Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

CMS systém založený na frameworku Laravel

2017

Patrik Szkandera

Vedoucí práce:
Mgr. Martin Trnečka, Ph.D.

Studijní obor: Aplikovaná informatika,
prezenční forma

Bibliografické údaje

Autor:	Patrik Szkandera
Název práce:	CMS systém založený na frameworku Laravel
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2017
Studijní obor:	Aplikovaná informatika, prezenční forma
Vedoucí práce:	Mgr. Martin Trnečka, Ph.D.
Počet stran:	43
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliograhic info

Author:	Patrik Szkandera
Title:	CMS system based on Laravel framework
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2017
Study field:	Applied Computer Science, full-time form
Supervisor:	Mgr. Martin Trnečka, Ph.D.
Page count:	43
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

Práce pojednává o naprogramování CMS (content management system), který je úzce spjat s frameworkem Laravel a pro plné využití tohoto systému se předpokládá základní znalost frameworku Laravel. V práci jsou popsány využité technologie, programovací postupy a dále práce obsahuje informace pro uživatele a programátory.

Synopsis

This thesis is about programming of CMS (content management system) closely linked to the Laravel framework. The basic knowledge of the Laravel framework is assumed for the full use of this system. The thesis describes used technologies, programming procedures and furthermore the work contains information for users and programmers.

Klíčová slova: CMS; Laravel; TDD; VueJS

Keywords: CMS; Laravel; TDD; VueJS

Děkuji Martinu Trnečkovi za odborné vedení bakalářské práce.

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

Seznam obrázků

Seznam tabulek

Seznam zdrojových kódů

1 Úvod

Při tvorbě webových stránek je ve většině případů nutnost vytvořit i administrační rozhraní, které bude aplikaci spravovat a zpřístupní tak ovládání stránek i uživatelům bez programátorských znalostí. Administrace, která toto uživateli umožňuje, se nazývá systémem pro správu obsahu (CMS z anglického content management system).

Záměrem mé práce bylo vytvoření CMS, které úzce spolupracuje s frameworkem Laravel. Hlavní motivací bylo, že jsem již několik webových stránek pod tímto frameworkem vytvořil a chtěl jsem co nejvíce zautomatizovat proces tvorby administračního prostředí. Při tvorbě administrace webových stránek jsem totiž vždy narazil na velmi podobné elementy. Většinou se jednalo o správu dat nad tabulkou (popř. tabulkami), která je tvořena výpisem dat, vytvořením nové položky, její editaci a smazáním. V některých případech je ale potřeba vykonávat i další akce, jako je např. generování PDF, zasílání emailů nebo zasílání dat službě třetí strany. Vytvořené CMS by tak mělo co nejvíce usnadňovat návrh administrace, ale zároveň by programátor měl mít možnost převzít kontrolu nad celou administrací a upravit si ji dle potřeb.

1.1 Existující řešení

Myšlenka systému pro správu obsahu samozřejmě není nijak nová a v současné době existují desítky možná i stovky hotových řešení. Při studiu již existujících populárních řešení jsem ale vždy narazil na vlastnosti, které se mi u daného CMS nelíbily.

U nejrozšířenějších CMS řešení jako je Wordpress nebo Joomla!, jsou hlavními problémy zastaralé technologie, nemožnost převzít kompletní správu nad administrací bez složitých konstrukcí a nízká rychlost. I přesto, že tyto systémy skvěle plní roli CMS, je u nich znatelně cítit absence frameworku, který by programátorům nabídl více možností.

V době začátku psaní této práce již existovalo i pár CMS řešení postavených právě na frameworku Laravel. Všechny ale byly v počáteční fázi vývoje a žádný z nich plně nesplňoval mou představu o ulehčení tvorby administračního prostředí. V době dokončování práce se mé představě nejvíce přibližuje OctoberCMS.

1.2 Vlastnosti implementovaného CMS

Jak již bylo zmíněno CMS vytvořené v rámci této práce velmi úzce spolupracuje s frameworkem Laravel. Pro jeho úplné využití se tak počítá s alespoň základními znalostmi tohoto frameworku a jeho architektury. Vytvořené CMS dostalo pracovní název LaWell, které je spojením slov Laravel a well (anglicky studna) a má znázorňovat nekonečnou studnu možností. Vytvořené CMS poskytuje následující funkcionalitu:

- Správu obsahu nad tabulkami, které v CMS existují po jeho instalaci ale i nad tabulkami, které si vytvoří uživatel.
- Správu uživatelů a možnost vytvořit několik druhů uživatelů. Jednoduše tak lze v administraci odlišit např. administrátory od zákazníků nebo jiných uživatelů.
- Správu oprávnění pomocí speciálních řetězců, které umožňují definovat přístupy pro každý požadavek a popř. i definici vlastních oprávnění.
- Tvorbu multijazyčného rozhraní a možnost překladu knihoven třetích stran pomocí uživatelského rozhraní.

2 Použité technologie

Při vývoji systému pro správu obsahu jsem se snažil vždy využít ty nejmodernější dostupné technologie pro tvorbu webu v jazycích PHP a Javascript. Volba těchto jazyků plyne hlavně z jejich všeobecné rozšířenosti a dále široké dostupnosti výukových materiálů.

2.1 Skriptovací jazyk PHP

I přes klesající zájem je jazyk PHP stále jeden z nejpopulárnějších jazyků pro tvorbu webových aplikací [1]. Tato skutečnost je dána hlavně díky značné jednoduchosti jazyka. První verze jazyka PHP vznikla v roce 1995 a od té doby jazyk prošel velmi znatelným vývojem [2]. Aktuálně nejpoužívanější verzí PHP je PHP 5 [3], nejnovější je pak verze PHP 7. PHP lze použít nejen na serveru ale také i v konzolových nebo desktopových aplikacích, k čemuž složí jeho kompilovaná verze.

2.1.1 PHP 7

PHP 7 je nejnovější verzí jazyka PHP, která přináší oproti předchozí verzi spoustu vylepšení a novinek. PHP ve verzi 6 jako takové nikdy nebylo vydáno kvůli implementačním problémům a jeho vývoj byl úplně ukončen. Aby pak nedocházelo ke zmatkům mezi již nevyvíjenou verzi PHP 6 a nově vznikající verzí, byla nová verze PHP označena jako PHP 7 [4]. Jednou z hlavních vylepšení je úplně nová a restrukturalizovaná verze Zend Enginu, což je technologie interpretující jazyk PHP. Tato verze Zend Enginu dostala nový název PHP next generation. Díky této změně PHP 7 dosahuje až dvojnásobné rychlosti oproti svému předchůdci [5]. Dále nová verze PHP přináší nové funkcionality a konstrukce do jazyka samotného, např. anonymní třídy, deklarace skalárních typů pro funkce a metody, deklaraci návratových typů, a mnohé další [6]. Celkově se PHP díky této verzi mnohem více přiblížilo moderním programovacím jazykům a standardům.

2.2 Composer

Aby bylo programování co nejefektivnější je dobrým zvykem využívat knihovny napsané jinými programátory. Pokud již využíváme nějakou knihovnu třetí strany, tak je potřeba ji průběžně aktualizovat – ať už kvůli bezpečnostním opravám, opravám chyb anebo novým funkcionalitám. Abych tady tyhle úkony nemusel dělat ručně, využil jsem nástroj s názvem Composer, který se stará o instalaci, aktualizaci i odinstalování požadovaných balíčků [7]. Pro účely kategorizace potřebných knihoven vznikla webová služba Packagist.org, kde jsou k dispozici všechny knihovny stáhnutelné pomocí Composeru.

2.2.1 Kompatibilita knihoven a sémantické verzování

Díky Composeru tedy můžeme jednoduše aktualizovat knihovnu, abychom mohli odstranit možné chyby. Ne vždy ale chceme aktualizovat knihovnu na nejnovější verzi, protože bychom mohli ztratit funkcionalitu, se kterou jsme při vývoji počítali, ale kterou nejnovější verze knihovny již nemá. Aby tedy nedocházelo k potížím s nekompatibilitou mezi knihovnami, či v projektu samotném, většina tvůrců knihoven se drží tzv. sémantického verzování, kdy se každá nově vydaná verze skládá ze tří čísel mezi sebou oddělených tečkou. Dále platí, že změna jakéhokoliv čísla vždy vynuluje čísla následující.

1. **Major změna**
   První číslo sémantického verzování by se mělo změnit pouze, pokud nastala změna, která není kompatibilní s předchozími verzemi knihovny. Aktualizace major změny tedy může zapříčinit nefunkčnost aplikace.

2. **Minor změna**
   Druhé číslo se mění v případě, že do knihovny byla přidána nová funkcionalita. Knihovna je ale stále kompatibilní s předchozí verzí

3. **Patch změna**
   Poslední číslo se mění po opravě chyby v knihovně [8].

Díky znalosti sémantického verzování, pak lze nástroji jako je Composer říci, že chceme pouze nové verze knihovny, které nám neporuší zpětnou kompatibilitu (tzn. změny minor nebo patch). Bohužel sémantické verzování je záležitost, která je zcela v rukou tvůrce knihovny a je jen na něm, jestli a jakým způsobem jej bude dodržovat. V praxi se tedy na to nelze nikdy stoprocentně spolehnout.

2.3 Framework Laravel

2.3.1 Opodstatnění frameworku

I přestože díky Composeru je zajištěn snadný přístup ke knihovnám třetích stran a tak také možnost využít obrovské množství již existujících knihoven, má jakýkoliv framework stále své opodstatnění. Jedná se totiž o ekosystém, který nejen že zastřešuje všechny základní funkcionality nutné pro tvorbu webových stránek, ale také nám předkládá konvence a osvědčené postupy, jakým způsobem by měl vývoj probíhat [9]. Tyto postupy jsou nutné, pro tvorbu kódu, který bude v budoucnosti snadno udržovatelný a srozumitelný. Díky frameworku tedy eliminujeme jeden z velkých nedostatků PHP, který programátorovi nechává až přespříliš volnou ruku a v praxi to většinou vede k velké nepřehlednosti zdrojových kódů výsledné aplikace (jazyk PHP totiž dovoluje, aby celá webová aplikace byla v extrémním případě umístěná i v jediném souboru).

