

Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky



Studijní program: Aplikovaná informatika

Obor: Kognitivní informatika

Návrh a vývoj frameworku pro vývoj webových aplikací administrativního typu

DIPLOMOVÁ PRÁCE

Student : Bc. David Feldstein

Vedoucí : Ing. PhDr. Antonín Pavlíček, Ph.D.

Oponent : Ing. Tomáš Fried

Praha, Listopad 2019

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a že jsem uvedl všechny použité prameny a literaturu, ze které jsem čerpal.

V Praze dne 28. listopadu 2019

.....

David Feldstein

Poděkování

Rád bych poděkoval panu Ing. PhDr. Antonínu Pavlíčkovi, Ph.D. za věcné připomínky a vstřícnost při konzultacích a vypracování diplomové práce.

Abstrakt

Cílem práce je vytvořit framework, který by sloužil k vývoji webových aplikací administrativního typu. Co je myšleno webovou aplikací administrativního typu je vysvětleno v úvodu práce. Hlavní myšlenkou frameworku je **nalezení rovnováhy mezi usnadněním práce vývojáře a limitováním jeho možností**. Další zásadou frameworku je stavět na něčem, co je známé, aby se **vývojářská komunita ve frameworku snadno zorientovala**. K tomuto účelu poslouží framework **Laravel**, který je nejpoužívanějším frameworkem mezi webovými vývojáři.

Klíčová slova

Návrh a vývoj, Framework, Administrativní aplikace, Webdesign, Laravel

Abstract

The objective of the thesis is to create a framework for designing a web based administrative software. What is meant by the web based administrative software is explained in the introduction of the thesis. The main idea behind the framework is **finding a balance between simplifying the work of the software developer and limiting his possibilities**. Another base principle is to build upon something that is well known to **allow developers community to easily orientate in the framework**. To this cause the **Laravel** framework will be used since it is most commonly used among the web developers.

Klíčová slova

Designing and Implementation, Framework, Administrative software, Webdesign, Laravel

Obsah

1	Úvod	7
1.1	Důvody vývoje a využití frameworku v praxi	8
1.2	Jak na to?	8
1.3	Hlavní výzvy	9
2	Popis nástroje	10
2.1	Princip fungování.....	10
2.2	Popis architektury nástroje.....	12
2.2.1	Laravel	12
2.2.2	Koncept MVC.....	13
2.2.3	Náčrt konceptu frameworku AnyAdmin	13
3	Systém definičních souborů	15
3.1	Dílčí datové objekty definičních souborů	15
3.1.1	Položka menu (_menu_item).....	15
3.1.2	Uživatelské omezení (_access).....	17
3.1.3	Tlačítko (_button).....	17
3.1.4	Akce (_action)	18
3.1.5	Sekce vstupního formuláře (_section)	20
3.1.6	Datový vstup (_input)	21
3.2	Hlavní definiční soubor app.json.....	26
3.3	Definiční soubory entit	27
3.4	Definiční soubory stránek	29
4	Entity	31
4.1	Základní popis entity	31
4.2	Vztahy mezi entitami.....	32
5	Stránky.....	33
5.1	Funkce stránek.....	34
6	Komponenty	35
6.1	Složení komponenty	35
6.2	Komunikace s aplikací	35
6.3	Přednastavené základní komponenty	36
6.3.1	Komponenta tabulky (table).....	36
6.3.2	Komponenta formuláře (form)	38
6.3.3	Komponenta detailu (detail).....	39
6.4	Vlastní komponenty definované vývojářem	41
6.4.1	Postup vytvoření nové komponenty	41

7	Vývojářem definované funkce.....	43
7.1	Globální funkce	43
7.2	Funkce entit.....	43
7.3	Komunikace s Front-endovým controllerem	44
7.4	Interakce s uživatelem	46
8	Front-endový controller.....	48
8.1	Komunikace s back-endovou částí aplikace	48
9	Generátor a databáze.....	49
9.1	Princip funkce generátoru.....	49
9.2	Modelová situace: Hlídní struktury a integrity	50
9.3	Pevné části databázových tabulek entit.....	52
9.4	Relace	53
10	Uživatelské prostředí	54
10.1	Základní design.....	54
10.2	Responzivita.....	55
10.3	Jazykové mutace	55
11	Využité technologie	56
11.1	Back-endové jazyky a nástroje	57
11.1.1	Programovací jazyky a databáze	57
11.1.2	Frameworky	58
11.1.3	Pluginy	58
11.2	Front-endové jazyky a nástroje	59
11.2.1	Programovací jazyky	59
11.2.2	Frameworky a knihovny	60
11.2.3	Pluginy	61
11.3	Podpůrné technologie	67
11.3.1	Kompilovatelné extenze programovacích jazyků.....	67
11.3.2	Verzovací programy.....	68
11.3.3	Dependency Managers	68
12	Případová studie	69
12.1	Zadání	69
12.1.1	Základní entity	69
12.1.2	Vztahy mezi entitami	70
12.2	Implementace	70
12.2.1	Konfigurace aplikace	70
12.2.2	Nadefinování jednotlivých stránek	71
12.2.3	Nadefinování jednotlivých entit a jejich vztahů	71
12.2.4	Nastavení databáze.....	74

12.2.5 Výsledek	75
12.3 Závěr	77
13 Živá ukázka aplikace	78
14 SW nároky a zprovoznění frameworku	79
14.1 SW nároky	79
14.1.1 Operační systém	79
14.1.2 Server	79
14.1.3 Git	79
14.1.4 Dependency managers	80
14.2 Postup zprovoznění frameworku (instalace)	80
15 Závěr	82
Seznam literatury	83
Terminologický slovník	85
Seznam obrázků a výpisů	86
Seznam obrázků	86
Seznam výpisů z programu	87

1 Úvod

Hlavním cílem práce je vytvořit framework pomocí jehož by programátor mohl rychle a efektivně vytvořit jakoukoli **webovou aplikaci administrativního typu**. Nejprve je důležité si pojmenovat, co takovou webovou aplikaci administrativního typu rozumíme.

Slovem „**webová**“ míníme, že aplikace je postavena na webové bázi. To má celou řadu výhod. Jelikož se jedná o webové řešení, data i samotnou aplikaci lze sdílet prakticky na jakémkoli zařízení, které umožňuje zobrazit webový obsah, což je v praxi většina zařízení, se kterými administrativní pracovníci pracují. Řadíme sem mobilní telefony, tablety, laptopy či stolní počítače.

Slovem „**administrativní**“ rozumíme aplikace, které jsou zaměřeny na správu informací a práci s daty. Patří sem zobrazování, třídění, tvorba či editace dat. Jde tedy o korporátní systémy, které pomáhají firmám udržet si přehled, „udělat si pořádek“ ve svých procesech či nějaké části zautomatizovat. Příkladem takových aplikací jsou různé databáze klientů, fakturační systémy, CRM systémy, apod.

Framework, který vytváříme nazveme pracovním názvem AnyAdmin.



Obrázek 1: Logo AnyAdmin

1.1 Důvody vývoje a využití frameworku v praxi

Diplomové práci obětuji poměrně velkou část své pozornosti a času, proto se domnívám, že je dobré tuto energii vložit do něčeho, co neslouží pouze ke splnění mých studijních povinností, nýbrž do něčeho, co mohu později využít. Živím se jako vývojář na volné noze a chci vytvořit něco, co usnadní práci nejen mě, ale i jiným vývojářům. Framework jsem se rozhodl vytvořit z důvodu, že si všímám repetitivních úkolů, které musím jako vývojář webových aplikací neustále opakovat, a přitom se snažím vždy vytvořit v zásadě velmi podobnou věc. Většina firem tuto problematiku řeší tak, že jednoduše kopíruje staré projekty nebo jejich části, na jejichž základu staví projekty nové. Toto má za následek neustálé zanášení kódu a záplatování.

Proto jsem si položil otázku, zda není možné vývojářům poskytnout výchozí podobu aplikace a od ní se odrážet. V aplikacích administrativního typu nakonec všichni řeší to samé, ať už jde o přihlašování, uživatelská práva či zobrazování a editaci nějakých dat v různých formách. Víme, že existuje nespočet redakčních systémů a frameworků, nenašel jsem ale žádný, který by řešil přímo administrativní aplikace, na které se zaměřuji a ve kterých vidím budoucnost. Frameworky řeší buď její části nebo jsou pro rychlý vývoj příliš robustní.

1.2 Jak na to?

Chceme vývojáři poskytnout kvalitní nástroj, ale zároveň ho nechceme limitovat. V práci se tedy snažím právě o nalezení této rovnováhy mezi **usnadněním práce vývojáře a limitováním jeho možností**. Celá myšlenka za tím, jak toho docílit je v tom, že pro vývojáře jsou vytvářeny komponenty sloužící k řešení dílčích problémů, ale nikdy na ně vývojář není omezen. **Vývojář může kteroukoli komponentu přepsat nebo framework doplnit o novou.**

Framework při tom vychází z nejpopulárnějších nástrojů užívaných komunitou webových vývojářů, především pak z frameworku Laravel, v kódu by se tedy měl každý vývojář, který zná framework Laravel **snadno zorientovat**.

Dnešní internet vývojářům nabízí širokou škálu velmi užitečných pluginů a nástrojů na řešení dílčích problémů, jak v back-endové části, tak v části front-endové. Většina těchto pluginů a nástrojů je poskytována vývojářům zadarmo jako open source. Mým záměrem je právě tyto pluginy a nástroje poskládat dohromady a vytvořit tak perfektní prostředí pro rychlý a efektivní vývoj administrativních aplikací.

Vývoj firemních systémů je zpravidla realizován na zakázku a bývá drahý, jelikož se programuje od základu, proto si ho mohou dovolit zpravidla větší firmy. Věřím, že firemní systémy ale **nemusí být pouze výhradou velkých firem**. Zejména pak pro malé a střední podniky mohou být jejich vlastní administrační systémy velkou pomocí při zvládání procesů a nároků dnešní uspěchané doby. Přitom to, co potřebují je si velice podobné, jen pracují

s různými daty. V českém prostředí je stále co zlepšovat a digitalizace může přispět k odstranění chaosu a zpřehlednění procesů. Tedy k tomu, aby lidé pracovali efektivně a nezatěžovali se zbytečnými úkony, které se díky dnešním technologiím mohou snadno zautomatizovat.

1.3 Hlavní výzvy

Framework AnyAdmin se snaží najít rovnováhu mezi **usnadněním práce vývojáře a limitováním jeho možností**. Hlavními výzvami při tvorbě tohoto nástroje bude volba vhodných pluginů a správné propojení mezi nimi (tedy architektura). Jelikož neustále vznikají nové a nové formy pluginů a záleží také na jejich vhodném propojení, nelze je jednoduše zanalyzovat a porovnat a musí se dojít k určitým kompromisům. V tuto chvíli nemám lepší možnost než vycházet z vlastních zkušeností a ze zkušeností vývojářské komunity s těmito pluginy a nástroji. Práce se tedy nebude zaměřovat na analýzu a porovnávání možností, jde o prakticky zaměřenou práci s hlavním cílem aplikaci vytvořit, primární činností je tedy **vlastní programování**.

Jednoduše vycházím z předpokladu, že to, co je obecně vývojářskou komunitou hojně používané je nejvhodnější. V praxi tomu opravdu tak je, a to z prostého důvodu. I kdyby na dílčí problém existoval o něco lepší nástroj, je lepší využít ten, který používá většina vývojářské komunity. S nástrojem, který komunita zná totiž umí pracovat a **nemusí se učit nic nového**. Rozdíly mezi nástroji jsou minimální a rychlost, kterou jdou technologie dopředu je výrazně vyšší než doba, za kterou je vývojář schopen se s nástrojem seznámit. Neustále se objevují nové nástroje, tím pádem by vývojář nedělal nic jiného, než se učil.

2 Popis nástroje

Jak již bylo řečeno framework AnyAdmin se snaží najít rovnováhu mezi usnadněním práce vývojáře a limitováním jeho možností. Framework tedy přebírá veškerou práci s databází, zpracování datových vstupů a zobrazování. Ale zároveň ponechává vývojáři svobodu měnit toto chování, případně možnost doprogramovat chování nové.

Tím pádem je možné pomocí frameworku vytvořit **funkční plnohodnotnou aplikaci** včetně struktury databáze během **několika minut až hodin**, a to pouze za pomoci definičních souborů. Ale zároveň je možné jakékoli chybějící funkce doprogramovat či pozměnit stávající. Jak vytvořit takový jednoduchý fakturační systém během několika minut je ukázáno v případové studii v kapitole 12.

2.1 Princip fungování

Vycházíme z předpokladu, že v jednoduchosti je síla a že pokud nějaká základní komponenta je dostatečně jednoduchá a vše pojímající, dá se využít pro cokoli. Cílem tedy není vytvářet cílené funkční prvky, ale nechat prostor přirozené kolektivní inteligenci ať do nástroje vstoupí. Principem je tedy vymezit základ a **kdykoli se naskytne dílčí problém** k řešení, **není účelem ho ihned tvrdě ošetřit** podmínkou, ale **nechat k řešení prostor** člověku, tedy vývojáři, kterého lze, navzdory až podezřele stroji podobajícího se chování, za člověka stále pokládat.

V aplikacích administrativního typu jsou zažité principy, od kterých se můžeme odrazit. Tyto principy tedy budeme pokládat za základní stavební kameny:

- Aplikace je rozdělená do jednotlivých sekcí (má nějakou strukturu)
- Do aplikace je nutné se přihlásit
- Uživatelé v aplikaci mají na některé úkony práva, na jiné nikoli
- Uživatel s aplikací interaguje:
 - Zobrazuje data
 - Zapisuje data
 - Spouští událost (funkci)

Vše uvedené výše lze zapsat pomocí definičních souborů, implementaci již zařídí framework. Framework tedy má výchozí chování pro jakoukoli z těchto možností a v případě, že si vývojář přeje chování pozměnit, jednoduše daný prvek přepíše či aplikaci doplní o nový. Výchozí podoba aplikace po přihlášení je k nahlédnutí na obrázku č. 2 níže.



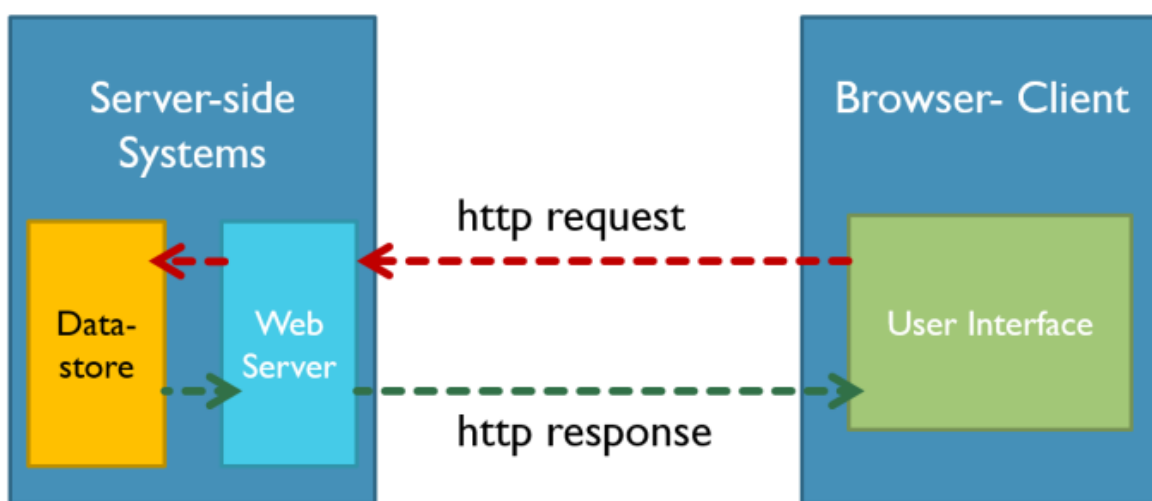
Obrázek 2: Rozložení prvků na stránce aplikace

Jak aplikace funguje si vysvětlíme z obrázku č. 2 výše. Po kliknutí na položku v menu se zobrazí nějaká **stránka**. Tato stránka má obsah, který se skládá z akčních tlačítek, položek v menu a prostoru pro komponentu (dynamicky generovaný pohled). Kliknutím na položky menu či na akční tlačítka se mění obsah v prostoru pro komponentu či se spustí nějaká akce. I dynamický obsah může obsahovat tlačítka, které spouštějí akci. Tím je **zajištěna univerzální komunikace s uživatelem**. V aplikaci přitom pracujeme s **entitami**, vývojářem definovanými datovými objekty. K práci s entitami pak slouží právě **komponenty**. V obsahu pro komponentu se může zobrazit cokoli: tabulka, formulář, faktura, detail produktu, graf, či cokoli jiného, co vývojář vytvoří. Výchozí jsou komponenty tabulky (pro zobrazení dat) a formuláře (nová instance entity, editace). Ostatní může definovat sám vývojář. Stejně tak může vývojář definovat jakoukoli funkci (např. vygenerování faktury podle zákazníka, rozeslání emailů atd.).

