

Formuláře. NETTE FOUNDATION [online]. 2017b [cit. 2017-02-21]. Dostupné z: https://doc.nette.org/cs/2.4/forms

Gulp documentation. NPM*: Build amazing things* [online]. [cit. 2017-04-13]. Dostupné z: https://www.npmjs.com/package/gulp

HTTP request & response. NETTE FOUNDATION [online]. 2017e [cit. 2017-04-18]. Dostupné z: https://doc.nette.org/cs/2.4/http-request-response

Latte. NETTE FOUNDATION [online]. 2017d [cit. 2017-02-21]. Dostupné z: https://latte.nette.org/cs/

MÁLEK, M. *K čemu je dobrý Bootstrap a frontend frameworky? Zdroják: Zdroják, o tvorbě webových stránek a aplikací* [online]. 2013 [cit. 2017-04-24]. Dostupné z: https://www.zdrojak.cz/clanky/k-cemu-je-dobry-bootstrap-frontend-frameworky/

MONUS, A. *10 PHP Frameworks For Developers – Best of. HONGKIAT* [online]. 2015 [cit. 2017-02-11]. Dostupné z: http://www.hongkiat.com/blog/best-php-frameworks/

MVC aplikace & presentery. NETTE FOUNDATION [online]. 2017c [cit. 2017-02-21]. Dostupné z: https://doc.nette.org/cs/2.4/presenters

Nette Tester – pohodové testování. NETTE FOUNDATION [online]. 2017f [cit. 2017-03-15]. Dostupné z: https://tester.nette.org

OŽANA, R. *Gulp vs. Grunt: souboj bez vítěze a poraženého*. *Zdroják: Zdroják, o tvorbě webových stránek a aplikací* [online]. 2014 [cit. 2017-03-23]. Dostupné z: https://www.zdrojak.cz/clanky/gulp-vs-grunt-souboj-bez-viteze-a-porazeneho/

RIPRAN: Metoda pro analýzu projektových rizik. RIPRAN [online]. Brno: ACSA, 2016 [cit. 2017-04-03]. Dostupné z: http://ripran.cz/

STANĚK, Z. *Automatizace frontendových úkonů - část 2. Webová integrace* [online]. 2014 [cit. 2017-04-01]. Dostupné z: http://www.web-integration.info/cs/blog/automatizace-frontendovych-ukonu-cast-2/

TinyMCE Documentation. TINYMCE: Full featured web editing [online]. [cit. 2017-04-13]. Dostupné z: https://www.tinymce.com/docs/

11 Seznam obrázků

Obr. 1 Schéma architektury MVC 20

Obr. 2 Jednotlivé dílčí fáze životního cyklu Vodopádového modelu Zdroj: Tvorba informačních systémů, 2012 24