2.3.2 Základní informace

Framework Laravel si v posledních letech získal oblibu mezi velkým počtem PHP vývojářů [10]. Jedná se o framework využívající architekturu MVC[1], která nám dělí aplikaci do tří základních vrstev:

1. **Model**, stará se o získávání a perzistenci dat (komunikace s databází);
2. **View**, slouží pro prezentaci dat (HTML, XML, JSON, ...);
3. **Controller**, zpracovává požadavky uživatele a zprostředkovává jakýsi most mezi výše zmíněnými vrstvami (získává data voláním Modelu a zasílá je konkrétnímu View).

2.3.3 Práce s databází a Eloquent ORM

Laravel umožňuje jednoduchou práci s databází pomocí své abstrakční vrstvy, která mimo jiné odstiňuje programátora od konkrétní implementace databáze. Díky této abstrakci je možné jednoduše zaměňovat jednotlivé implementace databází za pomocí pár jednoduchých kroků a to bez nutnosti měnit zdrojový kód. Aktuálně podporované databázové systémy jsou: MySQL, PostgreSQL, SQLite a SQL Server. Možná je však podpora i dalších databázových systémů pomocí balíčků třetích stran.

Základem práce s databází je QueryBuilder, třída umožňující tvořit a spouštět všechny možné druhy dotazů pomocí jednoduché a intuitivní syntaxe. Na této abstrakci je pak postaven daleko rozsáhlejší nástroj s názvem Eloquent ORM. Vychází z principu ORM (object-relational mapping), kdy každá n-tice relace je namapována na instanci třídy, která reprezentuje danou relaci. Díky tomu jsme schopni s jednotlivými n-ticemi pracovat jako s objekty, nad kterými můžeme volat široké spektrum metod. Navíc nejsme omezeni pouze na metody, které nám dodává Eloquent ORM, ale můžeme snadno definovat i metody vlastní.

2.3.4 Definice vztahů mezi modely

Jeden z nejsilnějších nástrojů Eloquent ORM je možnost jednoduchého definování vztahů mezi jednotlivými modely. Pro definici vztahu mezi dvěma modely je pouze potřeba vytvořit metodu, která obsahuje volání jedné či více metod Eloquent ORM. Volaná metoda se liší podle vztahu mezi jednotlivými modely, jak uvádí tabulka 1.

2.3.5 Pojmenovávací konvence

Pro usnadnění práce je dobré dodržovat pojmenovávací konvence Laravelu pro tabulky a modely, které tabulky reprezentují. Díky dodržování pár jednoduchých

[1] zkratka pro model-view-controller

Tabulka 1: Seznam základních vztahů mezi modely

Název vztahu	Metoda definující vztah	Metoda definující inverzní vztah
1:1	hasOne	belongsTo
1:N	hasMany	belongsTo
M:N	belongsToMany	belongsToMany

pravidel můžeme např. při psaní modelu vypustit informaci o tom, jakou tabulku reprezentuje. Největší výhodou ale je ještě jednodušší definice vztahů. Pokud bychom nedodržovali tyto konvence, tak pak např. při definici vztahu M:N bychom museli metodě předávat až 4 parametry. Naopak při dodržování konvencí pak stačí předat pouze název třídy modelu, se kterým chceme vztah definovat.

Jméno tabulky je vždy v množném čísle a je psáno pomocí tzv. snake_case, kdy jsou všechna písmena malá a slova jsou oddělena podtržítkem. Název modelu je naopak v jednotném čísle a psán pomocí tzv. StudlyCase, kdy každé písmeno nového slova je velké a jednotlivá slova se mezi sebou neoddělují. Názvy cizích klíčů se odvozují podle názvu metody, která definuje daný vztah. Laravel vezme název metody v jednotném čísle, přetransformuje jej do snake_case a jako suffix přidá název primárního klíče.

Speciálním případem je pak definice vztahu M:N, kdy je potřeba další tabulka pro korektní uložení vztahu. Název této tabulky je dán složením názvů oněch tabulek v jednotném čísle a oddělením pomocí podtržítka. Cizí klíče jsou dány názvem modelu a jménem jeho primárního klíče.

2.3.6 Automatizace testování

Framework Laravel byl vytvořen s ohledem na automatické testování a tak i bez jakýchkoliv dalších knihoven má programátor možnost začít s rychlým psaním testů. Laravel dokonce ve své poslední verzi (Laravel 5.4 v době psaní) přidává nástroj s názvem Laravel Dusk, který umožňuje simulování uživatelova chování přímo v prohlížeči. Jedná se tak o nástroj, který umožňuje psát akceptační testy.

2.3.7 Další funkcionality frameworku

Laravel poskytuje vývojářům spoustu dalších nástrojů, které urychlují a zkvalitňují vývoj webových aplikací. Popsat je všechny není záměrem této seminární práce, proto níže uvádím již jen zkrácený výčet základních funkcionalit a nástrojů.

Service container služba zajišťující tzv. dependency injection. Dependecy injection je termín, který říká, že závislosti třídy jsou ji předány metodou nebo konstruktorem.

Service providers přidává do frameworku další vrstvu, která se stará o samotnou inicializaci aplikace a potřebných závislostí.

Middleware jedná se o další vrstvu frameworku, která umožňuje filtrovat jednotlivé HTTP požadavky, ještě předtím, než se dostanou k samotné aplikaci (typicky controlleru).

Routing jedná se o nástroj, který nám umožňuje jednoduše a efektivně zpracovávat požadavky uživatele na server (navštívení konkrétní adresy, odeslání formuláře, ...).

Artisan console příkazový řádek poskytující množství funkcí, které urychlují tvorbu aplikací ve frameworku. Artisan také přináší rozhraní umožňující přidávat vlastní funkce.

Blade jednoduchý ale i tak velmi silný šablonovací jazyk, který je používán v jednotlivých view. Šablonovací jazyk přináší hlavně značně přehlednější strukturu jednotlivých šablon oproti použití čistého PHP.

2.4 Automatizace testování

Při tvorbě jakékoliv aplikace je vždy nutností otestovat, zda napsaný kód funguje tak, jak jsme při jeho psaní zamýšleli. Jednoduché aplikace, popřípadě aplikace, u kterých nepředpokládáme další vývoj, můžeme testovat klasickým způsobem, kdy vždy po dopsání kódu ověříme v prohlížeči jeho funkčnost. U aplikací větších rozměrů by se ale klasické testování mohlo časově velmi prodražit. Kontrolovat po každé rozsáhlejší změně veškeré funkcionality aplikace by mohlo zabrat desítky minut ne-li hodiny, dále nemluvě o chybovosti či nepozornosti osoby provádějící testy. Aby programátor netrávil hodiny pouze testováním, využívá se takzvaného automatického testování.

Díky automatickému testování si programátor může připravit sadu testů, které simulují chování uživatelů aplikace a testují výsledky jejich počinů. Takto vytvořené testy lze kdykoliv spustit a ověřit tak funkčnost aplikace v aktuálním stavu. Při vytváření sad automatických testů si tak programátor může být alespoň částečně jist, že jeho aplikace funguje správně. Obecně se ale nelze na automatické testování stoprocentně spolehnout, protože ne vždy lze napsat test pro každou situaci, která může nastat.

2.4.1 Framework pro testování - Codeception

Aby bylo testování ještě efektivnější, nabízí se využít framework, který byl vytvořen přímo pro testování webových aplikací napsaných v jazyce PHP. Jedním z takových frameworků je právě Codeception [11]. Codeception programátorovi poskytuje spousty možností, jak testovat svou aplikaci a podporuje modularitu testů (využití různých kusů testovacích kódů napříč testy).

Existuje mnoho druhů testů, které se liší tím, co testují, popřípadě jak to testují. Bohužel v testovacím žargonu se můžeme setkat s definicemi, které nejsou úplně vymezeny, a každý je vysvětluje trochu jinak. Já v této práci uvádím definice tak, jak je chápe právě framework Codeception, protože mi připadají z

hlediska testování webových aplikací nejlogičtější. Codeception dělí testy do tří základních skupin:

Akceptační testy – testy z pohledu koncového uživatele
Jedná se o testy, které se provádějí přímo v prohlížeči (ať už v reálném prohlížeči nebo v emulátoru prohlížeče). Výhodou těchto testů je, že mohou být napsány pro jakoukoliv aplikaci dokonce i bez znalosti jejího zdrojového kódu. Akceptačními testy jako jedinými lze také testovat funkčnost JavaScriptu. Velkou nevýhodou těchto testů je jejich rychlost. Využití prohlížeče (popřípadě jeho emulátoru) vyžaduje značnou režii a tak je tato skupina testů nejpomalejší.

Funkcionální testy – testování pomocí simulování webového serveru
Funkcionální testy jsou podobné těm akceptačním, akorát se při jejich vykonávání nevyužívá prohlížeč, což má samozřejmě drastický účinek na jejich rychlost. Díky tomu, že je možné nasimulovat požadavky na webový server a i jeho odpověď, můžeme snadno testovat aplikaci jako celek. Skutečnost, že se při testování nevyužívá prohlížeče, však má i značnou nevýhodu, pomocí funkcionálních testů nelze testovat JavaScript.

Unit testy – testování funkčnosti jednotlivých metod či funkcí
Unit testy jsou základem každého automatizovaného testování. Jedná se o nejrychleji prováděné testy, pomocí kterých můžeme snadno pokrýt i krajní případy použití. Nevýhodou testů je, že s nimi nemůžeme testovat aplikaci jako celek. Dále jsou tyto testy nejvíce náchylné na změny ve zdrojovém kódu a často musí být upravovány pro aktuální požadavky aplikace.

2.4.2 Programování řízené testy

S automatizací testování blízce souvisí programování řízené testy (anglicky TDD – test driven development). Jedná se o styl programování, který probíhá opačným způsobem než klasické psaní automatických testů. V první řadě programátor napíše test pro funkcionalitu, která ještě neexistuje, a následně se snaží tento test splnit. Tedy programátor potřebuje nejdříve správně definovat jaké vstupy a výstupy má požadovaná funkcionalita mít, zanést tytu požadavky do testu a následně programovat tak dlouho, dokud tyto požadavky nesplní. Tímto stylem programování přirozeně vzniká aplikace pospolu s automatizovanými testy.

Při psaní CMS jsem se snažil držet tohoto programovacího stylu. Díky tomu vzniklo celkem 170 testů (60 funkcionálních a 110 unit testů), které testují různé aspekty této aplikace.

2.5 NPM

Již byl zmíněný správce knihoven pro PHP, kterým je Composer. Pro správu knihoven JavaScriptu existuje podobný nástroj, který se jmenuje NPM [12].

Uplatňují se zde stejné principy pro verzování kódu a práce s NPM je velice podobná práci s Composerem.