2.2 Popis architektury nástroje

2.2.1 Laravel

Framework, který vytvářím je nadstavbou frameworku Laravel, který tvoří jádro celého systému. Laravel je PHP framework, který řeší vše na úrovni webové komunikace, tedy zpracuje požadavek a vrátí výsledek. Princip požadavku (request) a odpovědi (response) je nastíněný na obrázku č. 3. Framework Laravel obsahuje také celou řadu nástrojů, od zpracování formulářů, validace datových vstupů až po odposlouchávače událostí. Představuje tedy ideální základnu, na které budeme stavět.

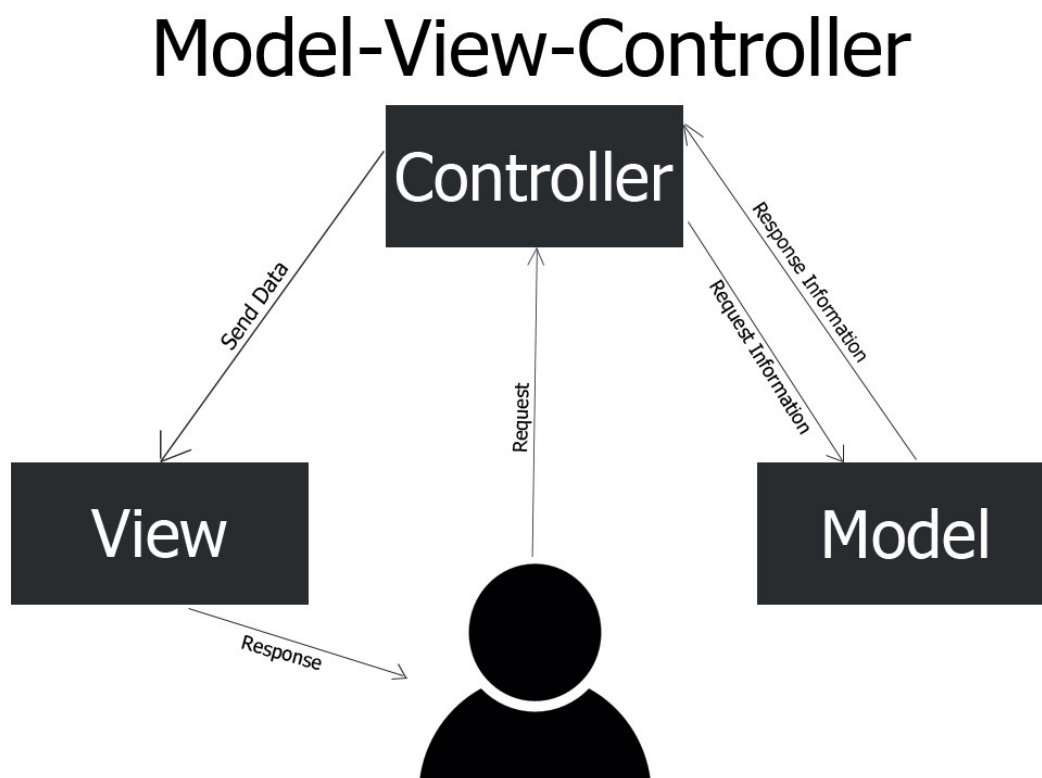


Obrázek 3: Request-Response model^[8]

Abychom porozuměli rozdílu mezi Laravelem a nadstavbou, kterou se snažím vytvořit je důležité říci, že i když je Laravel vynikající nástroj, který lze využít na širokou škálu aplikací i webových stránek, pro naše účely je stále příliš robustní, jelikož námi vytvářený framework se zaměřuje na vývoj velice konkrétního typu aplikací. Na Laravelu je možné postavit vše od webové stránky, přes webovou aplikaci až po API. V našem případě se snažíme o rapidní vývoj úzce zaměřeného typu aplikací. A těmito aplikacemi jsou právě aplikace administrativního typu.

2.2.2 Koncept MVC

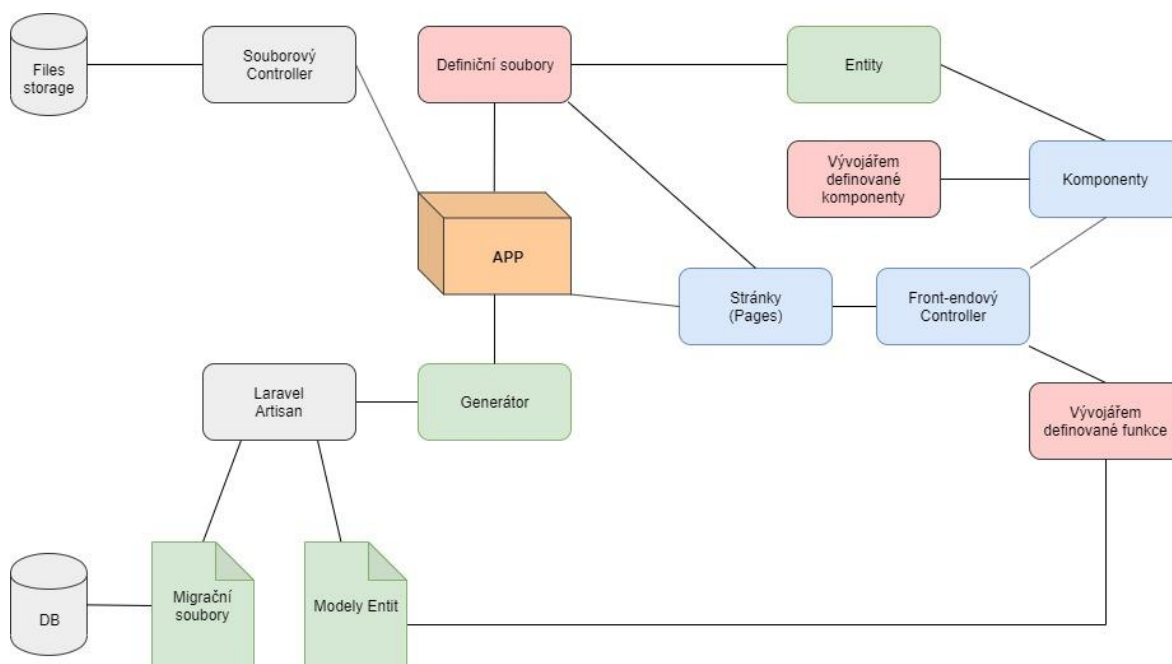
Základní architektura frameworku AnyAdmin na úrovni webového prohlížeče, řešící problém požadavku a odpovědi sleduje klasickou strukturu MVC (Model-View-Controller), kterou využívá i framework Laravel,^[3] na kterém je framework postaven. Princip fungování MVC architektury je nastíněn na obrázku č. 4.



Obrázek 4: Architektura MVC (Mode-View-Controller)^[7]

2.2.3 Náčrt konceptu frameworku AnyAdmin

Jak již bylo řečeno v předchozí kapitole, framework AnyAdmin staví na architektuře MVC, ale počítá kromě konečného uživatele také s vývojářem. Na obrázku č. 5 vidíme náčrt konceptu frameworku, který je dovysvětlen dále.



Obrázek 5: Konceptuální schéma funkčních prvků frameworku

Vše, co je na obrázku č. 5 vyznačené **modře** slouží ke zobrazování dat koncovému uživateli aplikace, tedy „View“ z konceptu MVC. To je v podstatě podoba toho, co vidí uživatel na monitoru, podle konceptu, který je znázorněn na obrázku č. 2. Vše ostatní se odehrává na serveru, mimo webový prohlížeč.

Červeně vyznačené položky jsou položky, do kterých vstupuje vývojář. Tedy definiční soubory, a vývojářem definované funkce a komponenty.

- **Definiční soubory** popisují podobu aplikace, jednotlivé stránky (sekce), práva i entity, se kterými aplikace pracuje.
- Na úrovni **funkcí** může vývojář definovat jakoukoli akci, kterou lze v aplikaci spustit a pomocí definičního souboru tuto akci přiřadit ke konkrétnímu akčnímu tlačítku. Tlačítko se může vyskytovat také u jednotlivých instancí entit.
- Na úrovni **komponent** může vývojář definovat nové chování aplikace, zobrazit data jiným způsobem, či pozměnit jejich zápis.

Zeleně vyznačené položky jsou Generátor, který umožňuje generování položek a dynamicky generované položky. Jsou to modely **Entit aplikace**, se kterými uživatel v aplikaci pracuje (např. faktury) a **migrační soubory**, které upravují strukturu databáze podle potřeby.

Šedě jsou na obrázku vyznačené pomocné položky.

3 Systém definičních souborů

Jedním z hlavních pilířů frameworku je systém definičních souborů, pomocí kterého vývojář popisuje strukturu aplikace, entity, se kterými se pracuje a jejich vzájemnou provázanost včetně práv uživatelů pro přístup k jednotlivým částem aplikace či akcím. Definiční soubory jsou ve formátu JSON.

Definiční soubory se v aplikaci nacházejí v adresáři „resources/def.“ Systém definičních souborů má následující strukturu:

- **app.json** – hlavní informace o aplikaci, název, struktura menu, apod.
- **pages** – jednotlivé sekce aplikace (stránky)
 - page1.json
 - page2.json
 - ...
- **Entities** – entity vyskytující se v aplikaci (např. zákazníci, faktury)
 - entity1.json
 - entity2.json
 - ...

3.1 Dílčí datové objekty definičních souborů

Dílčími datovými objekty definičních souborů jsou datové objekty, které se objevují ve více definičních souborech a mají stále stejnou strukturu, patří sem například tlačítka, akce, uživatelská omezení, atp.

3.1.1 Položka menu (_menu_item)

Objekt položka menu se vyskytuje v hlavním definičním souboru app.json. Jde o popis položky v menu, která odkazuje na určitou stránku.

Struktura datového objektu položky menu:

- **title** = název, který se zobrazí v menu
nepovinné – pokud je **target** konkrétní stránka, použije se název stránky
typ: string
- **icon** = ikona (CSS třída)
nepovinné – pokud je **target** konkrétní stránka, použije se ikona stránky
typ: string
- **tooltip** = informace, které se zobrazí po najetí myši na položku
typ: string
- **target** = název stránky, nebo výčet dalších položek menu (vytváří submenu)
typ: string | array
 - pokud je **string** jde o název stránky
 - pokud je **array** aplikace předpokládá výčet dalších položek menu (`_menu_item`). Tímto způsobem vzniká submenu.

Názorná ukázka položek menu i se submenu je zřejmá z výpisu z programu č. 1. První položka má definovaný název a ikonu. V případě druhé a třetí položky název i ikona definovány nejsou a přechází tedy z nastavení cílové stránky. Ve čtvrtém případě jde o submenu s výčtem dalších položek menu, které odkazují na stránky „users“ a „user_groups“.

```
"menu": [  
  {  
    "title": "Customers",  
    "icon": "fa fa-user-tie",  
    "target": "customers"  
  },  
  {"target": "projects"},  
  {"target": "invoices"},  
  {  
    "title": "Users",  
    "icon": "fas fa-users",  
    "target": [  
      {"target": "users"},  
      {"target": "user_groups"}  
    ]  
  }  
]
```

Výpis 1: Ukázka dílčích datových objektů definičních souborů: položky menu (`_menu_item`)

3.1.2 Uživatelské omezení (_access)

Objekt uživatelské omezení se vyskytuje v definičních souborech stránek (pages) a v dílčích datových objektech akcí (_action). Uživatelské omezení zakazují či naopak povolují určité skupině uživatelů zobrazit sekci (stránku) či provést nějakou akci (např. zobrazit komponentu nebo spustit nějakou funkci).

Struktura datového objektu uživatelského omezení:

- **type** = typ přístupu, může nabývat hodnot:
výchozí: denied
typ: string
 - **denied** = hodnota **groups** bude obsahovat uživatelské skupiny, které práva nemají, ostatní skupiny oprávnění mají, pokud je tedy hodnota **groups** prázdná, přístup mají všichni přihlášení uživatelé
 - **allowed** = hodnota **groups** bude obsahovat uživatelské skupiny, které do práva mají, ostatním uživatelům bude přístup odepřen, pokud je hodnota **groups** prázdná, přístup nemá nikdo
 - **free** = přístup do sekce je povolen všem, včetně nepřihlášených uživatelů
- **groups** = názvy uživatelských skupin oddělené čárkou
typ: string

Názorná ukázka datového objektu uživatelského omezení je k nahlédnutí v následujícím výpisu z programu.

```
"access": {  
  "type": "allowed",  
  "groups": "admins,power_users"  
},
```

Výpis 2: Ukázka dílčího datového objektu definičních souborů: uživatelské omezení (_access)

3.1.3 Tlačítko (_button)

Objekt tlačítko se vyskytuje v definičních souborech stránek a entit. U stránek hraje roli akčních tlačítek (buttons) a položek menu (menu). U entit se tlačítka objevují jako vývojářem definované akce (custom_actions), které se zobrazují ve výpisech a detailech konkrétní instance entity. Tlačítka fungují jako iniciátory interakce s uživatelem aplikace. Spouští vývojářem definované funkce a zobrazují stránky či komponenty.

Struktura datového objektu tlačítka:

- **title** = název položky
povinné
typ: string
- **icon** = ikona (CSS třída)
typ: string
- **tooltip** = nápověda, po najetí myší
typ: string
- **classes** = CSS třídy
typ: string
- **bs_color** = název bootstrap barvy tlačítka (např. primary, success, danger)
typ: string
- **action** = akce, která bude následovat po kliknutí
povinné
typ: object
 - obsahuje dílčí datový objekt akce **_action** (viz následující kapitola)

Názorná ukázka datového objektu tlačítka je k nahlédnutí v následujícím výpisu z programu.

```
{  
  "title": "New customer",  
  "icon": "fa fa-plus-square",  
  "bs_color": "success",  
  "action": {"type": "component"...}  
}
```

Výpis 3: Ukázka dílčího datového objektu definičních souborů: tlačítko (`_button`)

3.1.4 Akce (`_action`)

Objekt akce se vyskytuje v datovém objektu tlačítek, jde o akci, která se provede, pokud uživatel interaguje s aplikací. Objekt akce se může vrátit také jako výzva k další akci od serveru při zpracování vývojářem definované funkce. Tím pádem je možné uživatele provést sérií akcí, které vedou k výsledku. K akci je možné také připojit datové vstupy v případě, že je to potřeba.