Obr. 3 Základní Use Case Diagram systému 42

Obr. 4 Eriksson-Penkerův diagram základních procesů systému 43

Obr. 5 Sekvenční diagram pro požadavek na zobrazení stránky 46

Obr. 6 Diagram aktivit pro kontrolu autentizace a autorizace 47

Obr. 7 Use Case Diagram pro správu obsahu webu 48

Obr. 8 Use Case Diagram pro správu internetového obchodu 50

Obr. 9 Stavový Diagram pro objednávku 51

Obr. 10 Use Case Diagram pro veřejně přístupnou část 52

Obr. 11 Diagram Aktivit pro proces uživatelské objednávky 53

Obr. 12 EER diagram pro návrh produktů 55

Obr. 13 EER diagram pro objednávku 57

Obr. 14 Drátěný model pro administrační sekci 59

Obr. 15 Výsledné rozvržení adresářové struktury 60

Obr. 16 Domovská stránka administrace – dashboard 63

Obr. 17 Ukázkový datagrid pro správu uživatelů 65

Obr. 18 Formulář pro nastavení oprávnění pro vybranou uživatelskou roli 67

Obr. 19 Ukázka vzhledu frontendového modulu 73

Obr. 20 Proces sestavení souborů skrze nástroj Gulp v konzole 75

12 Seznam tabulek

Tab. 1 Metoda RIPRAN - převodní tabulka pro hodnoty rizika 39

Tab. 2 Metoda RIPRAN - určení hrozeb a kvantifikace rizik projektu 39

Tab. 3 Metoda RIPRAN – návrhy na opatření 40

Přílohy

A Kompletní zadání požadavků společnosti

Předmluva

Jakékoliv návrhy na zlepšení jsou vítané. Já již mohu být zabředlý v těch administracích, které používám léta, a tudíž můžu mít takzvanou profesní slepotu. Nicméně se však stále rád vracím k té nejstarší tabulkové administraci, se kterou je práce nejvíce intuitivní, avšak programátor administraci s každým webem trochu upraví, někde například přidá položku, nic zásadního, ale postupem času se z toho stal neudržitelný nepořádek. Pro představu například *WordPress* mi sám o sobě přijde zbytečně dost složitý, ale líbí se mi, jak se tam jednoduše pracuje se soubory a fotkami. Zbytek *pro WordPress* je zbytečně komplikovaný, ale jeho síla leží v doplňcích - například *Visual Editor je* pro lidi „*neprogramátory*“ skvělý.

Přihlášení do administrace:

www.xxxxx.yy/tech

login: webmaster

pass: vygenerovaný (písmena velké, malá, čísla, 12 znaků)

Uživatelé s administrací neumí. Čím více funkcí, možností a „*klikátek*“ bude mít, tím horší to pro ně bude. Poměrně dobře chápou tabulky, takže nějaké jednoduché UX. Pro jednoduchost například zmíněné tabulky by nebyly špatnou volbou. O UX to pak asi bude hodně z mé strany, nebo také z tvé? Případné návrhy zpracuje náš designer.

Odhlášení

1. Uživatele to odhlásí z administrace na stránku, kde bude kontaktní formulář s možností napsat na *podpora@ideatech.cz* ohledně něčeho co by chtěl na webu dodělat, případně poradit. To však nemusí být pravidlem, stačí když zde bude příslušný odkaz na kontaktní formulář.
2. Formulář by měl mít jen 1 input = textové pole a tlačítko odeslat, nic více
3. Pak by tam ještě mělo být něco jako "*Přihlásit se*" a tím se dostanu zase zpět na začátek.

Hlavní nabídka – menu – administrace

1. **Obsah stránek**
2. **Aktuality** (pokud web bude mít)
3. **E-shop**
4. **Partnerské loga a sociální sítě**
5. **Překlady** (nepovinná položka)
6. **Uživatelé a práva**

7. **Nastavení a SEO**
8. **Sběr uživatelských e-mailových adres**
9. **Video návody k administraci**

Obsah stránek

(Tohle bude celé „grow" administrace v základní verzi)

1. **Zařazení a zobrazení položek**
   - **1.1.** Položky, které mají být na webu zařazené v hlavním menu, budou v administraci zvýrazněné a pod tyto hlavní-menu položky se budou zařazovat položky v submenu.
   - **1.2.** Pokud bude 3. submenu, tak princip zobrazení zařazení v administraci zůstává stejný jako v případě hlavního menu a submenu.
2. **Jeden řádek = 1 položka => parametry položky jsou**:
   - **2.1. Input Pořadí =>** určovat se bude podle hodnoty celého čísla - čím vyšší číslo, tím bude mít položka vyšší prioritu a proto bude zařazena v menu na vyšším místě.
   - **2.2. Název položky =>** Pod tímto názvem se bude položka na webu zobrazovat, vyplňuje se až po kliknutí na *Upravit*.
   - **2.3. Sekce Galerie => každá podstránka může mít na svém konci zařazenou galerii** - maticovou, či *slider*. Pro plnění těchto galerií bude vždy sloužit sekce v administraci pro danou správu médií. (*Výsledné zobrazení na webu a chování slideru/galerie se už bude určovat individuálně, ale pokud to naprogramuješ tak, že si budu moct zaškrtnout jestli chci slider v úvodu podstránky, nebo galerii maticově na konci, tak to bude výhoda.*)
     - 2.3.1. Po kliknutí pro vložení se zobrazí "*Vyberte soubory*", což by také mělo fungovat i pro nahrávání fotek z galerie mobilu. Nahrávání z PC/mobilu by mělo jít hromadně, množství položek by nemělo být nijak omezeno. Limit velikosti fotky by měl být nastaven na 5MB/fotka. Pokud však bude nahrávána fotka vyšší jak 800Kb, mělo by se zobrazit upozornění, aby uživatel fotky nahrával v kvalitě pro webové prohlížení.
     - 2.3.2. Až se začnou fotky nahrávat, tak uvidím progres nahrávání – není povinné.
     - 2.3.3. Až se všechny nahrají, atributu „*alt*" se automaticky přiřadí název fotky.
     - 2.3.4. Bude existovat možnost změnit název fotky, atribut *alt* a pořadí fotek. Pořadí fotek opět podle čísel. A název fotky může být použit na webu třeba pro nadpis, zobrazený nad každou fotkou.
     - **2.3.5.** U každé fotky musí být současně i možnost volby pro hromadné odstranění.