2.6 ECMAScript

Pro JavaScript existuje standardizace s názvem ECMAScript, která určuje, jakým způsobem se bude JavaScript dál vyvíjet. Nejnovější verzí ECMAScriptu je ES7 a další verze (ES8) je plánována na rok 2017. Díky rychlému vývoji JavaScriptu, který se v posledních letech udál, nestíhají jednotlivé prohlížeče dostatečně rychle implementovat nové standardy a pro to vznikla nutnost pro překlad JavaScriptu z nové verze do starší. Jedním z takových překladačů je Babel [13].

2.7 Bootstrap

Bootstrap je sada CSS stylů a JavaScriptového kódu, které nám ulehčují práci při vytváření responzivních webových stránek a uživatelského rozhraní [14]. Bootstrap obsahuje šablony, které jsou založeny na HTML a CSS, určené pro stylizaci základního rozložení stránek, typografie, formulářů, tlačítek a dalších elementů, které se vyskytují na webových stránkách. Díky tomuto frameworku tedy může designer velmi rychle načrtnout základní rozložení stránek a to za pomocí jednoduchého HTML kódu a vhodných CSS tříd.

Kromě výše zmíněných šablon, Bootstrap obsahuje i složitější komponenty, pro jejichž fungování je nutné využít JavaScript. Všechny tyto komponenty jsou dále závislé na JavaScriptové knihovně jQuery [15], která výrazně ulehčuje manipulaci s HTML. Ke komponentám, které vyžadují JavaScript se řadí například modální dialogové okna, rozbalovací nabídka, bublinová nápověda, a další.

2.7.1 Bootstrap V4

V době psaní této práce je vývoji 4. verze frameworku Bootstrap. Tato nová verze je kompletně přepracovaná a zpětně nekompatibilní. Jednou z hlavních změn je využití tzv. flexible box (flexbox) pro rozvržení struktury webové stránky. Flexbox je nová komponenta CSS3, která se zaměřuje na správné zobrazení obsahu pro displeje různých velikostí a umožňuje vytvářet rozložení elementů, kterého bylo možné doposud dosáhnout jen pomocí hacků nebo JavaScriptu (například stejná velikost elementů v jednom řádku).

2.8 Načítání knihoven

Pokud bychom všechny JavaScriptové a CSS knihovny načítali zvlášť, tak rychle narazíme na pomalé načítaní celého webu. Což je způsobeno tím, že prohlížeče mají limitovaný počet souběžně stahovaných souborů. Je proto potřeba všechny používané knihovny sloučit do jednoho velkého souboru, který je pak jako jediný

načítán webovou stránkou. Dalším urychlením načítání je samozřejmě zmenšení velikosti načítaných souborů, což se u JavaScriptu a CSS provádí tzv. minifikací[2].

2.8.1 Webpack

Jeden z nástrojů, který to umožňuje je Webpack [16]. Po instalaci pomocí NPM, je možné JavaScriptovými funkcemi určit jednotlivé akce, které jsou prováděny po spuštění Webpacku. Tyto akce nám umožňují např.: sloučit skupiny JavaScriptových nebo CSS souborů, přeložit JavaScriptové soubory pomocí překladače Babel, kopírovat soubory, ...

Pro efektivní využití Webpacku je potřeba jej spustit s příznakem watch. Díky tomuto příznaku bude Webpack kontrolovat JavaScriptové a CSS soubory a pokud programátor provede nějakou změnu, kterou uloží, Webpack se opět spustí a vykoná požadované akce (sloučení, minifikace, překlad).

2.9 Vue.js

Vue.js je JavaScriptový framework, který se zaměřuje na budování uživatelského rozhraní [17]. Tento framework získal v posledních dvou letech spoustu pozornosti, která stále narůstá [18]. Jeden z důvodů vysoké popularity tohoto frameworku je určitě fakt, že si jej zvolil Laravel jako základní JavaScriptový framework a tak každý nově vytvořený projekt v Laravelu má automaticky připravené vše pro práci s Vue.js.

Vue.js byl navrhnut tak, že jej lze snadno integrovat s již existujícími knihovnami. Není proto problém Vue.js využívat i s velmi rozšířenou JavaScriptovou knihovnu jQuery. Na druhou stranu je možné Vue.js také využít i jako samostatný framework pro budování tzv. SPA[3]. Single page application je označení webové aplikace, která se snaží napodobit chování uživatelského rozhraní, jak jej známe pro desktopové aplikace.

2.9.1 Deklarativní vykreslování

Jednou ze základních funkcionalit Vue.js je tzv. deklarativní vykreslování, které nám umožňuje jednoduchým způsobem aktualizovat stránku pomocí speciálních direktiv v HTML. Využívá se k tomu tzv. reaktivních proměnných, které si vyžadují speciální definici. Tato definice umožňuje, aby hodnoty proměnných mohly být „sledovány“ a po jejich změně je příslušným způsobem aktualizován i HTML kód stránky. Speciální syntaxe Vue.js nám v HTML umožňuje například schovávat a zobrazovat obsah na základě pravdivostní hodnoty, vypisovat obsah proměnné nebo iterovat přes obsah proměnné.

[2] odstranění komentářů, přebytečných mezer a řádků
[3] zkratka pro single-page application

2.9.2 Modely

Pro práci s formulářovými prvky využívá Vue.js opět speciální syntaxe v HTML. Jedná se o direktivu v-model, které se jako hodnota předá název reaktivní proměnné. Pokud na nějakém formulářovém prvku správně použijeme direktivu v-model, stane se tento prvek automaticky oboustranně reaktivní. Znamená to, že po upravení obsahu ve formulářovém prvku se změní obsah reaktivní proměnné a také, že po upravení reaktivní proměnné se automaticky aktualizuje hodnota formulářového prvku na hodnotu reaktivní proměnné.

2.9.3 Komponenty

Komponenty jsou podle mě největší výhodou Vue.js. Když pominu to, že díky nim snadno vzniká znovupoužitelný kód, tak hlavní výhodou je, že nutí programátora dodržovat určitou strukturu kódu. JavaScript jako takový je totiž opět jazyk, který nechává programátorovi absolutně volnou ruku v tom, jakým způsobem bude svůj kód strukturovat. Na každou komponentu se můžeme dívat jako na třídu, rozdělenou do několika částí:

Template předpis pro vzhled komponenty. Vue.js umožňuje definovat šablonu přímo v kódu dané komponenty, nebo i v odděleném souboru.

Props definice dat, které do komponenty přichází z vnějšku. Pomocí props můžeme komponentě předávat různá data přímo z HTML.

Data definice reaktivních dat.

Computed data jedná se o speciální data, jejichž hodnota je vypočítána až v okamžiku, kdy se k nim přistupuje. V objektově orientovaném programování se jedná o náhradu za tzv. getters.

Watchers umožňuje nám sledovat změny hodnot proměnných. V objektově orientovaném programování se jedná o náhradu za tzv. setters.

Methods část určená pro definování metod komponenty.

Components definice zanořených komponent.

Speciální metody volané při změně stavu komponenty díky těmto metodám můžeme v komponentě reagovat na určité důležité změny, které mohou v jejím životě nastat (např. vytvoření nebo zrušení komponenty).

3 Uživatelská příručka

3.1 Instalace

CMS bylo vytvořeno jako knihovna, kterou lze nainstalovat a dále spravovat pomocí nástroje Composer. Jednotlivé kroky instalace jsou následující:

1. Nejprve je nutné vytvořit nový projekt s frameworkem Laravel. Nejrychlejší způsob jak toho docílit je za pomoci Composeru a stačí na to jednoduchý příkaz: `composer create-project --prefer-dist laravel/laravel nazev-projektu`

   Tento krok může chvíli trvat, protože se bude stahovat a instalovat framework Laravel a všechny jeho závislosti.

2. Po vytvoření nového projektu je potřeba nastavit alespoň údaje pro připojení k databázi, což lze udělat v souboru *.env*.

3. Následujícím krokem je přidání adresy repozitáře. Jelikož je celá knihovna prozatím umístěna pouze v soukromém repozitáři, je nutné před jejím stažením napovědět Composeru, kde jí má hledat. Docílíme toho upravením konfiguračního souboru Composeru. Tento soubor nalezneme v kořenovém adresáři nově vytvořeného projektu pod názvem *composer.json* a je nutné do něj přidat následující kód:

```
1  "repositories": [
2      {
3         "type": "vcs",
4         "url": "https://Szkandy:tUBGdTFVSRNDAysneab4@bitbucket.org/
              lawell/core.git"
5      }
6    ]
```

Zdrojový kód 1: Příprava Composeru pro instalaci ze soukromého repozitáře

4. Nyní by měl být Composer připraven stáhnout knihovnu ze soukromého repozitáře. Samotné stahování započne zadáním příkazu: `composer require lawell/core:dev-master`

5. V dalším kroku je potřeba informovat Laravel o nové knihovně. Pro tento účel je třeba v projektu zaregistrovat nový Service Provider. Všechny Service Providery jsou umístěny v souboru *config/app.php* v poli pod klíčem 'providers'. Přidáme tedy do tohoto souboru Service provider knihovny.

6. Po přidání Service Provideru je možné spustit samotnou instalaci CMS, která se postará o vytvoření databáze a zkopírování potřebných souborů.

```
1   /*
2    * Package Service Providers...
3    */
4    Laravel\Tinker\TinkerServiceProvider::class,
5    LaWell\Core\Providers\LaWellServiceProvider::class,
```

Zdrojový kód 2: Ukázka přidání Service Provideru

Instalace se spustí pomocí Artisan příkazového řádku a to konkrétně příkazem: `php artisan lawell:install [-d]`. Nepovinný přepínač *-d* v příkazu nainstaluje knihovnu s ukázkovou administrací.

Po provedení všech výše uvedených kroků je CMS připravené k použití. Pro přístup do administrace je nutné přidat za adresu projektu */admin*. Přednastavené přihlašovací údaje do administrace jsou: admin@site.com, 123456.

3.2 Nastavení

Obecná nastavení týkající se vytvořeného CMS jsou uložena po vzoru Laravelu, v souboru *lawell.php* jako asociativní pole. Tento soubor je po instalaci zkopírován do adresáře *config*. V tomto souboru lze například upravit jazyky, do kterých má být administrace lokalizována nebo zda má být menu automaticky vyskládáváno. Pokud se dále v práci budu odkazovat na nastavení CMS, budu tím myslet právě tento soubor.