Například, máme definovanou funkci rozesílání e-mailů. Jako uživatel se rozhodneme, že chceme rozeslat emaily (kliknutí na tlačítko v sekci). Akce vyvolá funkci, která vyžaduje uživatelské vstupy, samozřejmě musíme zvolit příjemce, například z adresáře zákazníků. Po zadání příjemců odešleme data funkci a ta nám zpětně vrátí výsledek, že vše proběhlo v pořádku a případně nás vyzve nás k další akci (Přejete si: Odeslat další e-maily, Označit zákazníky, kterým email odešel, apod.). Všechny tyto funkce samozřejmě už musejí být definované vývojářem, nejedná se o výchozí chování. O předávání dat a výzvy k akcím se stará Front-endový controller (viz kapitola 8)

Struktura datového objektu akce:

- **type** = typ akce, může nabývat hodnot:
povinné
typ: string
 - **page** = načtení nové stránky
 - **component** = načtení komponenty ve vyhrazeném prostoru na aktuální stránce
 - **function** = spuštění vývojářem definované funkce
- **access** = přístupová práva k provedení akce
typ: object
 - obsahuje dílčí datový objekt uživatelského oprávnění **_access**
- **confirm** = vyžádat od uživatele potvrzení, že opravdu chce operaci provést (otevře se dialogové okno s možnostmi ano/ne)
výchozí: false
typ: boolean
- **target** = cíl akce (název stránky, komponenty nebo funkce)
povinné
typ: string
- **data** = data, která se předají cílové stránce, funkci či komponentě. Například komponentě tabulky, kterou entitu má zobrazit, jak má záznamy seřadit, apod.
typ: object
- **inputs** = seznam uživatelských vstupů, v případě, že uživatel musí při akci doplnit nějaké informace.
typ: object
 - obsahuje výčet dílčích datových objektů datového vstupu **_input**

Názorná ukázka datového objektu akce je k nahlédnutí na následujícím výpisu z programu.

```
"action": {  
  "type": "component",  
  "target": "table",  
  "data": {  
    "entity": "customer",  
    "order": "name:asc",  
    "page_limit": 15,  
    "filters": true,  
    "show_fields": "",  
    "cond": ""  
  }  
}
```

Výpis 4: Ukázka dílčího datového objektu definičních souborů: akce (`_action`)

3.1.5 Sekce vstupního formuláře (`_section`)

Objekt sekce vstupního formuláře se vyskytuje ve vstupním formuláři v definičních souborech entit. Sekce vstupního formuláře rozděluje vstupní formulář do sekcí, které zpřehledňují jeho obsah. Funkce sekcí je tedy primárně estetická. Sekce umožňují vývojáři formulář přehledně poskládat a pomáhají se správným nastavením responzivního chování.

Struktura datového objektu sekce vstupního formuláře:

- **title** = název sekce
povinné
typ: string
- **icon** = ikona (CSS třída)
typ: string
- **classes** = CSS třídy sekce, v kombinaci s mřížkovým systémem bootstrapu zajistí responzivitu a dokonalou kontrolu nad rozložením sekcí ve formuláři.
typ: string
- **display_header** = true/false, zobrazit hlavičku sekce
výchozí: true
typ: boolean

- **inputs** = seznam uživatelských vstupů
typ: object
 - obsahuje výčet dílčích datových objektů datového vstupu **_input**
(viz následující kapitola)

Názorná ukázka datového objektu sekce vstupního formuláře je k nahlédnutí na následujícím výpisu z programu.

```
"main_info":{
  "title": "Main Information",
  "icon": "fa fa-user-tie",
  "classes": "",
  "display_header": true,
  "inputs": {...}
},
```

Výpis 5: Ukázka dílčího datového objektu definičních souborů: sekce vstupního formuláře (*_section*)

3.1.6 Datový vstup (*_input*)

- **title** = název datového vstupu v aplikaci
povinné
typ: string
- **icon** = ikona (CSS třída)
typ: string
- **type** = typ vstupu. Podle typu vstupu se vygeneruje potřebný sloupec v databázi a zobrazí se konkrétní podoba vstupu ve formuláři. Například typ „date“ bude mít v DB typ DATE a ve formuláři se zobrazí políčko s pluginem Bootstrap DateTimePicker, který umožní jednoduchý výběr datumu.
Typ datového vstupu má syntaxi: `type:arg1,arg2,...`
Argumenty typu jsou v hranatých závorkách kvůli znázornění, že jde o variabilní složku, do syntaxe však nepatří

Typ datového vstupu může nabývat následujících hodnot:

povinné

typ: string

- **string** = řetězec znaků
- **int** = celé číslo
- **float:[M],[D]** = desetinné číslo
 - **M** = max. počet číslic
 - **D** = počet číslic za desetinnou čárkou
- **date** = datum, ve vstupním formuláři se objeví plugin Bootstrap DateTimePicker
- **time** = čas, ve vstupním formuláři se objeví plugin Bootstrap DateTimePicker s možností výběru pouze času
- **datetime** = datum a čas, ve vstupním formuláři se objeví plugin Bootstrap DateTimePicker s možností výběru datumu i času
- **boolean** = pravda/nepravda
- **html** = formátovaný text v HTML, ve vstupním formuláři se zobrazí editor
- **text** = neformátovaný volný text
- **color** = výběr barvy, ve vstupním formuláři se objeví plugin jQuery Mini-Colors
- **select:[entity]** = výběr jiné entity, podmínkou je existence partnerské entity
 - **entity** = název partnerské entity
- **select_inv:[entity],[field]** = Inverzní výběr z jiné entity. Umožňuje manipulovat s daty zdrojové entity i z druhé strany ve formuláři partnerské entity. Podmínkou je existence datového vstupu typu **select** na zdrojové entitě a příslušném datovém vstupu.
 - **entity** = název zdrojové entity
 - **field** = název datového typu na zdrojové entitě s typem **select**
- **multiselect:[entity]** = výběr jedné nebo více entit, podmínkou je existence partnerské entity.
 - **entity** = název partnerské entity

- **multiselect_inv:[entity],[field]** = Inverzní výběr z jiné entity. Umožňuje manipulovat s daty zdrojové entity i z druhé strany ve formuláři partnerské entity. Podmínkou je existence datového vstupu typu **multiselect** na zdrojové entitě a příslušném datovém vstupu.
 - **entity** = název zdrojové entity
 - **field** = název datového vstupu na zdrojové entitě s typem **select**
- **file:[ext1],[ext2],[...]** = soubor, ve vstupním formuláři se objeví plugin Dropify
 - **ext** = povolené typy nahrávaného souboru
- **files:[ext1],[ext2],[...]** = soubory, ve vstupním formuláři se objeví plugin Dropzone.js
 - **ext** = povolené typy nahrávaných souborů
- **calculated:[func]** = Umožňuje dopočítat některé hodnoty, které chceme zobrazit, ale není nutné jejich zadání, například počet projektů u zákazníka.
 - **func** = Název vývojářem definovaná funkce v modelu entity, která dopočítá pole, pole není zapsané v DB
- **relation_extra_inputs** = seznam uživatelských vstupů. Tyto vstupy se zobrazí po každé když se uživatel pokusí přidat vztah definovaný typem select nebo multiselect. Toto je užitečné v případě, že ke vztahu potřebujeme přidat nějaké informace. Příkladem jsou kontaktní osoby, které přidáváme k zákazníkům a chceme vědět, jakou pozici v té konkrétní firmě zastávají. Jeden člověk například může být kontaktní osobou pro dva různé zákazníky a mít u každého z nich jinou pozici.
typ: object
 - obsahuje výčet dílčích datových objektů datového vstupu **_input**
- **generate_value_func** = Název vývojářem definované funkce, která vygeneruje políčko, např. číslo faktury, na rozdíl od typu **calculated** se hodnota uloží se do DB
typ: string
- **regenerate_on_edit** = Znovu vygenerovat políčko, pokud se entita změní
typ: boolean

- **index** = přidat do databáze nad příslušným sloupcem index, může nabývat hodnot:
typ: string
 - **index**
 - **primary**
 - **unique**
- **rules** = validační podmínky, viz. dokumentace Laravelu^[3]: Výčet dostupných možností zde: <https://laravel.com/docs/6.x/validation#available-validation-rules>
typ: string
- **fillable_add** = Políčko je možné vyplnit ve formuláři nového záznamu (pokud je false, políčko se zobrazí zašedlé)
výchozí: true
typ: boolean
- **fillable_edit** = Políčko je možné vyplnit ve formuláři při editaci (pokud je false, políčko se zobrazí zašedlé)
výchozí: true
typ: boolean
- **filterable** = ve výpisech lze podle pole filtrovat
výchozí: true
typ: boolean
- **filter_type** = typ filtrace datového vstupu
výchozí: text
typ: string
 - **text** = textové vyhledávání
 - **select** = výběr jedné možnosti
 - **multiselect** = výběr jedné nebo více možností
 - **range** = rozmezí, může být aplikováno pouze u datatových typů
 - **date**
 - **datetime**
 - **int**
 - **float**

- **display_in_form** = zobrazit tento datový vstup ve formulářích
výchozí: true
typ: boolean
- **display_in_listings** = zobrazit tento datový vstup ve výpisech
výchozí: false
typ: boolean
- **classes** = CSS třídy datového vstupu, umožňuje měnit vzhled v prohlížeči
typ: string
- **wrapper_classes** = CSS třídy HTML elementu, který je rodičem datového vstupu, umožňuje měnit vzhled v prohlížeči
typ: string
- **prefix** = hodnota datového vstupu se v aplikaci vždy zobrazí s touto předponou
typ: string
- **suffix** = hodnota datového vstupu se v aplikaci vždy zobrazí s touto příponou
typ: string

Názorná ukázka datového objektu datového vstupu je k nahlédnutí na následujícím výpisu z programu.

```
"address": {  
  "title": "Address",  
  "icon": "",  
  "type": "text",  
  "rules": "required",  
  "filterable": false,  
  "display_in_listings": false  
},
```

Výpis 6: Ukázka dílčího datového objektu definičních souborů: datový vstup (*_input*)

3.2 Hlavní definiční soubor app.json

Pomocí hlavního definičního souboru popíše vývojář vlastnosti aplikace jako její název, logo a vytvoří strukturu menu. Příklad použití je k nahlédnutí ve výpisu z programu č. 7 níže. Vývojář samozřejmě může v případě potřeby přidávat vlastní konfigurace, které dále v kódu využije.

```
1  {
2    "app_name": "AnyAdmin",
3    "logo": "img/logo.png",
4    "favicon": "img/favicon.png",
5    "home_page": "customers",
6    "menu": [
7      {"target": "customers"},
8      {"target": "projects"},
9      {"target": "invoices"}
10   ]
11 }
12
```

Výpis 7: Příklad použití definičního souboru app.json (výchozí podoba)

Struktura hlavního definičního souboru app.json:

- **app_name** = název aplikace
typ: string
- **logo** = název souboru s logem (v adresáři „public“)
typ: string
- **favicon** = název souboru s ikonou (v adresáři „public“)
typ: string
- **home_page** = název domovské stránky aplikace (sem je uživatel přesměrován po přihlášení)
typ: string
- **menu** = pole položek menu, jednotlivé položky mají následující podobu:
typ: object
 - obsahuje výčet dílčích datových objektů položek menu **_menu_item**

3.3 Definiční soubory entit

V definičním souboru entit je popsána nejen datová struktura entity, ale také jejich vzájemná provázanost, podoba vstupního formuláře při editaci a přidávání a také zde mohou být definovány vlastní operace, které s konkrétními instancemi entit lze provádět. Názorná ukázka zkráceného definičního souboru entity zákazníka je k nahlédnutí na výpisu z programu č. 8 níže.

```
1  {
2    "title": "Customer",
3    "icon": "fa fa-user-tie",
4    "title_field": "name",
5    "allow_add": true,
6    "allow_edit": true,
7    "allow_delete": true,
8    "input_form": {
9      "main_info": {
10        "title": "Main Information",
11        "icon": "fa fa-user-tie",
12        "classes": "",
13        "display_header": true,
14        "inputs": {
15          "name": {"title": "Name..."},
16          "address": {"title": "Address..."},
17          "ident_number": {"title": "Identification Number..."},
18          "vat_id": {"title": "VAT ID..."}
19        }
20      },
21      "contact_info": {
22        "title": "Contact Information",
23        "classes": "fa fa-address-card",
24        "inputs": {
25          "contact_persons": {"title": "Contact persons..."}
26        }
27      },
28      "more_info": {
29        "title": "More Info",
30        "classes": "fa fa-inbox",
31        "inputs": {
32          "projects_number": {"title": "Number of projects implemented..."},
33          "total_income": {"title": "Total income from customer..."}
34        }
35      }
36    },
37    "custom_actions": [
38      {
39        "title": "New Invoice",
40        "icon": "fa fa-check",
41        "bs_color": "success",
42        "tooltip": "New Invoice",
43        "action": {
44          "type": "function",
45          "target": "makeInvoice",
46          "data": {},
47          "inputs": {
48            "price": {"title": "Price..."},
49            "description": {"title": "Work description..."}
50          }
51        }
52      }
53    ]
54  }
```

Výpis 8: Ukázka definičního souboru entity zákazníka

Struktura definičního souboru entity:

- **title** = název entity v aplikaci
povinné
typ: string
- **icon** = ikona (CSS třída)
typ: string
- **title_field** = název datového vstupu, jež bude využit jako titulní (např. u zákazníka to bude název, tedy datový vstup „**name**“, který je definován mezi vstupy ve vstupní formuláři). Datový typ s tímto názvem musí mezi datovými vstupy vstupního formuláře existovat.
povinné
typ: string
- **allow_add** = povolit přidávání nových záznamů
výchozí: true
typ: boolean
- **allow_edit** = povolit editaci
výchozí: true
typ: boolean
- **allow_delete** = povolit odstraňování
výchozí: true
typ: boolean
- **input_form** = vstupní formulář (slouží zároveň jako popis datové struktury), na základě konfigurací v této sekci definičního souboru se vygenerují příslušné databázové tabulky. První vrstva formuláře obsahuje sekce a ty následně obsahují jednotlivé datové vstupy.
typ: object
 - obsahuje výčet dílčích datových objektů sekcí vstupního formuláře **_section**
 - **_section** následně obsahuje výčet dílčích datových objektů datového vstupu **_input**
- **custom_actions** = vlastní speciální funkce definované vývojářem, které je možné provádět s konkrétními instancemi entity. zobrazí se jako tlačítko ve výpisech vedle dalších operací jako editace, detail, smazat.
typ: array
 - obsahuje výčet dílčích datových objektů tlačítek **_button**

3.4 Definiční soubory stránek

V definičním souboru stránek je popsána její hrubá struktura, tedy struktura menu a akčních tlačítek na stránce. Zároveň je zde soubor základních nastavení jako název a ikona. Jak víme na stránce se je prostor pro vykreslení komponenty, do tohoto prostoru se vykreslí první komponenta z akcí v definovaném menu. Pokud zde není definovaná žádná komponenta, nic se nevykreslí.

Klíčové je také nastavení přístupu. Stránky se chovají jako celé sekce, na které je možno odkazovat z hlavního menu, pokud tedy na stránku není uživateli povolen přístup, v menu se ani nezobrazí. Tím aplikace automaticky docílí různé podoby pro různé skupiny uživatelů. Jednoduchá podoba stránky se zákazníky je k vidění na výpisu z programu č. 9 níže.

```
1  {
2      "title": "Customers",
3      "icon": "fas fa-user-tie",
4      "access": {
5          "type": "allowed",
6          "groups": "admins,power_users"
7      },
8      "menu": [
9          {
10             "title": "All",
11             "icon": "",
12             "action": {
13                 "type": "component",
14                 "target": "table",
15                 "data": {"entity": "customer"...}
16             }
17         }
18     ],
19     "buttons": [
20         {
21             "title": "New customer",
22             "icon": "fa fa-plus-square",
23             "bs_color": "success",
24             "action": {
25                 "type": "component",
26                 "target": "form",
27                 "data": {"entity": "customer"...}
28             }
29         }
30     ]
31 }
```

Výpis 9: Ukázka definičního souboru stránky (sekce) se zákazníky

Struktura definičního souboru stránky:

- **title** = název sekce, který se zobrazí v aplikaci
povinné
typ: string
- **icon** = ikona sekce (CSS třída)
typ: string
- **access** = přístupová práva k sekci (nepovinné)
typ: object
 - obsahuje dílčí datový objekt uživatelského omezení **_access**
- **menu** = pole, seznam tlačítek
typ: array
 - obsahuje výčet dílčích datových objektů tlačítek **_button**
- **buttons** = pole, seznam tlačítek
typ: array
 - obsahuje výčet dílčích datových objektů tlačítek **_button**

4 Entity

Entity jsou v aplikaci abstraktní datové objekty, které jsou popsány příslušným definičním souborem. Struktura definičního souboru je popsána v kapitole 3.3 Definiční soubory entit. Jako entitu si můžeme představit cokoliv s čím v aplikaci budeme pracovat, vytváříme-li například CRM systém, entitami mohou být zákazníci, kontaktní osoby, realizované nabídky, apod.