2.4. Sekce SEO – pro každou položku existuje možnost vyplnit:

2.4.1. **Title** (při založení položky se automaticky bude brát title z názvu položky, ale bude možnost jej změnit manuálně).

2.4.2. **Description** (při založení položky a vyplnění obsahu v textovém editoru, se vezme prvních 150 znaků automaticky, ale opět bude existovat možnost pro manuální nastavení. Počet znaků bude omeze na 150 znaků).

2.4.3. Případně sem zařadit další položky, co by SEO pomohly.

2.4.4. Pokud bude mít web jazykové mutace, pak bude sekce SEO rozdělená na karty/záložky:

1. SEO CZ
2. SEO EN
3. SEO DE, atd., parametry budou stejné jako u CZ**.**

2.5. Tlačítko Upravit – sekce bude rozdělená na karty/záložky:

2.5.1. **Záložka Obecné**:

1. Název položky
2. URL položky – vyplněno automaticky z názvu, možnost však URL změnit
3. Obrázek – "*Vyberte z PC*"
4. Zařazení v menu – „*Select*“ seznam, jaká položka bude vybrána, tak pod ní se bude tato nově přidávaná položka zobrazovat. Budou tedy vždy vybírány nadřazené položky. Pokud nebude vybráno nic, tak nově přidávaná položka je ta nadřazená a až pod ní se budou přiřazovat další položky.
5. Pořadí – klasika - číslo
6. *Perex* popis - Textový editor – toto pole nebude často využívané, ale musí zde být z toho důvodu, že nějaká položka může na sebe odkazovat nejenom z menu, ale i odněkud z webu a může mít perex obsah, kterým se podle uživatele bude lišil od samotného obsahu, umístěného v druhé záložce.

2.6. Záložka Obsah CZ

2.6.1. Může zde být položka „*Popisek*“ => jen prostý text. Web od webu se to vždy liší a například takovýchto položky zde může být i klidně více.

2.6.2. Dál už pouze editor, ideálně ten *CK editor*, který umožňuje vložit do obsahu fotku i soubor přímo z PC. Není možné to dělat tak, jak to mám ve starém administraci, že se vše musí nahrávat přes položku administrace „*data*“ a z editorů se na to pouze odkazuje, to

by uživatele odradilo. Dále tento editor musí obsahovat nějakou funkcionalitu, která bude umožňovat vkládání HTML elementů.

2.6.3. Uložit tlačítka vždy na každé straně administrace 2, jedno nahoře a jedno dole.

Pokud má web jazykové mutace, další záložky budou pro jazykové mutace - 1 záložka = 1 jazyková mutace, tedy například:

2.6.4. Záložka Obsah EN – parametry:

- Název
- URL (automaticky)
- Může zde být i popisek/nebo i 2 popisky, jako prostý text. Web od webu se to opět liší, hlavně podle toho, jak je navržený.

Potom jsou individuálnosti, které programátor řeší přidáním dalších záložek s editory. Není to úplně nejelegantnější způsob řešení, ale u většiny webů to vyhovující je. Potom však existují i weby dost přeindividualizované, na které je pomalu nutné navrhovat vlastní administraci. Setkal jsem již v některých řešeních s problémem, že jednotlivé podstránky jsou příliš dlouhé a vyskytuje se zde mnoho různých textových bloků na každé stránce, což z 1 editoru nelze poskládat. Rozdělit to na XX záložek po XX editorech, navíc doplněné o pevné a neměnné části by také nebylo optimální řešení, takže jsme to dříve vyřešili tak, že jakmile se přihlásíš do administrace, jdeš na web a aktualizuješ jej, tak od této chvíle lze do jakéhokoliv bloku textu kliknout a rovnou se objeví *CK-editor* přímo zde na webu. *(tato editace neplatí na bloky aktualit a jiných bloků, o kterých je jasné jak budou vypadat a mají své pevné místo, takže se přidávají běžně přes administraci)*