3.3 Správa oprávnění

Přístup k jednotlivým oprávněním je řízen pomocí rolí a oprávnění. Každý uživatel je přiřazen k právě jedné roli a každá role může mít jedno nebo více oprávnění. Definování jednotlivých oprávnění je inspirováno routováním v Laravelu, kdy každý HTTP požadavek je nasměrován na metodu controlleru, jenž má požadavek obsluhovat. Jednotlivé routovací akce mají tvar: *Controller@metoda*, kde Controller je celý název třídy controlleru včetně namespace a metoda je název metody controlleru.

Stejným způsobem probíhá definice oprávnění s tou výjimkou, že lze použít speciální zástupný charakter *, který zastoupí všechny možné kombinace znaků. Díky tomuto speciálnímu znaku lze definovat oprávnění pro celý controller nebo dokonce pro skupinu controllerů pod stejným namespace.

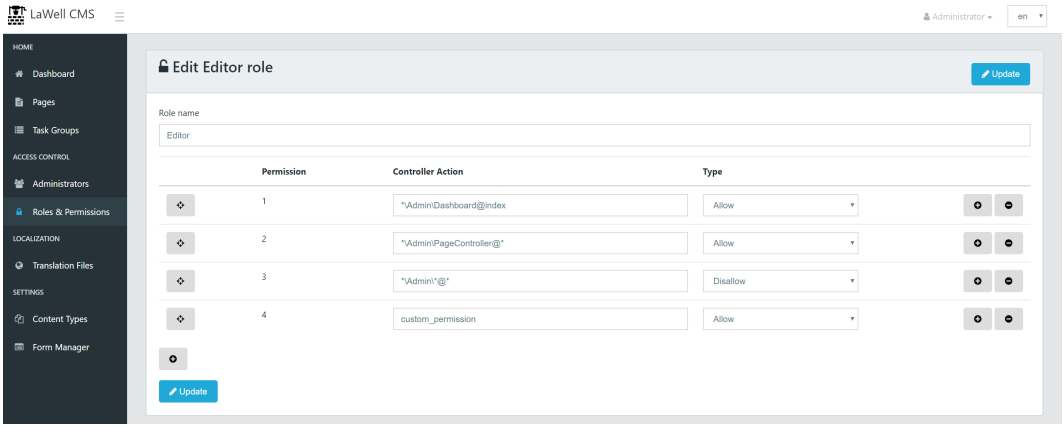
Příklady definic oprávnění:

Controllers\Site\Controller@method definuje oprávnění pro jednu metodu konkrétního controlleru.

Controllers\Site\Controller@* definuje oprávnění pro všechny metody konkrétního controlleru.

Controllers\Site*@* definuje oprávnění pro všechny controllery pod namespace *Controllers\Site* a všechny jejich metody.

Dále každé oprávnění nese informaci o pořadí a typu. Pořadí určuje, kdy se oprávnění bude vyhodnocovat a typ určuje, zda se jedná o oprávnění přijímající či zamítající. Obrázek 1 zobrazuje rozhraní pro správu oprávnění.



Obrázek 1: Správa oprávnění

3.3.1 Vyhodnocení oprávnění

Pro každou akci, která je kontrolována middlewarem LaWellMiddleware, se postupně vyhodnocují všechna oprávnění právě přihlášeného uživatele. A podle prvního oprávnění, které odpovídá dané akci, bude uživateli buďto umožněn přístup k akci, nebo bude přesměrován zpátky na předchozí akci, ze které byl požadavek odeslán. Pokud se nenalezne žádné oprávnění, které by odpovídalo vykonávané akci, bude požadavek automaticky zamítnut.

3.3.2 Uživatelsky definovaná oprávnění

Uživatel může také definovat svá vlastní oprávnění pomocí libovolného řetězce, který se nepodobá výše uvedeným definicím. Tato oprávnění nebudou testována automaticky, ale uživatel si je musí otestovat manuálně pomocí metody `can('nazev_opravneni')` třídy *User* nebo třídy *Role*. Tímto způsobem lze samozřejmě také testovat, zda uživatel má či nemá oprávnění přístupu k dané akci.

3.4 Správa překladů

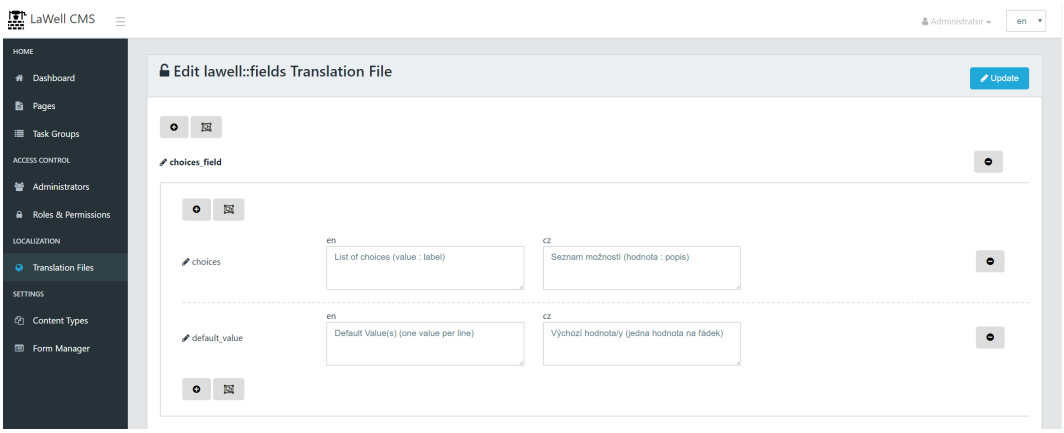
Framework Laravel a knihovny pro něj napsané ukládají své překlady v PHP souborech jako asociativní pole. Správce překladů je nástroj, který umožňuje tyto soubory vytvářet, editovat a mazat. Díky tomuto nástroji tak lze překládat

texty aplikace ale i knihoven. Na úvodní stránce správce překladů se v tabulce zobrazí nalezené překladové soubory aplikace všech nastavených jazyků. Nad tabulkou pak lze přepínat mezi překladem aplikace nebo knihoven.

Po otevření konkrétního souboru ve správci překladů se zobrazí víceúrovňová tabulka obsahující jednotlivé klíče a jejich překlady pro všechny nastavené jazyky. Klíče lze po kliknutí na jejich název upravovat.

Při přidávání nových klíčů si uživatel může vybrat, zda chce vytvořit jednoduchý překlad, nebo skupinu překladů, která vytvoří v tabulce nové zanoření. Názvy klíčů jsou povinné a musí být v každé úrovni mezi sebou unikátní, jinak by po uložení nemohlo vzniknout validně formátované asociativní pole. Tento požadavek vynucuje samotný správce překladů a po dokončení tvorby nebo úpravy klíče automaticky doplní a opraví jeho název.

Pokud uživatel bude překládat nebo upravovat texty nějaké knihovny, nebudou se tyto překlady propisovat do originálních souborů, ale automaticky se vytvoří jejich kopie. V administraci tak nikdy nelze upravovat překladové soubory knihoven a veškeré úpravy se projeví pouze na jejich kopiích (včetně mazání). Obrázek 2 zobrazuje rozhraní pro správu překladů.



Obrázek 2: Správa překladů

3.5 Meta atributy

Ve vytvořeném CMS se na několika místech využívají tzv. meta atributy. Jedná se o jednoduchou variantu, jak implementovat různorodé a proměnlivé informace, které by v klasické tabulce implementovat nešly (za předpokladu omezeného počtu sloupců v tabulce). Pomocí meta atributů lze ukládat částečné informace, jejichž zkombinováním získáme výsledná data. Jelikož byl CMS ze začátku vyvíjen s možností správy multijazyčného obsahu jsou i meta atributy multijazyčné a skládají se ze dvou tabulek: meta_attributes a meta_attribute_translations.

V tabulce meta_attributes jsou uloženy informace o názvu atributu a ke kterému objektu patří. Pro definici vlastnictví atributu se využívá tzv. polymorfní

relace, kdy se kromě id rodiče ukládá i jeho typ. Jako typ se používá název modelu, který atribut uložil. Díky tomu lze meta atributy využívat u více na sobě nezávislých tabulek.

V tabulce meta_attribute_translations se pak ukládají hodnoty. V případě multijazyčného atributu nese řádek i informaci o jazyku. Poslední informací je pak typ meta atributu, který je automaticky odvozen při ukládání meta atributu. Díky čemuž je možné do meta atributů jednoduše ukládat i číselné informace nebo celá pole.

3.5.1 Inicializace

Meta atributy může využívat každý model. Nejjednodušší způsob, jak toho docílit je využitím traity s názvem *LaWell\Core\Models\Traits\MetableTrait*. Využitím této traity se třída obohatí o spoustu funkcí pro práci s meta atributy. Pro povolení meta atributů je dále ještě nutné definovat názvy atributů, které není možné použít jako meta atributy. Tato definice probíhá v třídní proměnné `$disallowedMetaAttributes`. Typicky by se mělo jednat o pole, které obsahuje názvy sloupců tabulky, kterou model reprezentuje.

Dále lze u meta atributů nastavit přístup pomocí tzv. magických metod. Pokud definujeme třídní proměnnou na `$useMagicForMetaAttributes` na hodnotu pravdy, můžeme k jednotlivým meta atributům přistupovat, jako by se jednalo o atributy samotné třídy.

3.5.2 Základní operace s meta atributy

Výše zmíněný trait dává modelu několik jednoduchých metod pro práci s meta atributy. Pomocí metody `getMetaAttribute` lze vybrat meta atribut podle jeho jména. Pokud uživatel nepotřebuje přistupovat k meta atributu jako k objektu a stačí mu pouze jeho hodnota, může využít metodu `getMetaValue`. Pokud uživatel přistupuje k meta atributu s více překlady a neuvede jazyk, je automaticky použit aktuální jazyk aplikace. Toto nastavení se přebírá z Laravelu a je uloženo v souboru *config/app.php* pod klíčem locale.

Pro ukládání meta atributů jsou k dispozici dvě metody, která každá provádí ukládání trochu jinak. První metodou je `saveMetaAttribute`, která rovnou uloží meta atribut do tabulky. Před voláním této metody je tedy nutné aby model již existoval v databázi, kvůli propsání cizích klíčů. Druhou metodou pro ukládání meta atributů je `setMetaAttribute`. Tato metoda přímo meta atributy neukládá, ale řadí je do fronty pro uložení. Jakmile bude rodičovský model uložen, proběhne i hromadné uložení všech nastavených meta atributů. Pro odstranění slouží metoda `deleteMetaAttribute`.