4.1 Základní popis entity

Každá entita je popsána:

Souborem základních informací

- Patří sem například různé nastavení, název a ikona které se zobrazí ve výpisech

Vstupním formulářem (input form)

- Vstupní formulář obsahuje datové vstupy, pomocí kterých generátor vytvoří příslušné databázové tabulky
- Datové vstupy je možné zobrazit pouze v editačním formuláři či pouze ve formuláři pro nový záznam nebo vstup nezobrazit vůbec.
- Datové vstupy lze také namapovat na **vývojářem definované funkce**, tím pádem je zajištěna **plná kontrola i nad zpracováním datového vstupu**. Toto chování se hodí u vstupů, které nechceme po uživateli zadat, ale chceme je automaticky vygenerovat (např. číslo faktury).
- Formulář je rozdělen do **sekcí**, které datové vstupy shlukují.
- Jednotlivé datové vstupy i sekce mají možnost aplikace CSS tříd, tím pádem má vývojář **plnou správu nad vzhledem i responzivitou formuláře**

Vývojářem definovanými akcemi (custom actions)

- Toto je soubor možných akcí, které se u každé entity zobrazí jako tlačítka, jak ve výpisu, tak v detailu. Po kliknutí následuje vývojářem definovaná akce. Může jít o zobrazení komponenty, stránky nebo vývojářem definované funkce, jako v případě akčních tlačítek u stránek. Jak se vývojářem definované funkce chovají popisuje kapitola 7.4.

4.2 Vztahy mezi entitami

Entity lze mezi sebou navzájem provazovat. Framework toto dělá **plně automaticky**. Dávat entity do vztahů lze jednoduše určením typu datových vstupů v entitě. Pokud entita typy těchto datových typů obsahuje, **generátor tuto skutečnost automaticky rozpozná** a vygeneruje všechny potřebné databázové tabulky. Zároveň framework automaticky namapuje veškeré vztahy v aplikaci, které jsou pak dostupné přes Eloquent, který je součástí Laravelu. Jakým způsobem se vztahy mapují je popsáno v dokumentaci Laravelu zde: <https://laravel.com/docs/6.x/eloquent-relationships>.

Typy datových vstupů vytvářející vztah jsou:

- select
- select_inv
- multiselect
- multiselect_inv

Jak se tyto typy datových vstupů chovají a co znamenají je popsáno v kapitole 3.1.6 Datový vstup (_input)

5 Stránky

Stránky jsou jednotlivé sekce aplikace, ke kterým mohou mít přístup jen určití uživatelé. Jde tedy o jakési sekce, které aplikaci rozdělují. Tato vrstva tedy slouží jako **hlavní controller uživatelských oprávnění**. Stručně nastíněná podoba stránky je zřejmá z obrázku č. 6 níže.



Obrázek 6: Konceptuální rozložení prvků na stránce

Stránky obsahují:

- **Submenu**
 - Mohou přejít na jinou stránku, zobrazit komponentu ve vyhrazeném prostoru viz oranžově vyznačené pole na obrázku č. 6 (Dynamicky generovaný pohled).
- **Akční tlačítka**
 - Chovají se stejně jako submenu, mohou přejít na jinou stránku, zobrazit komponentu ve vyhrazeném prostoru, ale navíc mohou **spustit speciální funkci definovanou vývojářem**, která vrátí informaci o výsledku a případně vyzve k další akci. Více o akcích a funkcích viz kapitola 3.1.4 a 7.
- **Vyhrazený prostor pro komponentu**

- V tomto prostoru se zobrazí příslušná komponenta s obsahem či vstupním formulářem
- viz kapitola 6 Komponenty

5.1 Funkce stránek

Jak již bylo řečeno stránky rozdělují aplikaci do sekcí. Slouží tedy k jakémusi **zpřehlednění obsahu**.

Zároveň slouží jako **hlavní uživatelský controller**, pokud na některou stránku je uživateli zakázaný přístup, veškerý obsah i funkce této sekce jsou uživateli skryty.

Třetí funkcí stránek je **funkce zobrazování obsahu**. Stránka tedy slouží jako jakási šablona. Spolu s Front-endovým controllerem (viz kapitola 8), který odposlouchává interakci uživatele, zobrazuje uživateli požadovaný obsah ve vyhrazeném místě pro komponentu. Komunikace probíhá asynchronně přes rozhraní AJAX, uživatel v rámci sekce tedy nikdy nenačítá celou stránku znovu, pouze zobrazuje komponenty.

Ve vyhrazeném místě pro komponentu se jako výchozí komponenta zobrazí ta komponenta, která je v položkách menu jako první a na kterou má uživatel oprávnění. Pokud žádná taková komponenta neexistuje stránka zůstane prázdná. Tlačítka se zobrazí pouze v případě, že existuje nějaké tlačítko, na které má uživatel oprávnění. Pokud stránka neobsahuje žádný obsah, uživatel o tom dostane zprávu.

6 Komponenty

Komponenty zobrazují data, tedy zobrazují výpis jednotlivých entit ve formě tabulek, grafů či detailů jednotlivých instancí entit nebo mohou data také zpracovávat ve formě vstupních a editačních formulářů.

Dynamické komponenty jsou funkční celky aplikace a vyskytují se na stránce (viz předchozí kapitola) v ohraničeném prostoru. Komponentu vyvolává Front-endový controller (viz kapitola 8), který hlídá interakce uživatele a v případě požadavku na komponentu odešle asynchronní AJAXový dotaz spolu s daty pomocí POST requestu a následně komponentu zobrazí na stránce.

Vývojář není omezen pouze na výčet předem definovaných komponent, ale může libovolnou komponentu vytvořit a v aplikaci jí použít.

6.1 Složení komponenty

Komponenta sestává ze dvou souborů:

- PHP třída
 - Třídy komponent se nacházejí v adresáři „app/Components“ a mají vždy název: `ExampleComponent.php`. Tato třída extenduje třídu `App\Component` a jedinou podmínkou je, že musí obsahovat funkci **handle()**, která zpracovává požadavek a vrací odpověď.
- Pohled (blade) – *nepovinné*
 - Pohled je povinnou součástí, v případě, že funkce **handle()** třídy komponenty vrací v odpovědi pohled. Pohledy komponent jsou v adresáři „resources/components“ a jsou to klasické blade soubory, které používá Laravel^[3]. Více o šablonovacím systému blade zde: <https://laravel.com/docs/6.x/blade>

6.2 Komunikace s aplikací

Komunikaci s komponentou na straně prohlížeče, tedy požadavek na její zobrazení, zajišťuje Front-endový controller (viz kapitola 8).

Pokaždé když je daná komponenta v kódu zavolána, automaticky se zkontrolují veškerá oprávnění uživatele ji zobrazit. Do komponenty samotné již pak jen přijde požadavek a

následně z ní odejde odpověď, stejným způsobem, jako to řeší Controllery Laravelu. PHP třída komponenty je pouze prodlouženou rukou Controlleru, který komponentu zobrazuje. Pokaždé když požadavek na zobrazení projde, zavolá se v této třídě funkce **handle()**. Tato funkce jen musí vrátit klasickou odpověď Laravelu, tedy například pohled **view()** s příslušným blade souborem.

Pokud vytváří vývojář novou komponentu, má **naprostou kontrolu nad celým požadavkem i odpovědí**. Požadavek může libovolně zpracovat ve funkci `handle()` a odpověď, může libovolně zobrazit pomocí blade souboru.

6.3 Přednastavené základní komponenty

Framework obsahuje tři základní komponenty:

1. Komponentu pro zobrazení dat celé entity, tabulku (**table**)
2. Komponentu pro zobrazení dat konkrétní instance entity (**detail**)
3. Komponentu pro zpracování dat (zápis/editaci), formulář (**form**)

6.3.1 Komponenta tabulky (table)

Tabulka zobrazuje libovolnou entitu ve formě tabulky, zároveň řídí filtraci, stránkování a řazení dat.

Vstupní data komponenty „table“:

```
"action": {
  "type": "component",
  "target": "table",
  "data": {
    "entity": "customers",
    "order": "name:asc",
    "page_limit": 15,
    "filters": true,
    "show_fields": "",
    "cond": ""
  }
}
```

Výpis 10: Vstupní data komponenty „table“

Vysvětlení hodnot z výpisu z programu č. 10:

Na výpisu z programu č. 10 je část definičního souboru stránky se zákazníky (akce po kliknutí na položku menu), na které se má zobrazit komponenta tabulky s daty všech zákazníků, zákazníci budou seřazení podle abecedy a podle názvu a zobrazí se jich na stránku patnáct. Tabulku lze filtrovat. Parametry akce, jako „type“ a „target“ již známe z kapitoly 3.1.4 Akce (_action). Zajímá nás nyní obsah objektu „data“. Jde o data která předáváme komponentě a na základě kterých dostáváme zpět naplněnou tabulku.

- **entity** = entita, kterou má komponenta zobrazit, v našem případě jsou to zákazníci
povinné
typ: string
- **order** = [input_name]:[direction], seřazení dat v tabulce.
výchozí: created_at:desc – řazení podle data vytvoření
typ: string
- **page_limit** = počet záznamů na stránku
výchozí: 15
typ: int
- **filters** = povolit filtraci, zobrazit filtry
výchozí: true
typ: boolean
- **show_fields** = výčet datových vstupů, které se mají zobrazit v tabulce, pokud je hodnota prázdná, řídí se definičním souborem entity. Hodnota „title_field“, která reprezentuje entitu, je zobrazena vždy. Názvy datových vstupů entity oddělené čárkou.
typ: string
- **cond** = speciální podmínky, přímý dotaz do databáze, zpracovává laravel funkce **whereRaw**. Více viz Laravel Query Builder: <https://laravel.com/docs/6.x/queries>
typ: string

Zákazník	Počet realizovaných projektů	Výnos od zákazníka	Datum vytvoření	Nástroje
Sample customer s.r.o.	12	1 200 000,-Kč	15.10.2015	[edit] [delete] [search] [list]
Sample customer s.r.o.	12	1 200 000,-Kč	15.10.2015	[edit] [delete] [search] [list]
Sample customer s.r.o.	12	1 200 000,-Kč	15.10.2015	[edit] [delete] [search] [list]
Sample customer s.r.o.	12	1 200 000,-Kč	15.10.2015	[edit] [delete] [search] [list]
Sample customer s.r.o.	12	1 200 000,-Kč	15.10.2015	[edit] [delete] [search] [list]
Sample customer s.r.o.	12	1 200 000,-Kč	15.10.2015	[edit] [delete] [search] [list]
Sample customer s.r.o.	12	1 200 000,-Kč	15.10.2015	[edit] [delete] [search] [list]
Sample customer s.r.o.	12	1 200 000,-Kč	15.10.2015	[edit] [delete] [search] [list]
Sample customer s.r.o.	12	1 200 000,-Kč	15.10.2015	[edit] [delete] [search] [list]

Obrázek 7: Zobrazená komponenta tabulky na stránce se zákazníky

6.3.2 Komponenta formuláře (form)

Komponenta formuláře umožňuje uživateli konkrétní instanci nějaké entity vytvořit či editovat. V případě editace musíme komponentě předat také ID instance, kterou si přejeme zeditovat. S takovým případem se v definičním souboru prakticky nesetkáme, ale tímto způsobem je komponenta vyvolána z jiné komponenty. Například z komponenty tabulky, která obsahuje tlačítko editace u konkrétních záznamů.

Vstupní data komponenty „form“:

```
"action": {
  "type": "component",
  "target": "form",
  "data": {
    "entity": "project"
  }
}
```

Výpis 11: Vstupní data komponenty „form“

Na výpisu z programu č. 11 výše vidíme část definičního souboru, která zobrazuje formulář nového záznamu entity projektu.

Struktura dat předávaných komponentě formuláře:

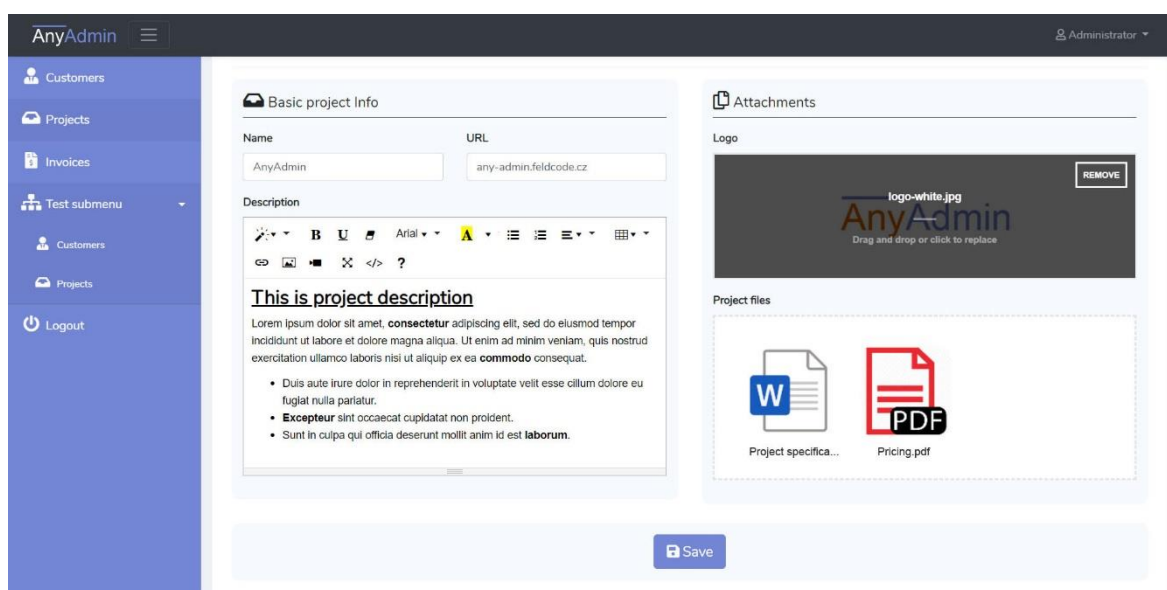
- **entity** = entita, se kterou bude komponenta pracovat (přidávat nový záznam či konkrétní instanci editovat)

povinné

typ: string

- **id** = ID konkrétní instance komponenty, pokud žádné ID zadané není, komponenta ví, že jde o zadání nového záznamu, pokud ID zadané je, zobrazí se editační formulář

typ: int



Obrázek 8: Zobrazená komponenta editačního formuláře entity projektů

6.3.3 Komponenta detailu (detail)

Komponenta detail jednoduše zobrazuje konkrétní instanci entity. Tedy například konkrétní projekt. Jelikož komponenta slouží výhradně k zobrazování konkrétních instancí entit, odkazuje se na ní z jiných komponent. V definičním souboru by bylo její vyvolání netypické i když je samozřejmě možné.

Vstupní data komponenty „detail“:

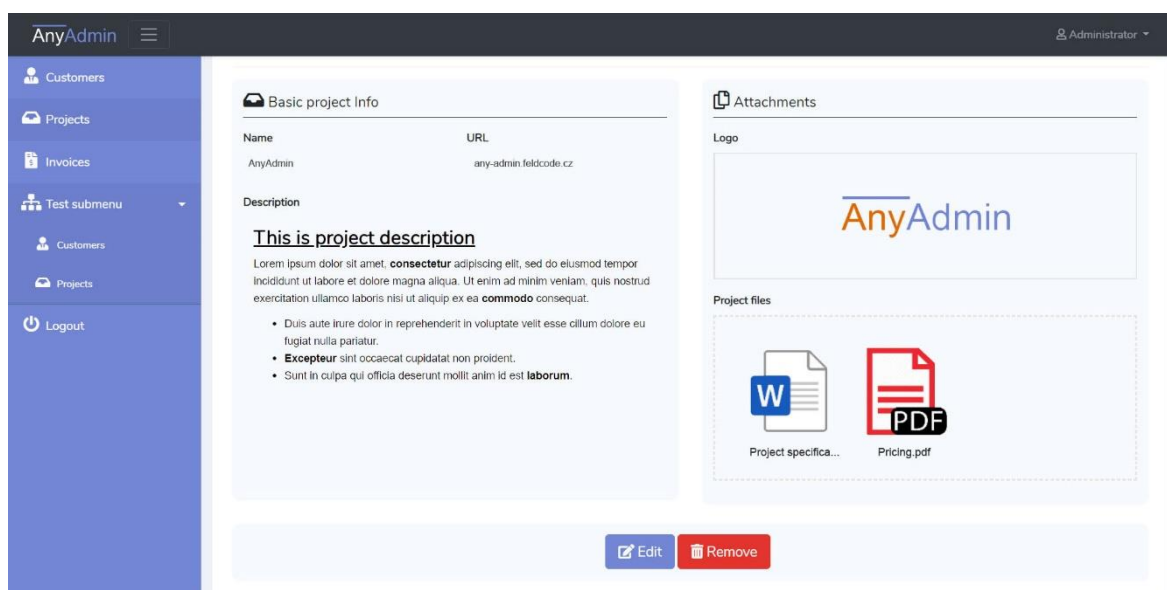
```
"action": {  
  "type": "component",  
  "target": "detail",  
  "data": {  
    "entity": "project",  
    "id": 5  
  }  
}
```

Výpis 12: *Vstupní data komponenty „detail“*

Na výpisu z programu č. 12 výše vidíme část definičního souboru, která zobrazuje detail konkrétní instance entity projektu.