Aktuality

Pokud jsou na webu aktuality, zpravidla to bývá 1 typ aktualit na 1 místě. Pokud by a webu bylo více aktualit na více místech, řešilo by se to rozřazením aktuality viz parametry níže

1. Datum

1.1. Automaticky předvyplněné z aktuálního data

1.2. Možnost vybrat kliknutím do kalendáře jiné datum (ne psát manuálně)

1.3. Možnost zaškrtnout *checkbox „Nezobrazovat datum“* – tím pádem se datum nezobrazí na webu

2. Obrázek

2.1. Kliknutím na "*Vybrat z PC*" musí fungovat také na mobil – vybrat z galerie fotek v mobilu

3. Název

3.1. Jenom čistý text s možností HTML elementů - *strong* a *br*

4. Textový editor

5. Možnost určit **prioritní pořadí**, protože pokud existuje jakákoliv dlouhodobější aktualita a je potřeba, aby se nad ni nezařadili nově vložené aktuality, tak této prioritní dlouhodobé aktualitě se přiřadí opět nějaké číslo, které přebije chronologické zařazení podle data. Jinak se aktuality řadí chronologicky tak jak jim bylo určeno datem vložení. Pokud určím datum vložení v budoucnosti, tak se aktualita zobrazí až od toho dne.

E-shop

Velmi často se objevují požadavky alespoň na jednoduchý e-shop, a proto by bylo velmi vítané jej mít jako součást komplexní administrace. Pokud by se jednalo čistě o administrativní systém, část pro obchod by měla jít z projektu jednoduše odstranit, nebo deaktivovat, čímž by uživatel vůbec neměl tušení, že jeho administrace obsahuje i část s internetovým obchodem. Specifické požadavky je nutno nastudovat, nebo budou doladěny před začátkem tvorby. Je ponechána možnost seberealizace, případně všechny detaily doladíme spolu, takže zde budou definovány jen základní požadavky, které jsou:

1. **Správa produktů**
   - **1.1.** Jednoduchý a variabilní produkt – produkt s variantami - například velikosti oblečení S, M, L atd.
     - 1.1.1. Možnost vytvářet jakékoliv atributy produktů
   - **1.2.** Základní vlastnosti produktů
     - 1.2.1. Název
     - 1.2.2. Cena
     - 1.2.3. Statusy produktů – publikovaný, skrytý, koncept
     - 1.2.4. Editovatelné informace o produktu
     - 1.2.5. Jestli je možné jej přidat do košíku
     - 1.2.6. URL adresa jednotlivých produktů
     - 1.2.7. Slevy produktů a akce
     - 1.2.8. Obrázek
   - **1.3.** Kategorie produktů – například tričko, čepice, tílko
     - 1.3.1. Filtrování produktů podle jeho vlastností
     - 1.3.2. Vyhledávání produktů na webu pro usnadnění nákupu
     - 1.3.3. Víceúrovňové menu podle kategorií produktů
2. **Košík**
   - **2.1.** Možnost přidávání produktů do košíku
   - **2.2.** Košík se bude skládat ze 4 částí – Košík, Údaje zákazníka, Volba dopravy a platby, Rekapitulace. Může zde být i pátá část – děkovná stránka.
3. **Správa objednávek**

3.1. Generování čísla faktury (variabilní symbol)

3.2. Stavy objednávky – Vytvořená, Dokončená, Expedovaná, Doručená, Pozastavená, Zrušená a přechod mezi nimi

3.3. Objednávka je spárovaná se zákazníkem, účetním a dodavatelem

3.3.1. Znalost adresy + dodací adresa zákazníka a všech potřebných informací o o účetním a dodavateli

3.4. Generování splatnosti faktury

3.5. Generování faktury do PDF

4. **Více měn** – možnost vytváření, editace, mazání měn a nastavování vlastního kurzu pro jednotlivé měny
5. **Správa zákazníků** – přehled zákazníku, informací o nich a přehled jejich objednávek
6. **Slevy**