3.5.3 Využití magických metod

Pro usnadnění práce lze k meta atributům přistupovat, jako by se jednalo o atributy modelu samotného. Lze toho docílit pomocí tzv. magických metod. Tyto

```
1  $model = new Model();
2  // Model není uložen, můžeme pouze připravit meta atributy
3  // pro uložení
4  $model->setMetaAttribute('prvni_atribut', 'hodnota'); ;
5  // Třetí parametr určuje jazyk meta atributu
6  $model->setMetaAttribute('druhy_atribut', 'V češtině', 'cz');
7  // Uložení modelu a jeho atributů
8  $model->save();
9  // Nyní již lze meta atributy rovnou ukládat
10 $model->saveMetaAttribute('prvni_atribut', 'jina_hodnota');
11
12 // Model reprezentující atribut
13 $atribut = $model->getMetaAttribute('druhy_atribut');
14 // Hodnoty lze získávat bez překladu nebo s konkrétním překladem
15 $hodnota = $model->getMetaValue('prvni_atribut');
16 $hodnota = $model->getMetaValue('druhy_atribut', 'cz');
17 // Následující hodnota je stejná jako předešlá
18 // za předpokladu že locale je nastaveno na 'cz'
19 $hodnota = $model->getMetaValue('druhy_atribut');
20
21 // Odstranění meta atributu automaticky smaže i všechny
22 // jeho překlady
23 $model->deleteMetaAttribute('druhy_atribut');
```

Zdrojový kód 3: Ukázka práce s meta atributy

metody jsou volány pokaždé, když je přistupováno k jakémukoliv atributu modelu. Pokud uživatel potřebuje přistupovat k meta atributům, které mají více překladů, lze toho docílit pomocí speciální notace využívající dvojtečku.

3.6 Podpora překladů

Podobně jako pro meta atributy lze i pro překlady využít speciální trait s názvem *LaWell\Core\Models\Traits\TranslatableTrait*. Pro využití její funkcionality se předpokládá s existencí dvou tabulek. První tabulka nese obecné informace a data, které nejsou překladatelné. Druhá tabulka nese všechny překladatelné informace ve všech jazycích.

3.6.1 Inicializace

Abychom se vyhnuli nutnosti definovat vztah mezi tabulkami zvlášť, je potřeba držet se dříve zmíněných pojmenovávacích konvencí, a navíc druhá tabulka musí obsahovat suffix *_translations*. Pokud uživatel nedodrží zmíněné konvence, lze název modelu s překlady uvést ve třídní proměnné `$translationModel`. Dále je potřeba definovat, které atributy jsou překladatelné pomocí pole s názvem `$translatableAttributes`.

```
1   $model = new Model();
2   // Meta atributy nastavované pomocí magických funkcí
3   // se ukládají až v okamžiku uložení modelu
4   $model->prvni_atribut = 'hodnota';
5   $model->{'druhy_atribut:cz'} = 'V češtině';
6   // Uložení modelu a jeho atributů
7   $model->save();
8
9   // Pomocí magických funkcí dostáváme rovnou hodnotu
10  $hodnota = $model->prvni_atribut;
11  $hodnota = $model->{'druhy_atribut:cz'};
```

Zdrojový kód 4: Ukázka práce s meta atributy s využitím magických metod

3.6.2 Práce s překlady

Pro práci s překlady se využívá podobných konvencí, jako u práce s meta atributy. Pro získání konkrétního překladu slouží metoda `getTranslation`, které se jako jediný parametr předává kód jazyka. Opět platí, že pokud nebude metodě předán žádný jazyk, použije se aktuální jazyk aplikace.

Pro ukládání překladu lze opět využít dvě metody `setTranslation` a `saveTranslation`, jejichž funkce je stejná jako v případě meta atributů. Novinkou oproti meta atributům je metoda `translate`, pomocí které lze uložit jeden nebo i více překladů. Pokud jsou metodě předány dva parametry - jazyk a překlad, je uložen pouze tento jeden překlad. Pokud je ale metodě předáno pouze pole, bude uložen jeden nebo i více překladů.

Mazání překladů probíhá pomocí funkce `deleteTranslation`.

K jednotlivým překladům lze opět přistupovat jako k atributům modelu, pomocí magických metod. Pro zapnutí této funkcionality je potřeba nastavit třídní proměnnou `$useMagicForTranslatableAttributes` na hodnotu pravdy.

3.7 Správa obsahu

Správa obsahu je samozřejmě nejdůležitější částí každého CMS a je tomu tak i u této bakalářské práce. Při vývoji jsem se snažil mít celou dobu na paměti, že nezle vytvořit dokonalé CMS pro každou možnou situaci. Proto by měla být správa obsahu snadno rozšiřitelná a v případě potřeby by měl uživatel mít možnost převzít nad správou obsahu kompletní kontrolu.

Správa obsahu je tvořena ze dvou částí. První částí jsou obsahové typy, pomocí kterých se definuje nad jakou tabulkou bude správa obsahu probíhat a další základní informace. Druhou částí jsou formuláře, které určují jaký obsah má daný obsahový typ spravovat.

```php
$model = new Model();
// Překlad je předáván jako asociativní pole
$model->setTranslation('cz', ['nazev' => 'hodnota']);
$model->save();

// Vytváření několika překladů zároveň
// Pomocí dvourozměrného pole
$model->translate([
        'en' => ['nazev' => 'EN Name'],
        'cz' => ['nazev' => 'CZ Nazev'],
  ]);
  // Pomocí jednorozměrného pole
  $model->translate([
    'nazev:en' => 'EN Name', 'nazev:cz' => 'CZ Nazev'
  ]);

  // Získání konkrétního překladu
  $model->getTranslation('cz');
  // Získání překladu aktuálního jazyka aplikace
  $model->getTranslation();

  // Odstranění překladu
  $model->deleteTranslation('en');

  // Přístup pomocí magických metod
  $model->{'nazev:en'} = 'EN Name';
  $model->{'nazev:cz'} = 'CZ Nazev';
  $nazevCz = $model->{'nazev:cz'};
```

Zdrojový kód 5: Ukázka práce s překlady

3.7.1 Obsahové typy

Pro vytvoření samostatného CRUD[4] bloku je nejdříve nutné vytvořit obsahový typ, který jej bude reprezentovat. Pro takový obsahový typ je dále také nutné vygenerovat controller a model. Tyto dvě třídy jsou pak v plné moci uživatele, který je může bez problémů dál upravovat, aniž by se musel bát, že by po aktualizaci o tyto změny přišel.

U každého obsahového typu můžeme definovat několik informací:

Název
: Název musí být unikátní a jednoznačně určuje daný obsahový typ. Název dále také určuje, jak se budou jmenovat vygenerované třídy. Při vytváření nového obsahového typu je dobré dbát na pojmenovávací konvence Laravelu a pojmenovat jej tak, jak chceme aby se jmenoval vygenerovaný model.

[4] zkratka z anglického create, read, update, delete

Typ
: Existují celkem čtyři možné typy, které značně ovlivňují chování obsahových typů.

- Základní obsah využívá předem vytvořené tabulky content a meta atributů. Pomocí tohoto typu lze jednoduše vytvářet jednoduché administrační bloky bez nutnosti vytvářet nové tabulky.
- Uživatel využívá tabulku users a umožňuje vytvořit nový druh uživatele a definovat k němu vlastní administraci.
- Vlastní tabulka vytváří správu obsahu nad uživatelem definovanou tabulkou.
- Nastavení využívá předem vytvořené tabulky settings a meta atributů. Od základního obsahu se liší tím, že se nejedná o CRUD. Pro nastavení je vytvořena pouze jedna stránka, na které je po otevření zobrazen přiřazený formulář.

Přidat do menu
: Určuje zda má být pro obsahový typ automaticky vygenerovaná položka v menu.

Přidat routy
: Určuje zda mají být pro obsahový typ automaticky vygenerovány routy vedoucí na controller daný jeho názvem.

Seznam sloupců
: Seznam sloupců oddělených čárkou určuje, jaké informace budou zobrazeny na výpisové stránce. Jako sloupec lze použít cokoliv, co je možné z modelu získat jako atribut. V případě, že model využívá meta atributy anebo překlady, lze využít i názvy meta atributů popř. jednotlivé atributy z překladu. Ve výchozím stavu se CMS snaží ze všech uvedených sloupců vytvořit odkazy, pomocí nichž lze data na výpise řadit. Pokud si uživatel nepřeje, aby šlo podle uvedeného sloupce řadit, stačí před jeho název přidat vykřičník.

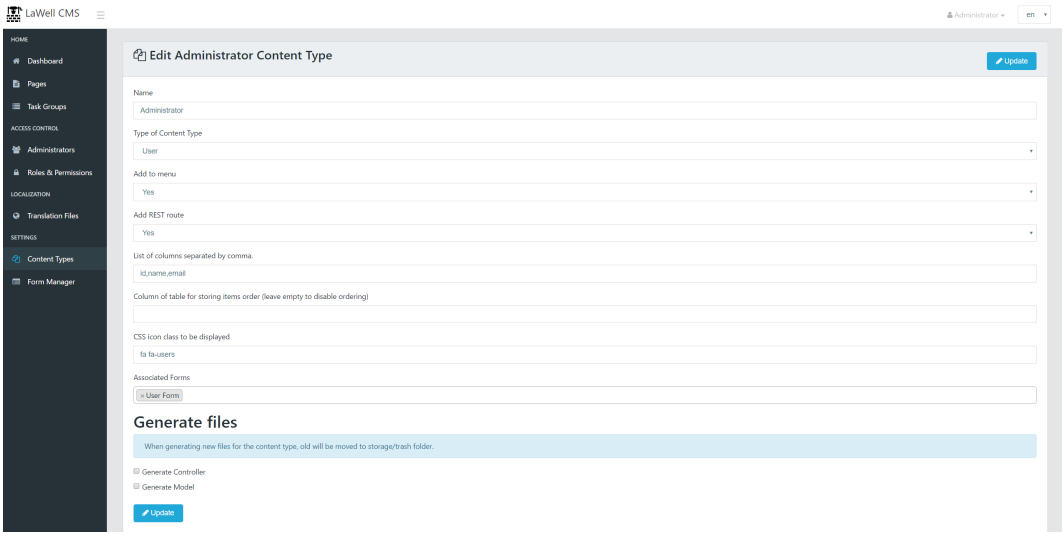
Název sloupce pro řazení
: Pokud si uživatel přeje, aby bylo povoleno řazení na výpisové stránce, stačí napsat název sloupce, do kterého se má pořadí ukládat. CMS poté na výpisovou stránku automaticky přidá možnost drag'n'drop řazení jednotlivých položek.