Struktura dat předávaných komponentě detailu:

- **entity** = entita, jejíž instanci má komponenta zobrazit
povinné
typ: string
- **id** = ID konkrétní instance komponenty. Na rozdíl od entity formuláře je ID v tomto případě povinné. Komponenta detailu pouze zobrazuje konkrétní instanci entity.
povinné
typ: int



Obrázek 9: Zobrazená komponenta detailu entity projektu

6.4 Vlastní komponenty definované vývojářem

Jak již bylo řečeno v úvodu kapitoly 6, vývojář může libovolně komponenty vytvářet a měnit tak způsob, jakým se data zobrazují či zpracovávají. Může tímto způsobem přidávat i nové funkcionality, jako například zobrazit konkrétní entitu ve formě grafů, jako statistiky a tak podobně.

Tímto způsobem je **ruka vývojáře v podstatě neomezená** a zároveň se výchozími, velmi jednoduchými komponentami pokrývá velká část možností, které mohou nastat. Je tedy možné **vytvořit základní a plně funkční administrativní aplikaci během několika málo minut až hodin**. V druhé fázi pak vývojář pouze doplní speciální funkcionality, které chybí a celá aplikace je připravena k použití.

6.4.1 Postup vytvoření nové komponenty

Vytvoření komponenty je snadné, stačí pouze vytvořit PHP třídu s názvem komponenty a v případě potřeby také pohledový soubor blade. Jakmile daný soubor s PHP třídou existuje, systém automaticky komponentu rozpozná a je možné se na ni odkazovat. Ukázka vzorové komponenty je k nahlédnutí na výpisu z programu č. 13 níže.

Adresář s PHP třídami komponent: „**app/Components**“

Adresář s pohledovými soubory blade: „**resources/views/components**“

```
1  <?php
2
3  namespace App\Components;
4
5  use App\Component;
6  use Illuminate\Http\Request;
7
8  class SampleComponent extends Component
9  {
10     public function processRequest(Request $request){
11         //process request if needed
12     }
13
14     public function handle(Request $request){
15         $this->processRequest($request);
16         return view('components.'.$this->getName());
17     }
18 }
```

Výpis 13: Ukázka vzorové komponenty (PHP třída)

Při vytváření nové komponenty je nutné pouze dodržet následující podmínky:

- **Název třídy**
 - Název třídy komponenty se vždy nazývá takto:
NázevKomponentyComponent
 - Na komponentu se pak v kódu odkazuje takto:
název_komponenty
(systém název automaticky převede)
- **Název PHP souboru** musí být stejný jako název třídy
- **Funkce handle(Request \$request)** je nedílnou součástí třídy. Do této funkce přijde požadavek na zobrazení

7 Vývojářem definované funkce

V některých případech si nepřejeme zobrazit data ve formě komponenty, ale chceme, aby měl uživatel aplikace možnost spustit nějaký proces. Například rozeslat hromadný e-mail určitým zákazníkům. K tomuto účelu ve frameworku slouží funkce, nad kterými má vývojář opět plnou kontrolu.

7.1 Globální funkce

Globálními funkcemi jsou funkce, které lze spustit libovolně z kterékoli části aplikace. Tyto funkce se zapisují do souboru „**app/Func.php**.“ Funkce slouží pouze jako vstupní bod, proměnná „**request**“ obsahuje veškerá data předávaná funkci, tak jak jsou uvedena v definičním souboru spolu s veškerými uživatelskými vstupy, v případě, že jsou vyžadovány. Pokud by funkcionality byla rozsáhlá, vývojář samozřejmě může vytvořit libovolný pomocný objekt a funkce jako taková pak může sloužit pouze jako můstek.

Způsob zápisu globální funkce je zřejmý z výpisu z programu č. 14 níže.

```
1  <?php
2
3  namespace App;
4
5  use Illuminate\Http\Request;
6
7  class Func
8  {
9      public function sampleFunction(Request $request){
10         //do something here
11     }
12 }
```

Výpis 14: Zápis globální vývojářem definované funkce

7.2 Funkce entit

Funkce entit jsou funkce, které lze vyvolat na konkrétních instancích entit. Typickou předdefinovanou funkcí je funkce smazání konkrétního záznamu. Funkce entit zapisujeme přímo jako funkce PHP třídy konkrétní entity, máme tedy k dispozici přímý přístup k objektu, na kterém funkci spouštíme. Lze také přidávat funkce, které lze vyvolávat na všech entitách bez ohledu na typ entity, tyto funkce patří do souboru „**app/Entity.php**.“

Vývojářem definované funkce konkrétních entit pak patří přímo do souboru s modelem konkrétní entity, který vygeneroval generátor. Tyto soubory generátor nemaže, pouze vytváří, aby se tak zamezilo nechtěné ztrátě dat, právě v podobě vývojářem definovaných funkcí.

Adresář s modely entit: „**app/Entities**“

Způsob zápisu funkce do modelu entity je zřejmý z výpisu z programu č. 15 níže.

```
2
3 namespace App\Entities;
4
5 use App\Entity;
6 use App\Traits\HasDynamicRelation;
7 use Illuminate\Http\Request;
8
9 class CustomerEntity extends Entity
10 {
11     use HasDynamicRelation;
12
13     protected $table = 'ent_customer';
14
15     public function sampleFunction(Request $request){
16         //do something here
17     }
18 }
```

Výpis 15: Zápis vývojářem definované funkce do modelu entity

7.3 Komunikace s Front-endovým controllerem

Definované funkce spouští Front-endový controller, který hlídá interakce s uživatelem a spouští předdefinované akce. Pokud je cílem akce funkce, pak odešle asynchronní dotaz na její zavolání a čeká na odpověď. V back-endové části frameworku dotaz zpracuje controller „**FuncsController**“, který zavolá příslušnou funkci a předá jí veškerá data jež obdržel z Front-endového controlleru. Funkce jako taková vrací JSON objekt, který je postupně předán zpět Front-endovému controlleru. Jako příklad si můžeme představit funkci na rozesílání hromadného e-mailu zákazníkům. Ukázky dotazu a odpovědi funkce jsou k nahlédnutí na výpisech z programu č. 16 a 17 níže.

```
{
  "title": "Odeslat hromadný e-mail",
  "icon": "fa fa-paperplane",
  "bs_color": "primary",
  "tooltip": "Nový hromadný e-mail",
  "action": {
    "type": "function",
    "target": "sendEmails",
    "data": {},
    "inputs": {...}
  }
}
```

Výpis 16: Tlačítko s dotazem na spuštění funkce rozeslání hromadného e-mailu

```
{
  "state": true,
  "msg": "E-maily úspěšně rozeslány vybraným uživatelům",
  "info": [
    "Pokud si přejete rozeslat další hromadný e-mail, klikněte na tlačítko níže",
    "Další informační hlášení"
  ],
  "actions": [
    {
      "title": "Odeslat další hromadný e-mail",
      "icon": "fa fa-paperplane",
      "bs_color": "primary",
      "tooltip": "Nový hromadný e-mail",
      "action": {
        "type": "function",
        "target": "sendEmails",
        "data": {},
        "inputs": {...}
      }
    }
  ]
}
```

Výpis 17: Odpověď funkce rozeslání hromadného e-mailu, předána zpět Front-endovému controlleru

Struktura odpovědi funkce:

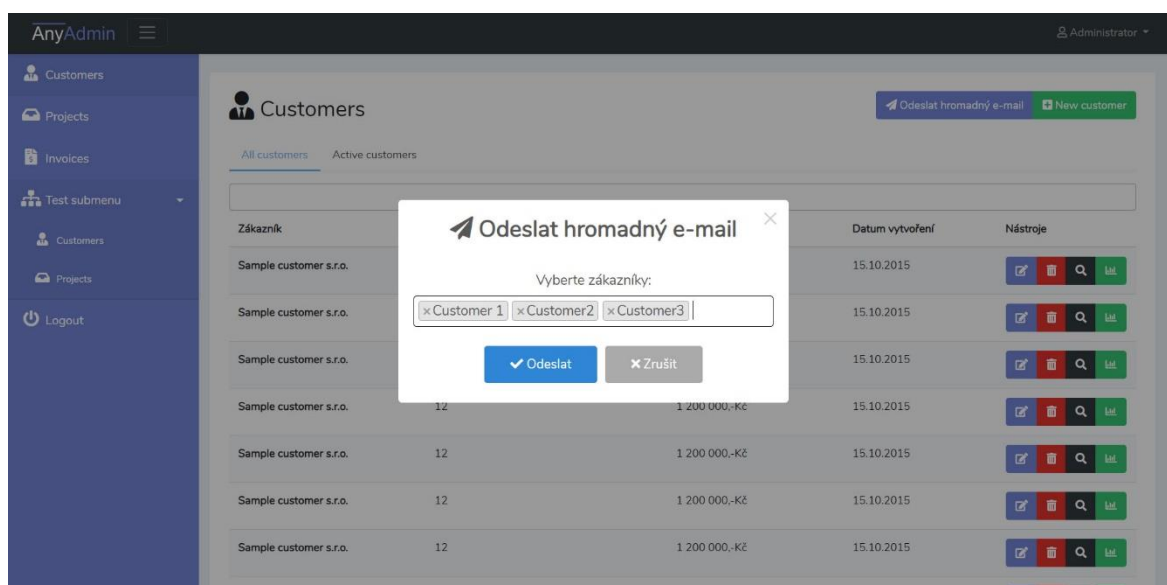
- **state** = výsledek, true = úspěch, false = chyba
povinné
typ: boolean
- **msg** = zpráva, která se zobrazí spolu se stavem (úspěch/neúspěch)
typ: string
- **info** = dodatečné informace, výčet zpráv, které chceme uživateli zobrazit
typ: array[string]

- **actions** = tlačítka s případnými dalšími akcemi, ke kterým uživatele chceme vyzvat
typ: array
 - obsahuje výčet dílčích datových objektů tlačítek **_button**
(viz kapitola 3.1.3 Tlačítko)

7.4 Interakce s uživatelem

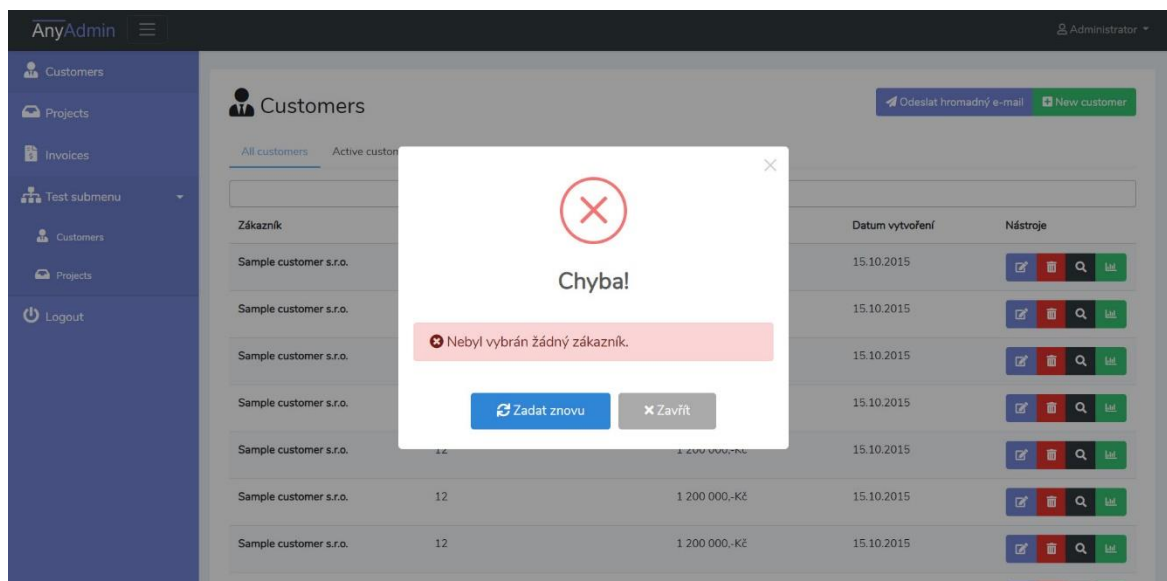
V předchozí kapitole jsme si ukázali, jak vypadá zpracování funkce z pohledu kódu. Nyní se podíváme, jak celou situaci vidí koncový uživatel, jenž funkci vyvolal.

Zůstaneme u příkladu z předešlé kapitoly, tedy funkce rozeslání hromadného e-mailu zákazníkům. Po spuštění akce, Front-endový controller rozpozná, že je vyžadován uživatelský vstup a tím je výběr příjemců (zákazníků). Uživateli se zobrazí dialogové okno, které se ho na požadovaný vstup zeptá, způsobem, který je patrný z obrázku č. 10 níže.



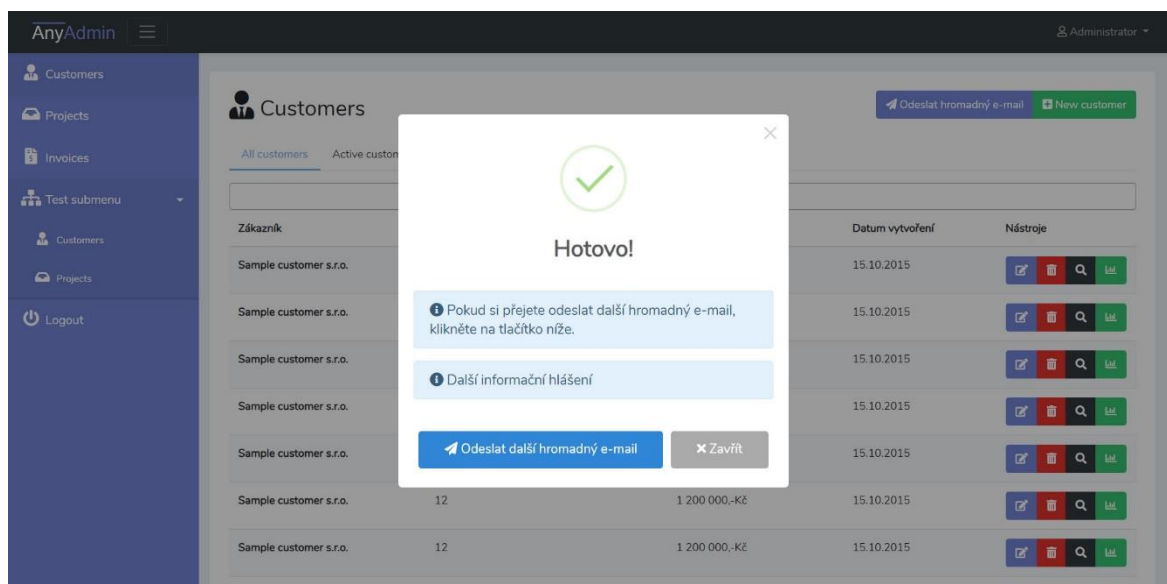
Obrázek 10: Spuštění funkce uživatelem – vyžádání datového vstupu od uživatele

Nastane-li chyba, tedy v případě, že by v ukázce odpovědi z výpisu z programu č. 17 byl parametr „state“ roven hodnotě false, uživatel je o této události upozorněn a je vyzván k případnému opakování akce. Chybu může vyvolat například chybné zadání datových vstupů, jak je patrné z obrázku č. 11 níže.



Obrázek 11: Spuštění funkce uživatelem – chyba

Pokud vše proběhne v pořádku, vidí uživatel hlášku spolu s dalšími dodatečnými informacemi, které by pro něho mohli mít hodnotu a také může být vyzván k další akci. Vše z obrázku č. 12 níže odpovídá ukázce odpovědi funkce z výpisu z programu č. 17. Tlačítek s akcemi zde může být samozřejmě více. Vývojář má volnou ruku.