6.1. Uživatelské podle velikosti nákupu

6.2. Kupóny

6.2.1. Limit podle počtu použití

6.2.2. Limit podle data platnosti

6.3. Velkoodběratelské slevy

7. **Odesílání mailů o informacích**

7.1. Při dokončení objednávky se odesílá informační email zákazníkovi

7.2. Při jakékoliv změně stavu objednávky se odesílá znovu email zákazníkovi

7.2.1. Odeslání i s PDF fakturou

7.3. Při nové objednávce se odesílá informativní email pověřené osobě

8. **Obecné nastavení e-shopu**

8.1. Účetní nastavení – zda je či není provozovatel plátce DPH

8.1.1. Výška DPH + všechny klíčové informace o provozovateli – IČO, název, kontakt, spisová značka atd.

9. **Dashboard** se statistikami se základními informacemi o webu

Partnerská loga

Pokud má web někde před patičkou umístěný *slider* s logy, nebo případně i kdekoliv jinde, tak v administraci bude existovat možnost tyto loga editovat. Parametry jsou:

1. **Kategorie** - může chtít na každé podstránce loga jiné, obvykle to nechtějí, ale už jsem se s tím setkal, takže by bylo vhodné, když by se s tím v administraci počítalo. Samostatné kategorie by nejspíše měly být všechny možné podstránky, které jsou na webu vytvořené a kde se dá něco zobrazit. Pokud uživatel nezvolí žádnou kategorii, budou se všude zobrazovat loga stejná.

2. **Jednotlivá položka** obsahuje:
   - **2.1.** Název (ve většině případů se pak nikde stejně nezobrazuje)
   - **2.2.** Obrázek – vybrat soubor
   - **2.3.** URL (kam bude logo odkazovat po kliknutí, vždy otevírat v novém okně)
   - **2.4.** Pořadí – číslo – čím vyšší, tím výš zařazené

Překlady

Pokud bude v systému možnost tvorby překladů, tak v příslušené sekci budou v tabulce umístěny všechny slova, které se na webu nachází a které nelze upravit z textových editorů. Takže to budou nadpisy sekcí, popisy tlačítek, nápovědy, *placeholdery*, upozorňovací hlášky a jiné.
Pokud bude mít web jazykovou mutaci, např. CZ, EN, DE, bude tabulka:

1. Sloupec CZ
2. Sloupec EN
3. Sloupec DE nadpisy sloupců

Editace bude fungovat tak, že se budou pouze dopisovat slova do tabulky, poté se vše potvrdí a je hotovo.

Uživatelé a práva

1. Možnost zamezení editace některých/všech sekcí
2. Možnost vytvořit uživatelské skupiny
3. Možnost vytvořit uživatele a zařadit je do uživatelských skupin
4. Možnost uživateli vygenerovat heslo
5. Logování akcí provedených uživateli

   Pro lepší dohledatelnost a přehled nad systém by bylo vhodné všechny akce logovat, čímž by byl získán přehled kdy a kým byly akce provedeny. Bylo by proto vhodné mít někde sekci, kde bude výpis provedených akcí. Jde potom jednoznačně určit, kdo je za akci zodpovědný a zjistit, proč ta situace nastala v případě potřeby.

Nastavení a SEO

Zde budou zařazeny věci, které se na webu objevují vícekrát, např. telefonní číslo v hlavičce a patičce, email, atd.

1. **Telefonní číslo**
2. **Kontaktní email**
3. **Email** – na tento se budou odesílat vzkazy/poptávky z formulářů
4. **Titulek stránky** – homepage 60 znaků
5. **Popisek stránky** – homepage 150 znaků

6. **Klíčová slova** – homepage (vyhledávače klíčová slova ignorují, klienti však ne, takže pro efekt)
7. Současně pokud bude web mít nějakou nezařaditelnou věc, tak ta bude také zde

Sběr uživatelských e-mailových adres

Pro práci s uživateli by měly být uchovávány veškeré e-mailové adresy, které návštěvníci stránky zadají do systému. Tyto adresy budou dále využívány pro zasílání *newsletterů* případně pro cílení reklamy a jiné účely. Bylo by vhodné je jednoduše pomocí nějaké hromadné akce vyexportovat do soboru, například s koncovkou CSV, pro pozdější práci s nimi. Kdyby šly odesílat maily přímo ze systému všem, případně jen zvoleným uživatelům, byla by to zajisté velmi vítaná funkcionalita, avšak prozatím se jedná jen o doplňkovou funkcionalit pro pozdější rozšíření.

Video návody

1. Bude zde seznam popsaných základních úkonů, jak s administrací pracovat.
2. Po kliknutí na nějakou položku se otevře stránka s YouTube skrze iframe s možností jednoduše, třeba skrze *select* ze seznamu, vybrat jiný video návod a to bez nutnosti vracení se zpět.