Přiřazené formuláře
: Seznam formulářů přiřazených k obsahovému typu. Všechny zde vybrané formuláře se budou zobrazovat při vytváření / úpravě jednotky obsahového typu. CMS se je poté bude snažit co nejlépe zpracovat (viz formuláře).

Generování controlleru a modelu

Po zakliknutí generování controlleru anebo modelu se vygenerují nové obslužné třídy. Této možnosti lze například využít po přejmenování obsahového typu, aby názvy tříd korespondovaly s názvem obsahového typu. Uživatel tímto způsobem ale dočasně ztrácí původní třídy a v nich provedené změny. Provedené změny nejsou ztracené úplně, protože před vygenerováním nové třídy je ta stará uložena ve složce *storage/trash*. Stejné pravidlo platí, pokud dojde ke smazání obsahového typu z administrace. Cestu, kam budou staré třídy přesouvány, lze definovat v nastavení CMS.

Obrázek 3 ukazuje editační stránku pro uživatelský obsahový typ.



Obrázek 3: Editace obsahového typu s názvem Administrator

3.7.2 Formuláře a pole obecně

Stavebním kamenem správy obsahu tohoto CMS jsou právě formuláře. Uživateli je umožněno tvořit formuláře, které následně budou zobrazeny u obsahových typů, ke kterým jsou přiřazeny. Každý formulář se skládá z názvu, který je určen čistě pro popis daného formuláře, a různých druhů polí. Celá struktura polí byla navržena tak, aby uživatel měl možnost kdykoliv dodefinovat pole vlastní (viz Customizace CMS).

Hlavním nastavením každého pole je jeho název a typ. Název u pole určuje, kam bude pole ukládáno. Typ určuje, jak bude pole vypadat, a také jakým způsobem bude vyplněná hodnota v něm uložena. Dále lze u polí nastavit například jejich popisek, zda mají být multijazyčné nebo jejich rozvržení pomocí sloupcového layoutu.

Většina polí má ještě další unikátní možnosti, kterými lze nastavit chování nebo zobrazení pole. Tyto možnosti mohou být jednojazyčné a nebo pro každý

jazyk unikátní. Pokud se jedná o multijazyčné nastavení, je toto nastavení zobrazeno pro každý jazyk zadaný v nastavení CMS.

3.7.3 Základní pole

Mezi základní pole řadím všechna obsahová pole a pole s možností volby.

Text input a textarea
: Tato pole jsou v celku přímočará a není u nich nic, co by si vyžadovalo zvláštní pozornost. Snad jen možnost u textarea zvolit počet řádků, které budou zobrazeny. Počet sloupců je vynechán schválně, protože se pole textarea automaticky roztahuje na celou šířku obrazovky.

Pole pro heslo
: Pole pro heslo je unikátní v tom, že před uložením je jeho hodnota zašifrovaná. Dále pokud do tohoto pole nebude vložena žádná hodnota, tak se ani neuloží - tzn. v databázi zůstane uložena předchozí hodnota. Poslední unikátnost spočívá v měření síly hesla, které se objeví, jakmile uživatel začne do pole psát.

Pole s možností volby
: Do této kategorie spadají všechna pole, u kterých uživatel nemůže vytvořit novou hodnotu, pouze má na výběr z předvyplněných hodnot. Jedná se tak o pole radio input, checkbox input a select. Všechna tato pole mají možnost definovat seznam svých hodnot dvěma způsoby.

- Zadáním dvou textů, od sebe oddělených dvojtečkou. První text bude použit jako hodnota, která bude následně uložena. Druhý text slouží jako popisek, který uvidí uživatel při vyplňování formuláře.
- Zadáním jednoho souvislého textu bez dvojtečky. V tomto případě se celý text použije pro hodnotu i popisek.

Pro oba dva způsoby platí, že se na každý řádek zadává pouze jedna hodnota, vytvořením nového řádku vzniká nová hodnota.

Repeater
: Repeater je speciální druh pole, který umožňuje definovat další pole v rámci svého nastavení. Ve formuláři se pak zobrazí skupina polí, které může uživatel libovolně přidávat, odebírat a řadit. Repeater ještě obsahuje další nastavení, které umožňuje ovlivňovat vzhled a chování pole. Prvním nastavením je možnost sklápěcí hlavičky, která se vyplatí, pokud je v repeateru definované větší množství polí. Dalším nastavením je možné zapnout nebo vypnout možnost řazení jednotlivých skupin polí. Poslední dvě nastavení slouží pro určení maximálního anebo minimálního počtu skupin polí, které mohou existovat. Po nastavení těchto hodnot pak repeater automaticky

kontroluje počet skupin polí a nedovolí uživateli přidávat další přes vyplněné maximum popř. nepovolí ubírat skupiny polí pod nastavené minimum. Pokud uživatel nechce žádné omezení, stačí vyplnit do obou polí nuly.

Hodnoty repeateru by měli být uloženy jako pole. Hodnoty tak lze jednoduše ukládat do meta atributů, které umí automaticky ukládat a načítat celé pole. Pokud by uživatel chtěl repeater ukládat do pole ve vlastní tabulce, je nutné, aby se sám postaral o jeho správné uložení a načtení.

3.7.4 Pole pro vzhled formuláře

Tato pole nehrají žádnou úlohu při správě obsahu, ale pouze pomáhají formátovat vzhled výsledného formuláře nebo zobrazují dodatečné informace. Tato kategorie obsahuje tři druhy pole.

Tab
: Tab jako jediný pomáhá s formátováním formuláře a rozděluje jej do záložek. Aby se jednotlivé záložky zobrazily stačí toto pole přidat nad skupinu polí, které mají být v dané záložce. Ve formuláři tak může být několik těchto polí, které vždy uvozují novou záložku.

Nadpis
: Zobrazuje pomocný nadpis, kterému lze definovat jeho velikost.

Zpráva
: Zobrazuje pomocnou zprávu, ve které lze používat html tagy. Dále lze zprávě definovat jeden z pěti přednastavených stylů.

3.7.5 Pole pro správu vztahů

Pole pro správu vztahů umožňují spravovat vztahy mezi jednotlivými modely. Celkově existují čtyři různé druhy polí, z nichž tři pole spravují vztahy BelongsTo a BelongsToMany a poslední pole spravuje vztahy HasOne a HasMany. U všech vztahů, kromě BelongsTo, platí, že pro správné uložení vztahu musí mít model vztah definovaný. Jelikož vztah BelongsTo lze uložit pouze pomocí hodnoty cizího klíče v tabulce, není definice vztahu nutná. Pokud požadovaný vztah nebude definován, dojde při ukládání k chybě, při které se CMS snaží napovědět, jaký vztah by měl pro správné fungování vzniknout.

BelongsTo a BelongsToMany
: Pro správu vztahů BelongsTo a BelongsToMany slouží následující pole: vztah s tabulkou, vztah s obsahovým typem a vztah s tabulkou obsahových typů. Všechna tato pole vychází z pole select, a proto u nich můžeme najít podobná nastavení. Pole obsahují také přepínač určující, zda má být dovolen výběr více hodnot. Právě tímto přepínačem se určuje, zda jde o správu vztahu BelongsTo nebo vztahu BelongsToMany.

Pole vztah s tabulkou umožňuje vytvořit vztah s jinou tabulkou. Po definování konkrétní tabulky musí uživatel ještě definovat cizí klíč a hodnotu, která bude zobrazena uživateli.

Vztah s obsahovým typem umožňuje vytvářet vztahy s ostatními obsahovými typy. Obsahuje stejná nastavení jako pole pro vztah s tabulkou až na to, že se jako cíl vztahu definuje obsahový typ a nikoliv tabulka. Pole je vhodné, pokud potřebujeme vytvořit vztah s tabulkou, ve které se nachází více obsahových typů - jako je tabulka users nebo content.

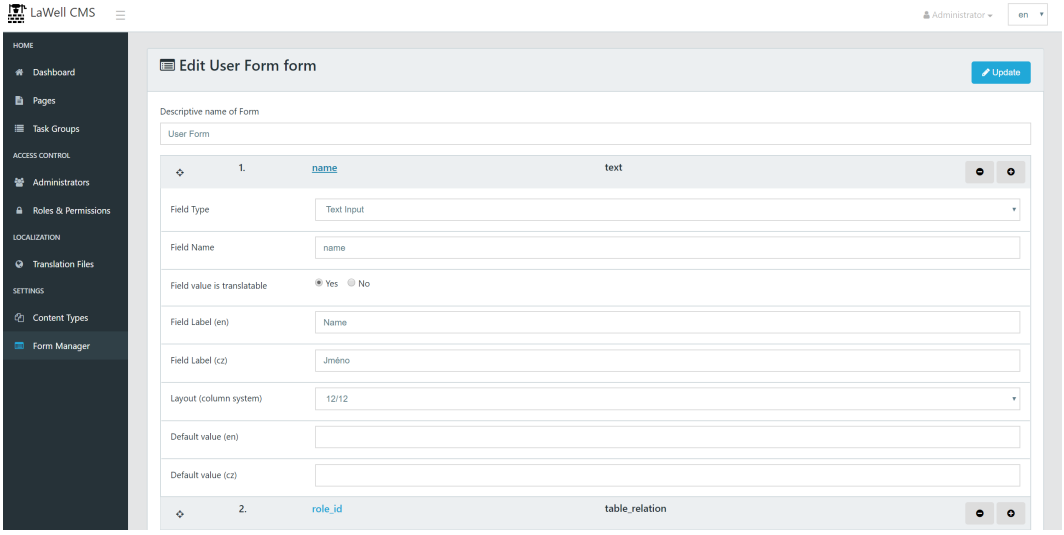
Posledním polem je vztah s tabulkou obsahových typů. Toto pole nám umožňuje výběr obsahových typů podle konkrétního typu (základní obsah, uživatel, vlastní tabulka, nastavení). Díky tomuto poli lze například přesouvat uživatele mezi jednotlivými uživatelskými typy.

HasOne a HasMany

Kdybychom v administraci měli používat pouze pole spravující vztahy BelongsTo a BelongsToMany, vznikla by sice funkční správa obsahu, která by ale byla velmi rozkouskována. Vznikla tedy myšlenka možnosti spravovat i relace HasOne a HasMany pomocí vnořených formulářů. Tímto způsobem pak dojde k ucelení celé administrace, kdy v jediném bloku lze spravovat více spolu souvisejících tabulek. O tuto funkcionalitu se stará pole inline vztahů. Vztah se definuje vždy na úrovni obsahových typů. Vnořený formulář vzniká z formulářů přiřazených k obsahovému typu, se kterým se má vztah navázat.