Obrázek 12: Spuštění funkce uživatelem – úspěch

8 Front-endový controller

Front-endový controller je součástí systému, která odposlouchává interakce uživatele s aplikací. Pokaždé když uživatel v aplikaci klikne na tlačítko či položku menu na stránce, je tato událost předána právě Front-endovému controlleru, který ji zpracuje a spustí požadovanou akci. Front-endový controller je postaven na programovacím jazyku JavaScript a využívá knihovnu jQuery.

Front-endový controller může vyvolat tyto tři typy akcí:

1. **Zobrazit stránku**

Přesměruje uživatele na jinou stránku (viz kapitola 5) v aplikaci

2. **Zobrazit komponentu**

Zobrazí požadovanou komponentu (viz kapitola 6) ve vyhrazeném prostoru na stránce

3. **Spustit definovanou funkci**

Spustí vývojářem definovanou funkci (viz kapitola 7)

Dále Front-endový controller také zajišťuje různé podpůrné činnosti pro lepší uživatelskou zkušenost s aplikací, například předběžnou validaci již na straně front-endu, bez nutnosti odeslání formuláře.

8.1 Komunikace s back-endovou částí aplikace

Pro komunikaci s back-endovou částí aplikace používá Front-endový controller asynchronní dotazy JavaScriptu, takzvaný AJAX^[9]. V případě **zobrazení stránky**, pouze přesměruje na určenou stránku.

V případě **zobrazení komponenty nebo spuštění funkce** volá asynchronní dotaz na URL adresu, na které aplikace běží s prvním parametrem za lomítkem, jako typem dotazu (tedy funkce či komponenta) a druhým parametrem jako názvem cíle. Všechny dodatečná data jsou poté předána metodou POST v objektu spolu s dotazem, ke kterému má komponenta či funkce přístup v proměnné **\$request**, jenž je předána zpracovávající funkci.

URL volání komponenty:

„http://my-application.com/component/název_komponenty“

URL volání funkce:

„http://my-application.com/func/(název_entity)/název_funkce“

- název_entity pouze v případě, že se nejedná o globální funkci

9 Generátor a databáze

Jak již bylo řečeno, framework se snaží o nalezení rovnováhy mezi **usnadněním vývojáři práce a limitováním jeho možností**. Co se struktury databáze týče, víme, že jsou určité zásady, které je důležité dodržovat. Nejdůležitější část práce s databází je promyšlení struktury, její samotné vytvoření je ale v zásadě repetitivní činnost, kterou můžeme vývojáře ušetřit. Vývojář si tedy pouze rozmyslí, jaké data bude která entita mít a ostatní je na frameworku. Může také doplnit další nastavení, jako indexy u důležitých sloupečků. (viz kapitola 3.3 Definiční soubory entit)

Struktura databáze se poté dá poměrně snadno odvodit z definičních souborů. To jediné, v čem vývojářem omezit nemůžeme jsou dotazy do databáze, tam má vývojář stále plnou kontrolu v podobě definování vlastních komponent a funkcí. Velkou výhodou je také hlídání změn, kdykoli přibude či zmizí nějaký datový vstup z definičního souboru, generátor tuto skutečnost rozpozná a vygeneruje příslušné **migrační soubory**, které zpracovává framework Laravel.

Migrační soubory, které provedli změny v databázi zůstávají zachovány a obsahují i funkci, která strukturu vrátí do původního stavu, proto můžeme libovolně přecházet zpět na původní verze struktury databáze. Pozor je třeba si dát pouze u dat, strukturu vzít zpět lze, ale data již nikoli. Například pokud byl odstraněn nějaký databázový sloupec, můžeme ho samozřejmě vrátit do struktury zpět, ale data, která v něm byla uložena k jednotlivým záznamům již bohužel zpět nedostaneme, k tomu pak slouží pravidelné zálohy, které jsou samozřejmě dobrou praxí, ale to už spíš úkol správce databázového serveru.

Více o migračních souborech Laravelu zde: <https://laravel.com/docs/6.x/migrations>

9.1 Princip funkce generátoru

Generátor byl vytvořen jako globální Middleware^[10] ve frameworku Laravelu. Řídí se podle definičních souborů jednotlivých entit. Kontroluje strukturu databáze a upravuje jí na základě změn v definičních souborech. Tím je usnadněna podstatná část programátorovi práce, jelikož se nemusí starat o změny ve struktuře databáze. Zároveň generuje soubory s modely entit, do kterých může vývojář následně přidávat funkce.

Jak již bylo řečeno generátor ve frameworku existuje jako takzvaný Middleware. Middleware je funkční celek Laravelu, který probíhá ještě před vlastním předáním http dotazu jádru frameworku a controllerům,^[10] tedy z pohledu uživatele před zobrazovací funkcí stránky. V praxi je zde řešena funkcionalita jako je iniciace session, přihlašování uživatelů, apod. V našem případě, generátor hlídá integritu aplikace, strukturu databáze a generuje případné chybějící soubory.

Generátor při každém dotazu do aplikace provádí postupně, v návaznosti tyto úkony:

1. Načte **definiční soubory** a zkontroluje jejich syntaxi
2. Pokud chybí soubory s **modely entit**, vytvoří je.
3. Načte si **strukturu databáze** a porovná ji s definičními soubory
4. V případě rozdílu mezi strukturou databáze a definičními soubory vytvoří potřebné **migrační soubory**, které lze spustit a tím strukturu databáze upravit.
5. V případě rozdílu mezi strukturou databáze a definičními soubory, zastaví aplikaci, upozorní vývojáře na tyto změny a vyzve ho ke spuštění **migračních souborů**.
6. V případě, že vývojář dal svolení ke spuštění migračních souborů, **spustí je**.

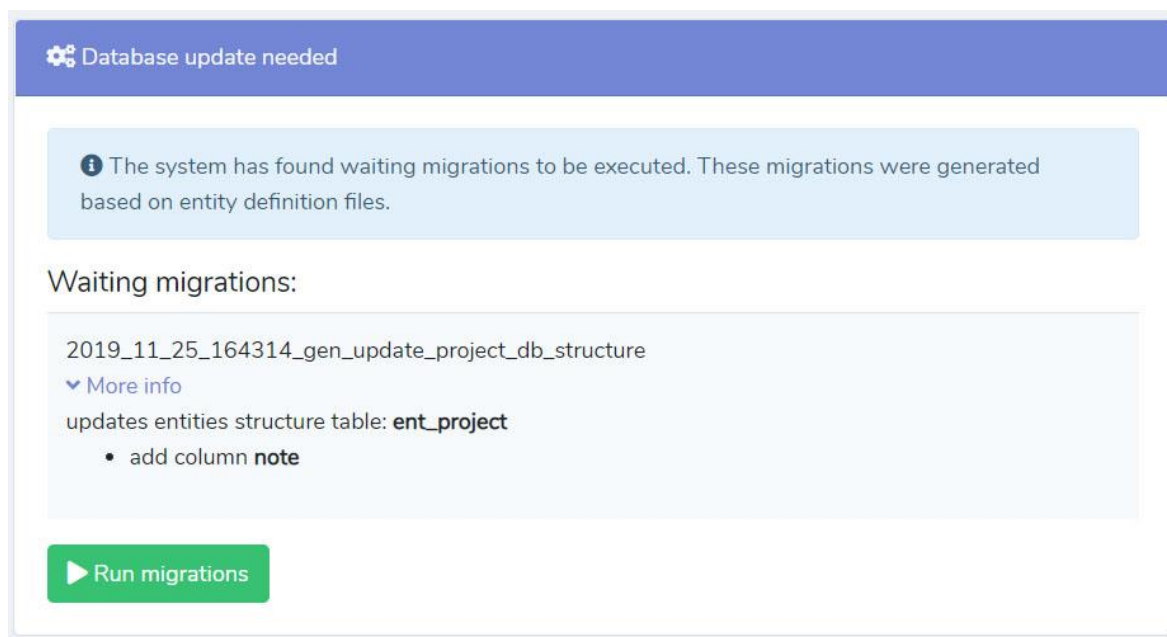
9.2 Modelová situace: Hlídání struktury a integrity

Nejlepší způsob, jak vysvětlit funkci generátoru je na názorném příkladu. Máme definiční soubor projektů, řekněme, že chceme přidat nový datový vstup, například interní poznámku. V definičním souboru tedy do vstupního formuláře přidáme příslušný datový vstup, jako na výpisu z programu č. 18 níže.

```
"note": {  
  "title": "Poznámka",  
  "type": "text",  
  "filterable": false,  
  "display_in_listings": false  
}
```

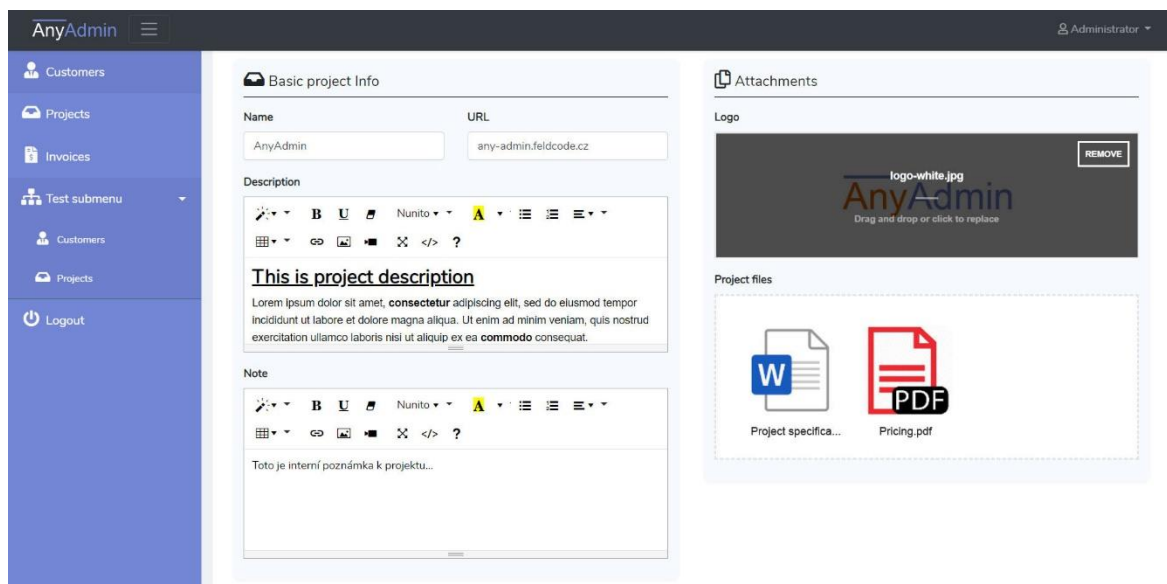
Výpis 18: Vložení datového vstupu „Poznámka“ do definičního souboru zákazníků

Jakmile se vrátíme do aplikace, vidíme, že generátor, tuto skutečnost zaznamenal (na obrázku č. 13 níže) a již nám připravil migrační soubor, který změní strukturu databáze. Jakmile migrační soubor spustíme, zůstane zachován, aby bylo možné kdykoli přejít na předešlé verze databázové struktury.



Obrázek 13: Upozornění generátoru na potřebné migrace – přidání sloupce „note“ k entitě projektů

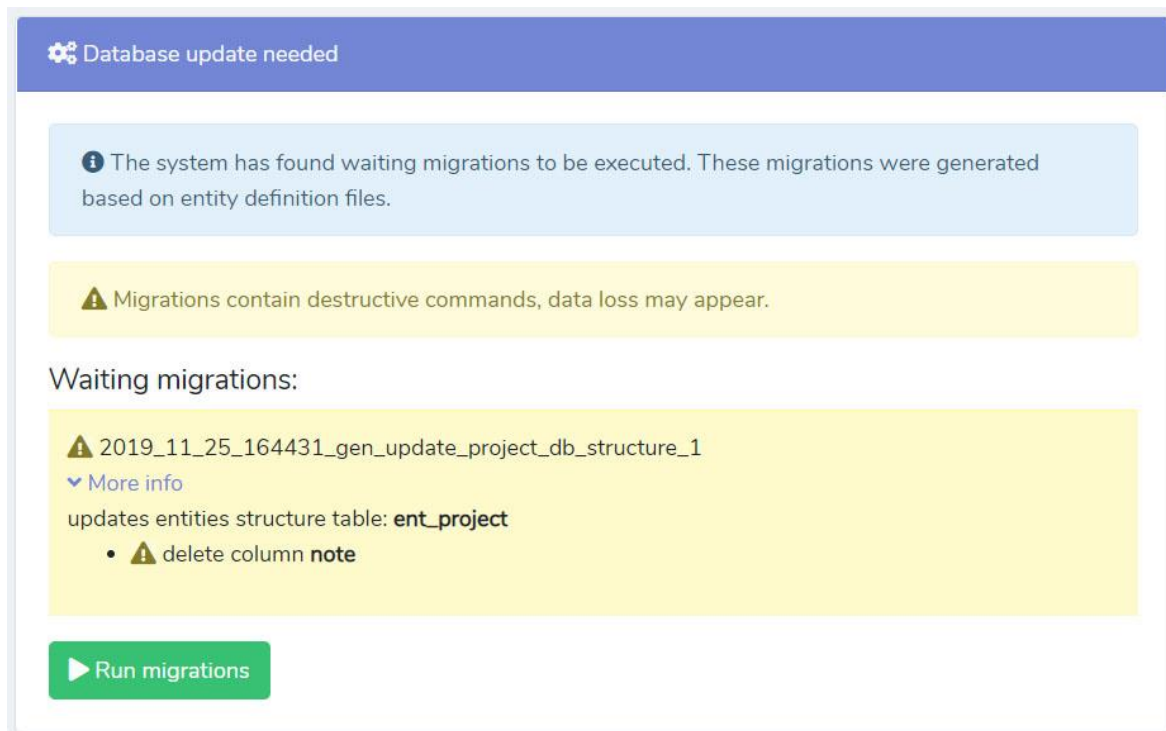
Po spuštění potřebných migrací je aplikace opět dostupná a poznámku je nyní možné v editačním formuláři projektu vyplnit.



Obrázek 14: Nový datový vstup je nyní dostupný ve formuláři

Nyní si změny rozmyslíme a sloupeček s poznámkou z definičního souboru opět odstraníme. Výsledkem je opět generátor upozorňující na nové migrace, nyní ale se zvláštním varováním. Chystáme se totiž na zásah do struktury databáze, který má destrukční dopad na data. Strukturu úspěšně změníme, ale o všechna data, která jsme předtím měli uložená ve sloupečku „note“, tedy všechny poznámky k zákazníkům, tímto úkonem přijdeme. Je tedy na

vývojáři, aby před změnou data řádně zálohoval. Upozornění na operace s destrukčním charakterem je vidět na obrázku č. 15 níže.



Obrázek 15: Upozornění generátoru na potřebné migrace – odstranění sloupce „note“ z entity projektů

9.3 Pevné části databázových tabulek entit

Každá tabulka, která je vytvořená pro konkrétní entitu, obsahuje spolu se sloupce datových vstupů z definičních souborů také pevně dané sloupce, tyto sloupce jsou:

- **id**
ID záznamu, je primárním klíčem tabulky a má nastavený Auto Increment
- **user_id**
ID uživatele, který záznam vytvořil
- **created_at**
Datum vytvoření záznamu
- **updated_at**
Datum upravení záznamu

9.4 Relace

Generátor hlídá také relační vztahy mezi jednotlivými entitami. Podle typu datového vstupu automaticky rozpozná, o jaký vztah se jedná a v případě vztahu vyžadující pomocné tabulky je automaticky vytvoří.