Toto pole, stejně jako předchozí, obsahuje přepínač pro určení, zda má být dovolen výběr více hodnot. Tento přepínač určuje, zda se jedná o správu vztahu HasOne nebo HasMany. Při správě vztahu HasOne se jedná o klasický formulář. Při správě vztahu HasMany se jedná o repeater. Právě proto je možné pro vztah HasMany nastavit podobná nastavení, jako pro repeater. Jedinou změnou je nastavení řazení. V případě pole inline vztahů už se řazení nezapíná přepínačem, ale vyplněním názvu atributu tabulky, do kterého má být pořadí uloženo. Pokud uživatel název atributu nevyplní, bude řazení zakázáno.

Na obrázku 4 lze vidět administrační stránku formuláře.



Obrázek 4: Editace formuláře pro správu uživatelů

4 Customizace CMS

Další důležitou součásti vytvořeného CMS je customizace celého řešení. Pro customizaci jsou hlavně určeny vygenerované třídy, domovská stránka administrace, formulářová pole a vzhled. Pokud by ale uživatel chtěl může pomocí dědění, vlastních rout a přepsání položek menu modifikovat jakoukoliv část administrace.

4.1 Vzhled

Jelikož byl celý systém pro správu obsahu vyvíjen jako knihovna pro Laravel, je možné jednoduchým způsobem modifikovat vzhled jakékoliv části. Laravel totiž umožňuje pro knihovny definovat dvoje umístění šablon. Prvním místem je složka v samotné knihovně a druhým místem je složka s názvem knihovny umístěna ve složce aplikace *views/vendor*. Při vykreslování jednotlivých šablon se pak Laravel nejdříve dívá do složky aplikace a až pak do složky knihovny. První nalezená šablona bude použita pro výpis.

Uživateli tak stačí pro úpravu vzhledu zkopírovat potřebné soubory z knihovny do složky *views/vendor/lawell*.

4.2 Domovská stránka administrace

Většina CMS obsahuje nějakou domovskou stránku, v angličtině nazývanou dashboard, kde jsou zobrazeny základní informace a události, které se v systému udály. Není tomu jinak ani v systému vytvořeném v rámci této bakalářské práce.

Pro správu jednotlivých položek zobrazených na domovské stránce, byla vytvořena třída *LaWell\Core\Dashboard\DashboardManager*. Tato třída je v service containeru zaregistrována jako singleton (service container nám pokaždé

vrátí stejnou instanci třídy) a umožňuje nám přidávat popř. odebírat jednotlivé položky. Třída by měla být použita v rámci service providerů a lze ji ze service containeru získat pomocí plného názvu nebo pomocí zkratky *dashboardManager*.

Pro vytvoření nové položky je potřeba vytvořit novou třídu, která bude dědit ze třídy *LaWell\Core\Dashboard\DashboardItem*. V této třídě je pak nutné implementovat jedinou metodu, která se jmenuje `getViewName`. Tato metoda musí vracet název šablony, která má být pro položku použita. Druhou důležitou metodou je metoda `prepareDataForRender`, která musí vracet asociativní pole s hodnotami, které potřebujeme mít v šabloně dostupné.

```
1  <?php
2  namespace App\Dashboard;
3
4
5  use App\User;
6  use LaWell\Core\Dashboard\DashboardItem;
7
8  class NewUsersDashboard extends DashboardItem
9  {
10
11     public function getViewName()
12     {
13        return 'admin.dashboard.new-users';
14     }
15
16     public function prepareDataForRender()
17     {
18        $users = User::orderBy('id', 'desc')
19           ->take(15)
20           ->get();
21
22        return ['users' => $users];
23     }
24 }
```

Zdrojový kód 6: Příklad nové položky pro domovskou stránku

4.3 Formulářová pole

Postup vytváření nebo modifikace formulářových polí je podobný jako u domovské stránky administrace. Opět vznikla třída, která má za úkol spravovat jednotlivá pole. Třída se jmenuje *LaWell\Core\FormFields\FieldManager* a měla by se používat pouze v rámci service providerů a lze ji získat ze service containeru také pomocí zkratky *fieldManager*.

Vytvoření nového formulářového pole je poměrně složité, proto tento postup rozděluji do několika podkapitol. Je dobré mít na paměti, že stejným způsobem jsou vytvořeny všechna ostatní formulářová pole. Pokud tedy bude mít uživatel

```php
<?php

namespace App\Providers;

use App\Dashboard\NewUsersDashboard;
use Illuminate\Support\ServiceProvider;

class DashboardServiceProvider extends ServiceProvider
{
   public function boot()
   {
      $dashboardManager = app('dashboardManager');
      $dashboardManager->add(new NewUsersDashboard());
   }
}
```

Zdrojový kód 7: Příklad registrace položky pro domovskou stránku

problém při vytváření nového pole, mělo by mu stačit podívat se, jakým způsobem jsou implementována pole základní.

4.3.1 Nové pole a jeho nastavení

Pro vytvoření nového formulářového pole je potřeba vytvořit novou třídu, která bude dědit ze třídy *LaWell\Core\FormFields\FieldType* nebo z některého z jejich potomků. Aby šlo vytvořené pole použít je nutné implementovat tři metody.

1. Metoda `type()` musí vracet unikátní označení pro pole (např. repeater, select, inline_relation, ...).

2. Metoda `category()` musí vracet označení kategorie, pod kterou pole spadá. V systému jsou k dispozici celkem tři kategorie: content, choices a relations. Uživatel si ale může bez problémů vytvořit svou vlastní kategorii.

3. Metoda `getViewName()` musí vracet název šablony, kterou bude pole využívat.

Dále je možné v poli nastavit několik třídních proměnných, které ovlivňují jeho chování a zobrazení v rámci tvorby formulářů. Proměnná `$title` určuje název, který bude zobrazen uživateli ve výběru polí. Pokud tato proměnná nebude zadána, bude název automaticky vygenerován z návratové hodnoty metody `type()`. Proměnná `$translatable` určuje, zda je pole překladatelné. Poslední proměnná `$repeatable` určuje, jestli může být pole umístěno v repeateru.

Kromě těchto základních nastavení má každé pole možnost definovat si své vlastní nastavení, které bude automaticky uloženo. Děděním z výše uvedené třídy získá uživatel možnost využívat několik metod, pro definici tohoto nastavení.

Jednotlivé metody umožňují přidávat textová pole, zaškrtávací pole, přepínače i rozbalovací pole.

```
1  <?php
2     public function initializeOptions()
3     {
4        parent::initializeOptions();
5
6        $this->addRadioOption('collapsible', ['0' => 'No', '1' => 'Yes
             '], 'Collapsible items with header', '1');
7        $this->addRadioOption('ordering', ['0' => 'No', '1' => 'Yes'],
              'Allow ordering of the items', '1');
8        $this->addTextOption('minimum', 'Minimum number of items (0
             for no minimum)', '0');
9        $this->addTextOption('maximum', 'Maximum number of items (0
             for no maximum)', '0');
10    }
```

Zdrojový kód 8: Definice nastavení repeateru

4.3.2 Příprava výpisu pole

Pokud uživatel potřebuje před výpisem pole provádět složitější operace, je možné pro to využít metodu `prepareDataForRender()`, která musí vracet asociativní pole s hodnotami, které jsou poté předány šabloně pole.

4.3.3 Uložení dat

Pro uložení dat je možné využít metod `onBeforeSave()` a `onAfterSave()`. První metoda se využívá v případech, kdy potřebujeme nastavit atributy modelu ještě před jeho uložením. Typicky se tak jedná o atributy tabulky. Druhá metoda je volána až po uložení modelu a používá se například při ukládání vztahů BelongsToMany, HasOne a HasMany.

Obou metodám jsou předány všechny hodnoty formuláře a je na vytvořeném poli, aby vybralo tu svou a následně ji uložilo. Metodám je samozřejmě také předáváno nastavení, které uživatel vyplnil při tvorbě formuláře. Dále je metodám předáván samotný model, u kterého má dojít k uložení dat.

4.3.4 Vyplnění pole existující hodnotou

Posledním krokem při tvorbě nového pole je vyplnění pole existující hodnotou. Pokud uživatel edituje existující položku obsahového typu, je potřeba, aby mu byla zobrazena dříve vyplněná a uložená hodnota. Pro tuto situaci je volána metoda s názvem `populateField()`, které jsou předány informace o nastavení pole, editovaném modelu a jazycích. V této metodě má formulářové pole za úkol

vytáhnout si z editovaného modelu potřebnou hodnotu a dočasně si ji uložit v předaném nastavení pomocí metody `setValue()`.

Takto nastavenou hodnotu pak bude pole mít k dispozici ve své šabloně.

4.4 Vygenerované třídy

Vygenerované třídy vždy dědí z tříd, které uživateli přináší další funkcionalitu. Jak již bylo řečeno, tak vše je pouze v uživatelových rukou a když si bude přát, tak může přepsat jakoukoliv zděděnou metodu nebo proměnnou. Co se týče vygenerovaných modelů, tak tam už byla většina funkcionality popsána výše v kapitolách o meta atributech a správě překladů. Další metody modelu slouží hlavně pro zpracování a ukládání dat z formulářů.

Z pohledu customizace jsou zajímavější vygenerované controllery a to konkrétně controllery týkající se CRUD (obsahové typy základního obsahu, uživatelů a vlastních tabulek). Níže uvádím některé ze základních úprav, se kterými by se mohl uživatel využívající toto CMS setkat.

4.4.1 Validace formuláře

Asi nejčastější úprava controlleru, kterou bude uživatel provádět je definice validačních pravidel. Aby uživatel nemusel upravovat celé metody, které se starají o ukládání nové položky nebo o aktualizaci již existující, je možné definovat několik nových metod. Všechny tyto metody pak musí vracet pole validačních pravidel, jak je popisuje Laravel.

První metoda nese název `getStoreRules()` a slouží pro definici pravidel pro ukládání nové položky. Druhá metoda se jmenuje `getUpdateRules()` a slouží pro definic pravidel při ukládání již existující položky. Poslední metodou je metoda `getRules()`, která slouží pro definici pravidel pro ukládání nové i existující položky. První dvě metody mají vyšší prioritu a pokud jsou definovány, tak metoda `getRules()` není vůbec volána.

Obrázek 5 ukazuje odeslání formuláře s validací emailové adresy.

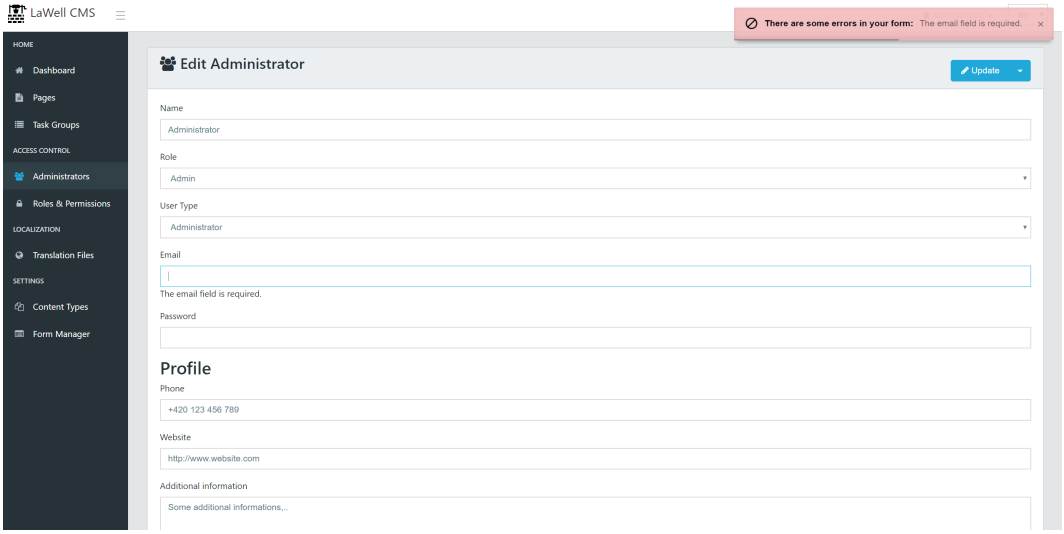
4.4.2 Úprava prováděného dotazu na stránce výpisu

Vytvořené CMS neumí automaticky spojovat model s jeho vztahy. Pokud je tedy potřeba vypsat nějaké informace založené na vztahu s jiným modelem, musí uživatel nejprve modifikovat dotaz, který se provádí pro výběr dat na stránce výpisu. Typicky se tato modifikace týká vytvoření spojení s další tabulkou.

Pro modifikaci dotazu lze použít metodu `eagerLoadRelations()`, které je jako parametr předán objekt reprezentující rozpracovaný dotaz.

4.4.3 Změna šablony

Pro změnu šablon, které jsou použity pro výpis, tvorbu nové položky a editaci, stačí pouze změnit obsah zděděných třídních proměnných. Jednotlivé proměnné



Obrázek 5: Odeslání formuláře s validací

`$indexView`, `$createView` a `$editView` obsahují vždy cestu k potřebné šabloně a je jen na uživateli, jestli použije šablony výchozí nebo definuje nové.

4.4.4 Úprava jednotlivých akcí

Jak již bylo zmíněno, může uživatel v případě nutnosti začít upravovat metody, které obstarávají jednotlivé akce. Názvy metod odpovídají konvencím Laravelu s jednou výjimkou, kterou tvoří nová metoda ukládající pořadí položek.

- `index` - výpis položek.
- `create` - zobrazení formuláře pro tvorbu nové položky.
- `store` - uložení nové položky.
- `edit` - zobrazení formuláře pro editaci existující položky.
- `update` - uložení existující položky.
- `destroy` - odstranění položky.
- `reorder` - uložení pořadí položek.

Závěr

Náplní bakalářské práce bylo implementovat správu obsahu za pomocí frameworku Laravel. Bakalářská práce měla dále umožňovat správu uživatelů a tvorbu multijazyčného obsahu. Všechny části byly implementovány a v průběhu vývoje řádně testovány a tedy splňují zadání.

V průběhu vývoje byl kladen důraz na správné strukturování zdrojového kódu, pro následnou údržbu a dodatečný vývoj nad rámec bakalářské práce. Dále při vývoji vznikla sada automatizovaných testů.

Text bakalářské práce slouží jako uživatelská a programátorská příručka. V textu jsou také popsány využité technologie a motivace.

Conclusions

The goal of the bachelor thesis was to implement content management system with Laravel framework. The bachelor thesis should also allow user management and creation of multilingual content. All parts have been implemented and properly tested during the development and therefore are in accordance with the assignment.

During development, the emphasis was put on the correct structure of the source code, for subsequent maintenance and additional development beyond the bachelor thesis. In addition, a set of automated tests has been developed.

The text of the bachelor thesis is used as a user and programming manual. The technology and motivation are also described in the text.

A Zprovoznění aplikace

A.1 Potřebné požadavky

Aplikace slouží jako systém pro správu webového obsahu, proto je potřeba zajistit spuštění serveru, na kterém aplikace poběží.

Požadavky na server jsou:

- Apache server.
- MySQL databáze (bakalářská práce byla vyvíjena na databázovém serveru MariaDB 10.1.9).
- PHP 7.1.
- Splnění požadavků pro provoz frameworku Laravel, které lze nalézt na stránce https://laravel.com/docs/5.4/installation#server-requirements.

A.2 Spuštění aplikace

Aplikaci lze nainstalovat a spustit podle návodu uvedeného v této práci. Pro rychlejší instalaci je ale také možné použít data z přiloženého CD.

Postup je následující:

1. Z přiloženého CD zkopírujeme obsah složky *app/demo* do kořenové složky našeho serveru.
2. Na serveru vytvoříme novou databázi.
3. Do této databáze vložíme z přiloženého CD obsah souboru *demo.sql*, který je umístěn ve složce *data*.
4. V kořenové složce zkopírované aplikace otevřeme soubor *.env*.
5. V otevřeném souboru nastavíme informace pro připojení k databázi.
6. Ve webovém prohlížeči otevřeme URL adresu našeho serveru a k adrese přidáme */admin*.
7. Do pole pod názvem *E-mail address* napíšeme text *admin@site.com*.
8. Do pole pod názvem *Password* napíšeme text *123456*.
9. Kliknutím na tlačítko *Login* se přihlásíme do webové aplikace.

B Obsah přiloženého CD

Přiložené CD se skládá ze tří základních složek, ve kterých nalezneme zdrojové kódy aplikace a dokumentaci.

Struktura CD je následující:

`app/`
: Tato složka je dále rozdělena na dvě podsložky *app/demo* a *app/core*. Složka *app/demo* obsahuje implementovanou aplikaci včetně frameworku Laravel a dalších potřebných knihoven. Složka *app/core* obsahuje pouze zdrojové kódy bakalářské práce.

`doc/`
: Složka obsahuje text bakalářské práce ve formátu PDF. Ve složce se také nachází všechny soubory, které jsou potřebné pro vytvoření tohoto souboru.

`data/`
: Složka obsahuje soubor *demo.sql*, ve kterém se nachází předem připravená data pro databázi, která jsou nutná pro instalaci aplikace z tohoto CD.

`readme.txt`
: Soubor popisuje minimální požadavky aplikace a také instrukce pro nasazení aplikace na server.

Literatura

[1] CARBONNELLE, Pierre. PYPL PopularitY of Programming Language. GitHub [online]. [cit. 2017-04-08]. Dostupné z: http://pypl.github.io/PYPL.html

[2] THE PHP GROUP. History of PHP. PHP [online]. [cit. 2017-04-08]. Dostupné z: http://php.net/manual/en/history.php.php

[3] Q-SUCCESS. Usage statistics and market share of PHP for websites. W3Techs [online]. [cit. 2017-04-08]. Dostupné z: https://w3techs.com/technologies/details/pl-php/all/all

[4] FAULDS Andrea; SURASKI Zeev. PHP RFC: Name of Next Release of PHP. PHP [online]. [cit. 2017-04-08]. Dostupné z: https://wiki.php.net/rfc/php6

[5] ZEND TECHNOLOGIES LTD. Get performance insight into the upcoming release of PHP 7. Zend [online]. [cit. 2017-04-08]. Dostupné z: https://www.zend.com/en/resources/php7_infographic

[6] THE PHP GROUP. Migrating from PHP 5.6.x to PHP 7.0.x - New Features. PHP [online]. [cit. 2017-04-08]. Dostupné z: http://php.net/manual/en/migration70.new-features.php

[7] COMPOSER. Introduction - Composer. GetComposer [online]. [cit. 2017-04-08]. Dostupné z: https://getcomposer.org/doc/00-intro.md

[8] KŘÍŽKA Jakub. Sémantické verzování 2.0.0. SemVer [online]. [cit. 2017-04-08]. Dostupné z: http://semver.org/lang/cs/

[9] CROFT Jeff. What is a framework? Alistapart [online]. [cit. 2017-04-08]. Dostupné z: https://alistapart.com/article/frameworksfordesigners#section1

[10] SKVORC Bruno. The Best PHP Framework for 2015: SitePoint Survey Results. Sitepoint [online]. [cit. 2017-04-08]. Dostupné z: https://www.sitepoint.com/best-php-framework-2015-sitepoint-survey-results/

[11] CODECEPTION. Introduction. Codeception [online]. [cit. 2017-04-08]. Dostupné z: http://codeception.com/docs/01-Introduction

[12] NPM. What is npm? NPMJS [online]. [cit. 2017-04-08]. Dostupné z: https://docs.npmjs.com/getting-started/what-is-npm

[13] BABEL. Babel - The compiler for writing next generation JavaScript. BabelJS [online]. [cit. 2017-04-08]. Dostupné z: http://babeljs.io/

[14] BOOTSTRAP. Bootstrap - The world's most popular mobile-first and responsive front-end framework. GetBootstrap [online]. [cit. 2017-04-08]. Dostupné z: https://getbootstrap.com/

[15] JQUERY. What is jQuery? jQuery [online]. [cit. 2017-04-08]. Dostupné z: https://jquery.com/

[16] WEBPACK. webpack - module bundler. GitHub [online]. [cit. 2017-04-08]. Dostupné z: https://webpack.github.io/

[17] VUEJS. What is Vue.js? Vuejs [online]. [cit. 2017-04-08]. Dostupné z: https://vuejs.org/v2/guide/

[18] YOU Evan. Vue in 2016. medium [online]. [cit. 2017-04-08]. Dostupné z: https://medium.com/the-vue-point/vue-in-2016-8df71d98bfb3