Typy datových vstupů vytvářející vztah jsou:

- select
- select_inv
- multiselect
- multiselect_inv

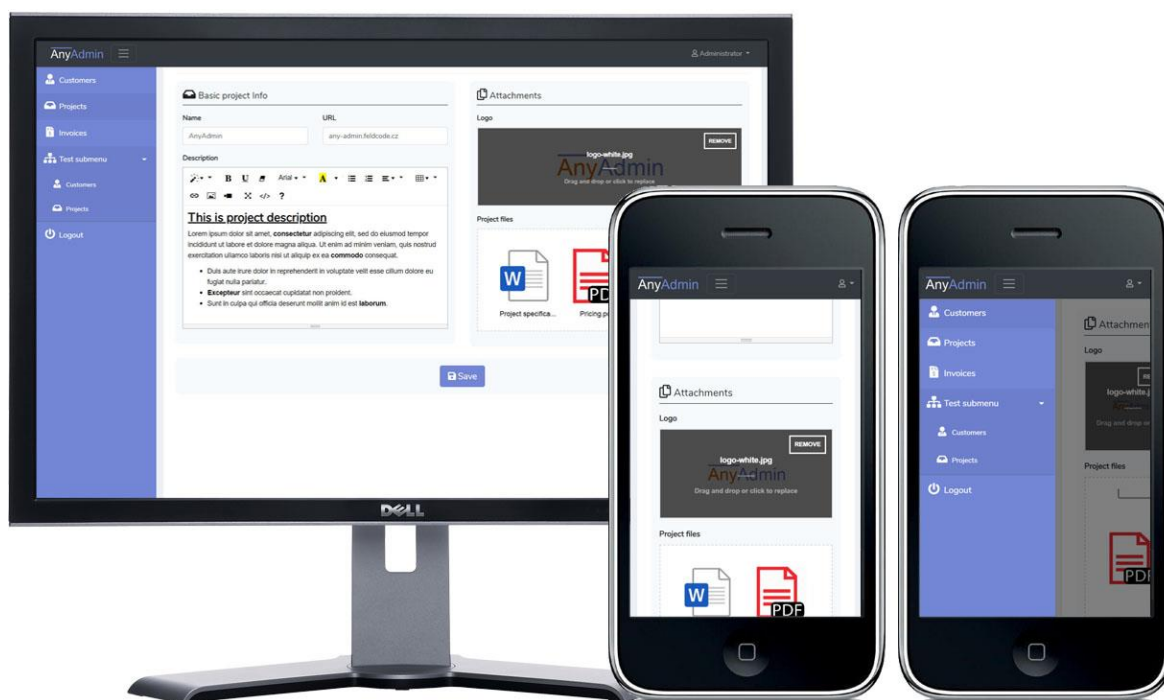
Jak se tyto typy datových vstupů chovají a co znamenají je popsáno v kapitole 3.1.6 Datový vstup (_input)

10 Uživatelské prostředí

Uživatelské prostředí aplikace využívá nástrojů front-endového frameworku Bootstrap a sestává z množství pluginů, které různým způsobem zpřehledňují aplikaci nebo mění uživatelské vstupy, například pro výběr datumu či barvy ve formulářích, kompletní výčet těchto pluginů je k nalezení v kapitole 11.2.

10.1 Základní design

Základní design aplikace je vytvořený tak, aby stejně jako zbytek aplikace dodržoval zásady jednoduchosti. Opět zde platí, že i vzhled je v plné kontrole vývojáře. Framework využívá nástroje Laravel mix a technologie SASS. Proto vývojář může libovolně měnit například barvy šablony. Vše vývojář upravuje tak, jak je zvyklý z Laravelu. Základní responzivní návrh designu aplikace je znázorněn na obrázku č. 16. Z něj může vývojář vycházet, ale může ho samozřejmě i měnit. Důležité je, aby měl na paměti výchozí konceptuální šablonu aplikace (viz kapitola 2.1).



Obrázek 16: Základní návrh designu aplikace

10.2 Responzivita

Kterákoli aplikace vytvořená frameworkem AnyAdmin je od počátku responzivní. Responzivita je zajištěna front-endovým frameworkem Bootstrap a jeho mřížkovým systémem. CSS třídy mřížkového systému lze také zapisovat do definičních souborů k jednotlivým datovým vstupům a vstupnímu formuláři jako takovému. Tím je zajištěna velmi snadná manipulace se vstupy a přehledností formulářů.

10.3 Jazykové mutace

Framework samozřejmě podporuje také jazykové mutace, aplikaci lze tedy vyvíjet v jakémkoli jazyce. Jelikož framework vychází z frameworku Laravel a používá k validaci vstupů právě tento framework, veškerá výchozí chybová hlášení a texty z aplikace jsou dostupná ve všech jazycích, které Laravel podporuje, včetně češtiny. V případě, že jazyk Laravel nepodporuje, musí vývojář doplnit příslušné soubory sám. Veškerý ostatní obsah definuje vývojář v definičních souborech. Vše, co vývojář do definičních souborů zapíše a zobrazuje se jako text v aplikaci, prochází funkcí překladu. Proto stačí vytvořit příslušný soubor s jazykovou mutací a do něho zanést překlady, převážně jde o názvy entit a tlačítek.

Adresář se soubory překladu: „resources/lang/název_jazyka“

Příklad použití souboru s překladem do češtiny je vidět na výpisu z programu č. 19 níže. Takto jednoduše přeložíme řetězců textu. Pokud tedy ve výpisu z programu č. 19 máme entitu nazvanou „Customer“ a máme nastavenou češtinu, v aplikaci se zobrazí jako „Zákazník.“ Pokud žádný odpovídající řetězcem není nalezen, zobrazí se výchozí, tedy „Customer.“ Samozřejmě je lepší pro mapování používat nějaké unikátní řetězce a přeložit je i pro zdrojový jazyk. Například „Customer“, by mohl být: „_CUSTOMER“ a v definičním souboru by se na něj muselo také takto odkazovat, pro příklad to ale stačí.

Více o jazykových souborech Laravelu: <https://laravel.com/docs/6.x/localization>

```
1  <?php
2
3  return [
4      'Customer' => 'Zákazník',
5      'Customers' => 'Zákazníci',
6      'Invoice' => 'Faktura',
7      'Invoices' => 'Faktury',
8      'Save' => 'Uložit',
9  ];
```

Výpis 19: Soubor s překladem do češtiny

11 Využité technologie

Jak již bylo řečeno, vybrat správné technologie, ve smyslu programovacích jazyků, pluginů a frameworků bylo jednou z hlavních výzev práce. Při jejich výběru jsem spoléhal na vlastní know-how a oblíbenost v řadách vývojářské komunity. Zanalyzovat všechny dostupné nástroje na konkrétní úlohy by totiž vyžadovalo obrovské úsilí a hromadu času. Rozdíly mezi nimi jsou však takřka mizivé. Proto není podstatné je zdlouhavě porovnávat. To, že je využívá developerská komunita a nejčastěji se na ně odkazuje považuji za dostatečné ověření. Níže jsou tedy popsány technologie, které byli při vývoji frameworku použity. Jsou to technologie obecně **nejznámější a nejvíce preferované vývojáři webových stránek**, proto framework jako takový by měl běžnému vývojáři webových stránek, který pracuje v Laravelu připadat povědomí a měl by se v něm **velice rychle a snadno zorientovat**.

Využité technologie jsem rozdělil do tří kategorií:

1. Back-endové
2. Front-endové
3. Podpůrné

Back-endové a front-endové technologie spočívají v technologiích mající přímý dopad na výsledek aplikace, řadím sem programovací jazyky, Frameworky, knihovny a Pluginy. Jaký je mezi nimi rozdíl nemusí být na první pohled jasné, proto si je vysvětlíme.

- **Programovací jazyky**
Tady je to zřejmé. Jedná se o jazyky, ve kterém je napsaný kód jako takový.
- **Frameworky**
Frameworky poskytují vývojáři sadu nástrojů a pomáhají mu nějakým způsobem uchopit vytvářené dílo, slouží jako podklad vytvářeného díla. Odstraňují repetitivní úkony a vývojáři umožňují se zaměřit na vyšší úroveň práce.
- **Knihovny**
Knihovny stejně jako frameworky poskytují vývojáři sadu nástrojů na řešení dílčích problémů, ale méně zasahují do práce vývojáře. Neslouží jako podklad, ale spíše jako pomocník.
- **Pluginy**
Pluginy jsou hotové celky, které do aplikace lze vložit a pomocí kódu s nimi komunikovat. Typickým příkladem je Bootstrap DateTimePicker. V kódu stačí přidat k tagu „input“ CSS třídu a z výchozího vstupního políčka je rázem sofistikované tlačítko pro výběr datumu.

Do **podpůrných technologií** řadím programy a aplikace pomáhající při vlastním vývoji, ale nemajícící přímý dopad do kódu. Jde například o kompilátory kódu.

11.1 Back-endové jazyky a nástroje

11.1.1 Programovací jazyky a databáze

PHP

„PHP (rekurzivní zkratka PHP: Hypertext Preprocessor, česky „PHP: Hypertextový preprocesor“, původně Personal Home Page) je skriptovací programovací jazyk. Je určený především pro programování dynamických internetových stránek a webových aplikací například ve formátu HTML, XHTML či WML. PHP lze použít i k tvorbě konzolových a desktopových aplikací. Pro desktopové použití existuje kompilovaná forma jazyka.“

[Wikipedia 2019: 11]

MySQL (databáze)

„MySQL je systém řízení báze dat uplatňující relační databázový model, vytvořený švédskou firmou MySQL AB, nyní vlastněný společností Oracle Corporation. Jeho hlavními autory jsou Michael „Monty“ Widenius a David Axmark. Je považován za úspěšného průkopníka dvojího licencování – je k dispozici jak pod bezplatnou licenci GPL, tak pod komerční placenou licenci.“

MySQL je multiplatformní databáze. Komunikace s ní probíhá – jak už název napovídá – pomocí jazyka SQL. Podobně jako u ostatních SQL databází se jedná o dialekt tohoto jazyka s některými rozšířeními.“

[Wikipedia 2019: 12]

11.1.2 Frameworky

Laravel

K zajištění backendové části webu nám velmi dobře poslouží framework Laravel. Jde o nejrozšířenější a nejoblíbenější framework mezi vývojáři PHP. Vychází z frameworku Symfony a obsahuje celou řadu pluginů a nástrojů usnadňující práci vývojářů^[3]. Pokrývá v podstatě veškeré výzvy, kterým programátor čelí. Poskytuje sadu nástrojů na řešení různých dílčích problémů od šablonovacích nástrojů (zde využívá například nástroj Blade^[3]) až po práci s databází a modely (k tomu využívá např. Eloquent^[3]).

11.1.3 Pluginy

Eloquent Dynamic Relation ^[13]

Eloquent Dynamic Relation je open-source plugin od vývojáře Rasel Rana Rocky z Bangladéše vystupujícím pod přezdívkou i-rocky na GitHubu. Jde o plugin, který umožňuje dynamicky (myšleno pomocí kódu) přidávat vztahy k Eloquent modelům v Laravelu. Frameworku tento plugin umožňuje automaticky namapovat vztahy mezi jednotlivými entitami. V Laravelu se totiž vztahy definují ručně, vytvořením metody příslušného modelu. Více o vztazích modelů Laravelu zde: <https://laravel.com/docs/6.x/eloquent-relationships>

DOMPDF Wrapper for Laravel 5 ^[14]

DOMPDF Wrapper for Laravel 5 je open-source plugin od vývojáře Barry vd. Heuvel z Nizozemí vystupujícím pod přezdívkou barryvdh na GitHubu. Jde o plugin, který velice šikovně převádí HTML kód na PDF. Zároveň integruje architekturu Laravelu, tím je možné v podstatě jakoukoli část systému (např. tabulku) zobrazit jako PDF. Frameworku slouží jako podpůrný nástroj exportu dat.

11.2 Front-endové jazyky a nástroje

11.2.1 Programovací jazyky

HTML5

„HTML5 je v informatice verze značkovacího jazyka HTML sloužícího pro tvorbu webových stránek. Finální specifikace byla vydána 28. října 2014. Proti předchozí verzi HTML4 z roku 1997 přináší podstatné změny, přičemž mezi nejdůležitější patří přímá podpora přehrávání multimédií v prohlížeči a podpora pro aplikace, které fungují i bez připojení k Internetu.“

[Wikipedia 2019: 15]

HTML popisuje **strukturu dokumentu**.

CSS3

„Kaskádové styly (v anglickém originále Cascading Style Sheets se zkratkou CSS) je v informatice jazyk pro popis způsobu zobrazení elementů na stránkách napsaných v jazycích HTML, XHTML nebo XML.

Jazyk byl navržen standardizační organizací W3C, autorem prvotního návrhu byl Håkon Wium Lie. Byly vydány CSS1, CSS2 a CSS3. Dne 7. června 2011 byla dokončena revize CSS 2.1 Hlavním smyslem je umožnit návrhářům oddělit vzhled dokumentu od jeho struktury a obsahu. Původně to měl umožnit už jazyk HTML, ale v důsledku nedostatečných standardů a konkurenčního boje výrobců prohlížečů se vyvinul jinak. Starší verze HTML obsahují celou řadu elementů, které nepopisují obsah a strukturu dokumentu, ale i způsob jeho zobrazení. Z hlediska zpracování dokumentů a vyhledávání informací není takový vývoj žádoucí.“

[Wikipedia 2019: 16]

CSS dává jednotlivým tagům HTML styly, řeší **vzhledovou podobu**.

JavaScript

„JavaScript je multiplatformní, objektově orientovaný, událostmi řízený skriptovací jazyk, jehož autorem je Brendan Eich z tehdejší společnosti Netscape. Jeho syntaxe (zápis zdrojového textu) patří do rodiny jazyků C/C++/Java, ale JavaScript je od těchto jazyků zásadně odlišný, sémanticky (funkčně, principiálně) jde o jiný jazyk. Slovo Java je součástí jeho názvu pouze z marketingových důvodů. JavaScript byl v červenci 1997 standardizován asociací ECMA (European Computer Manufacturers Association) a v srpnu 1998 ISO (International Organization for Standardization). Standardizovaná verze JavaScriptu je pojmenována ECMAScript a z ní byly odvozeny i další implementace, jako je například ActionScript. JavaScript byl původně obchodní název implementace společnosti Netscape, kde byl vyvíjen nejprve pod názvem Mocha, později LiveScript, ohlášen byl společně se společností Sun Microsystems v prosinci 1995 jako doplněk k jazykům HTML a Java. Pro verzi firmy Microsoft je použit název JScript. Ten je podporován platformou .NET.“

[Wikipedia 2019: 17]

I když je JavaScript multiplatformní používá se nejčastěji právě ve vývoji webových stránek.^[17] JavaScript dodává webovému rozhraní možnost **interakce**. Díky JavaScriptu nemusí být stránka statická, může implementovat různé animace, ale i asynchronně komunikovat z back-endovou částí webu bez nutnosti načíst znovu stránku.

11.2.2 Frameworky a knihovny

Bootstrap ^[18]

Bootstrap je open-source framework pro vývoj front-endové části webových stránek na technologiích HTML, CSS a JavaScript. Poskytuje celou řadu nástrojů, které obsahují výchozí vzhled různých prvků, jako jsou tlačítka, tabulky, apod. Zároveň v sobě obsahuje důmyslný mřížkový systém, který řeší problematiku responzivity.

jQuery ^[19]

„jQuery je javascriptová knihovna s širokou podporou prohlížečů, která klade důraz na interakci mezi JavaScriptem a HTML. Byla vydána Johnem Resigem v lednu 2006 na newyorském BarCampu. jQuery je svobodný a otevřený software pod licencí MIT.“

[Wikipedia 2019: 20]

AJAX

„AJAX (Asynchronous JavaScript and XML) je v informatice obecné označení pro technologie vývoje interaktivních webových aplikací, které mění obsah svých stránek bez nutnosti jejich kompletního znovunačítání za pomoci asynchronního zpracování webových stránek pomocí knihovny napsané v JavaScriptu. Na rozdíl od klasických webových aplikací poskytují uživatelsky příjemnější prostředí, ale vyžadují použití moderních webových prohlížečů.“

[Wikipedia 2019: 21]

Technologii AJAX implementuje knihovna jQuery.^[19]

11.2.3 Pluginy

Dropify ^[22]

Dropify je plugin, který přetváří zobrazení klasického HTML tagu „input“ typu „file“ (upload souboru) na uživatelsky přívětivější verzi. Porovnání klasického datového vstupu a přetvořeného pomocí pluginu Dropify je patrné z následujících obrázků č. 17 a 18.

Basic HTML input file

Vybrat soubor logo-white.jpg

Obrázek 17: Výchozí zobrazení HTML tagu „input“ typu „file“ v prohlížeči Google Chrome



Obrázek 18: Zobrazení HTML tagu „input“ typu „file“ pomocí pluginu Dropify

Dropzone.js ^[23]

Dropzone.js je plugin, který přetváří klasický formulář pro upload více souborů na uživatelsky přívětivější verzi. Přetváří formulář s tagy na vyhrazené místo, do kterého může uživatel přetahovat soubory, umožňuje upload více souborů najednou a při odeslání formuláře simuluje tagy „input.“ Soubory lze nahrát i kliknutím do tohoto prostoru. Ve vyskočeném dialogovém okně pak lze souborů vybrat více, na rozdíl od klasického tagu „input.“

Porovnání klasického formuláře, kdy musíme pro každý soubor zvlášť v kódu vytvořit tag „input“ typu „file,“ a přetvořeného formuláře pomocí pluginu Dropzone.js je patrné z obrázků č. 19 a 20 níže.

Basic HTML input file

Vybrat soubor	Soubor nevybrán
Vybrat soubor	Soubor nevybrán
Vybrat soubor	Soubor nevybrán

Obrázek 19: Výchozí podoba formuláře s několika tagy „input“ typu „file“ v prohlížeči Google Chrome



Obrázek 20: Zobrazení formuláře po aplikaci pluginu Dropzone.js

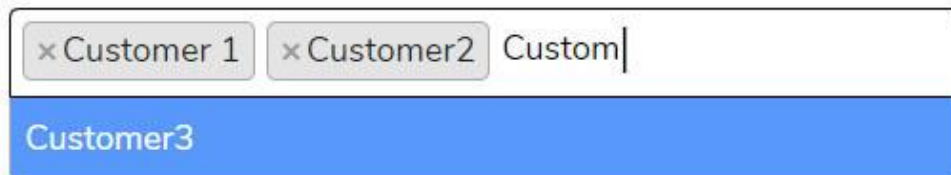
Select2 ^[24]

Select2 je plugin, který přetváří klasické zobrazení HTML tagu „select“ (výběr možností) na uživatelsky přívětivější verzi. Porovnání klasického datového vstupu a přetvořeného pomocí pluginu Select2 je patrné z obrázků č. 21 a 22 níže.

Nejvýrazněji plugin Select2 pomáhá u tagu „select“ s příznakem „multiple.“ Vstup se zobrazí jako klasický tag „input,“ pokud do něj uživatel klikne, vyskočí kontextové menu, ze kterého může vybírat, také může začít psát a v možnostech tak vyhledávat, což je velice šikovná funkce.

Basic HTML select multiple

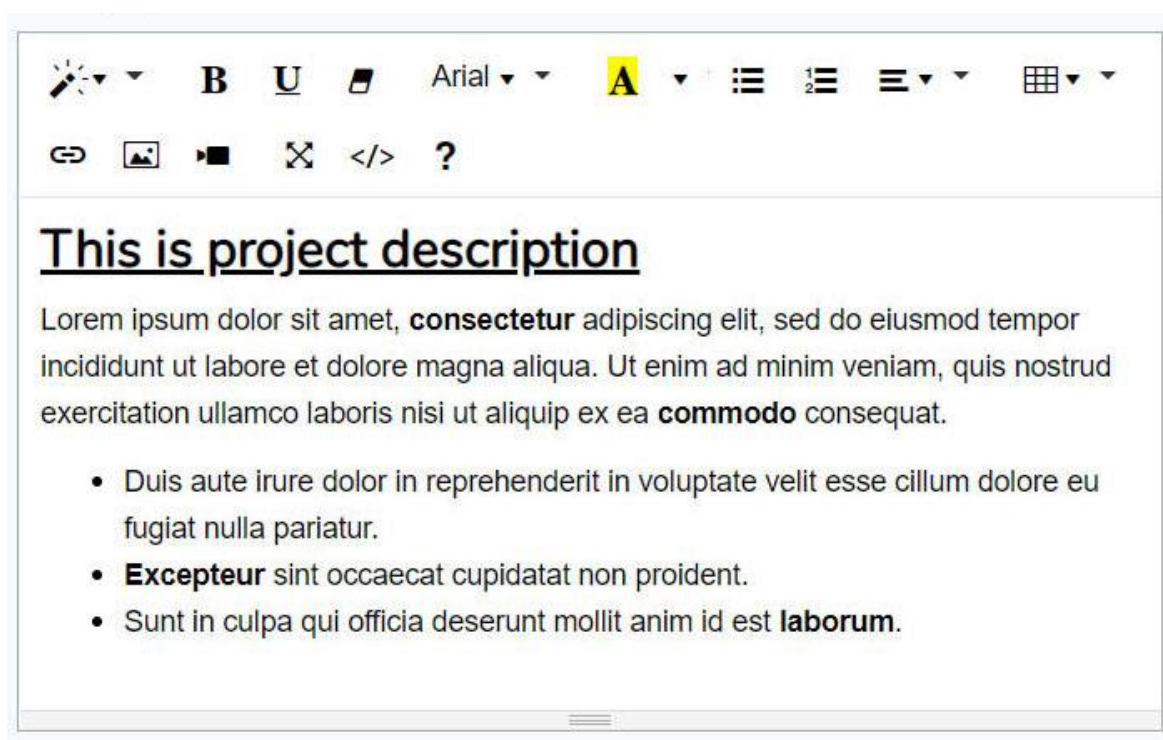
Obrázek 21: Výchozí zobrazení HTML tagu „select“ s atributem „multiple“ v prohlížeči Google Chrome



Obrázek 22: Zobrazení HTML tagu „select“ s atributem „multiple“ po aplikaci pluginu Select2

Summernote ^[25]

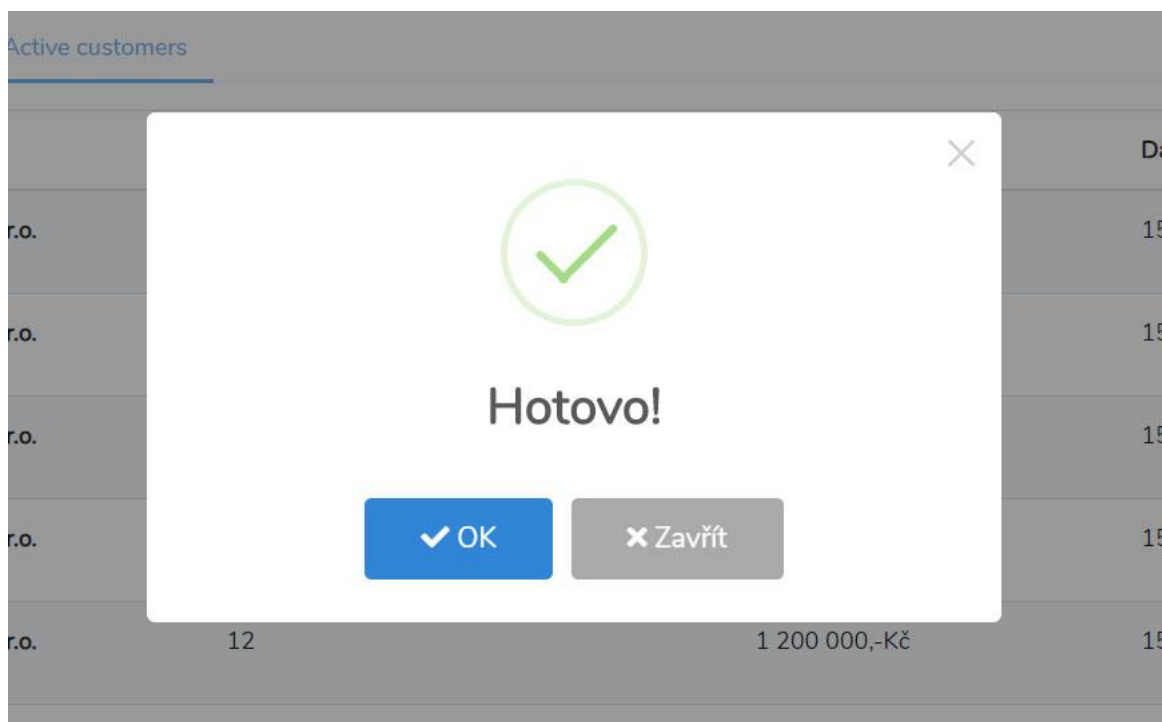
Plugin summernote přetváří klasické zobrazení HTML tagu „textarea“ na textový editor, uživatel tak může text snadno formátovat. Výsledná hodnota odeslaná ve formuláři obsahuje formátovaný text pomocí HTML značek. Textový editor Summernote je k nahlédnutí na obrázku č. 23 níže.



Obrázek 23: Textový editor – Plugin Summernote

SweetAlert2 ^[26]

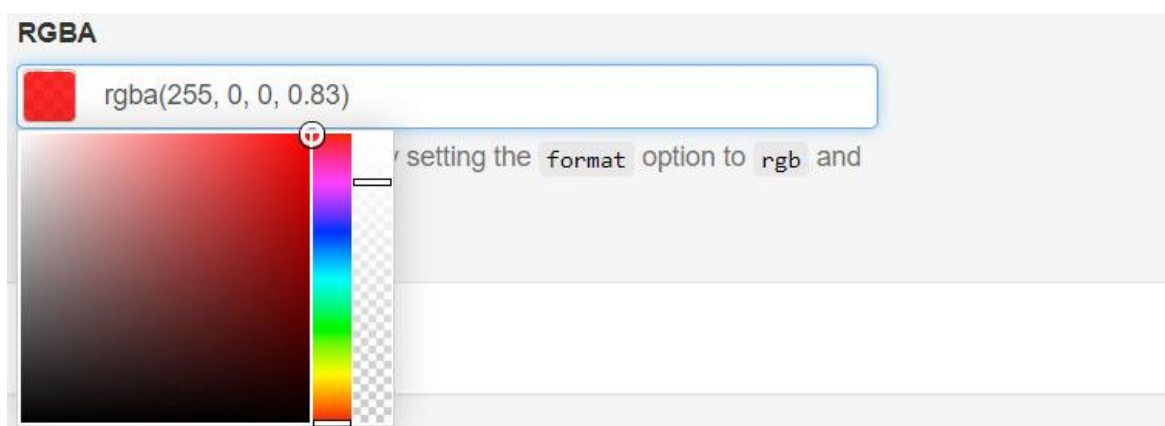
Plugin SweetAlert vytváří dialogové okno, do kterého je možné vkládat obsah ve formátu HTML a vyvolávat akce po kliknutí na tlačítka v okně. Plugin využívá animací a působí uživatelsky dobrým a hravým dojmem. Ve frameworku je použit jako dialogové okno v případě potvrzení akce uživatelem, jako informační okno nebo jako průvodce spuštěním funkce, jak bylo demonstrováno na příkladu v kapitole 7.4. Interakce s uživatelem. Názorná ukázka pluginu sweetAlert2 je viditelná na obrázku č. 24 níže.



Obrázek 24: Dialogové okno – plugin SweetAlert2

jQuery MiniColors ^[27]

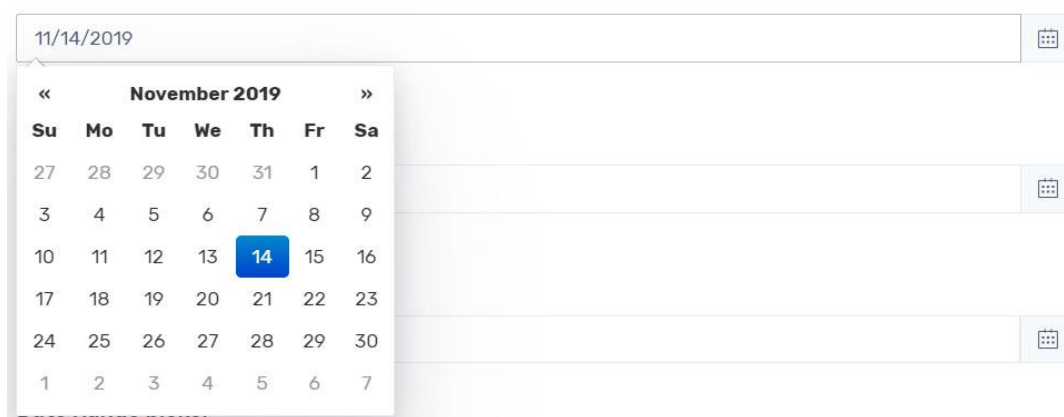
Plugin jQuery MiniColors přidává ke klasickému HTML tagu „input“ typu „text“ (datový vstup pro řetězec znaků) kontextové okno s možností výběru barvy. Datový vstup s aplikovaným pluginem jQuery MiniColors je viditelný na obrázku č. 25 níže.



Obrázek 25: HTML tag „input“ s možností výběru barvy – plugin jQuery MiniColors

Bootstrap DateTimePicker ^[28]

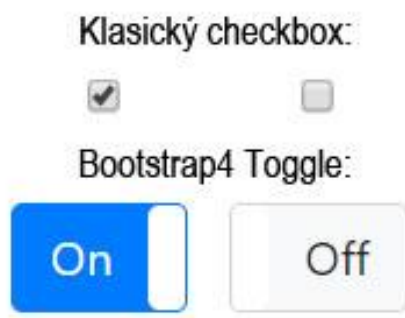
Plugin Bootstrap DateTimePicker přidává ke klasickému HTML tagu „input“ typu „text“ (datový vstup pro řetězec znaků) kontextové okno s možností výběru datumu a času. Datový vstup s aplikovaným pluginem Bootstrap DateTimePicker je viditelný na obrázku č. 26 níže.



Obrázek 26: HTML tag „input“ s možností výběru datumu – plugin Bootstrap DateTimePicker

Bootstrap4 Toggle ^[29]

Plugin Bootstrap4 Toggle přetváří HTML tag „input“ typu „checkbox“ (datový vstup pro typ boolean) na vypínač. Porovnání klasického datového vstupu a datového vstupu s aplikovaným pluginem Bootstrap4 Toggle je viditelný na obrázku č. 27 níže.



Obrázek 27: Porovnání klasického tagu „input“ typu „checkbox“ a pluginu Bootstrap4 Toggle

11.3 Podpůrné technologie

Jelikož i podpůrné programy nechávají ve výsledné aplikaci své stopy je vhodné je také uvést zde v práci. Jedná se především o verzovací programy či balíčkovací programy hlídající a spravující vzájemné závislosti a propojení jednotlivých pluginů mezi sebou.

11.3.1 Kompilovatelné extenze programovacích jazyků

Sass ^[30]

Sass je extenzí jazyka CSS. Jazyku CSS dodává funkce, které usnadňují práci a jsou velice užitečné při spravování šablony webu. Jednou z nejužitečnějších funkcí je možnost definovat proměnné, což je užitečné například při použití barev, velikosti odsazení, apod. Oproti klasickému CSS, kde barvy musí být definované explicitně každé třídě, tak může vývojář změnit barvu pouze na jednom místě a tím se změní celá barevnost šablony. Mezi další funkce pak patří importování a využití modulů nebo vnořování a dědičnost jednotlivých CSS tříd.

11.3.2 Verzovací programy

Git ^[31]

Git je open-source software spravující verze nejrůznějších aplikací. Frameworku zajišťuje kontrolu nad zdrojovým kódem a jednotlivými upgrady a zároveň slouží jako distribuční software. Framework je možné si za pomoci Gitu stáhnout jedním příkazem v příkazové řádce, jak je popsáno v kapitole 14 (SW nároky a zprovoznění frameworku).

Stažení zdrojového kódu frameworku z příkazové řádky:

```
> git clone https://gitlab.com/FeldCode/any-admin.git
```

11.3.3 Dependency Managers

Composer ^[32]

Composer je software spravující a hlídající vzájemné závislosti různých verzí použitých technologií v jazyce PHP. Spravuje tedy závislosti v back-endové části aplikace. Zároveň slouží k snadné instalaci frameworků, knihoven či pluginů z příkazové řádky.

Příklad instalace jedné z komponent frameworku pomocí Composeru:

```
> composer require i-rocky/eloquent-dynamic-relation
```

Node.js ^[33]

Node.js je JavaScriptový asynchronní nástroj k tvorbě škálovatelných síťových aplikací.^[33] I když ve frameworku Node.js nepoužívám. Node.js obsahuje balíčkovací software **npm** spravující a hlídající vzájemné závislosti různých verzí použitých technologií na front-endové části aplikace. Stejně jako Composer, také slouží jako instalátor technologií z příkazové řádky.

Příklad instalace jedné z komponent frameworku pomocí Node.js:

```
> npm install summernote --save
```

12 Případová studie

Jako modelovou situaci si zkusíme v nástroji vytvořit jednoduchý fakturační systém se správou zákazníků a projektů. Na uvedeném příkladu si ukážeme, jak lze jednoduše, během několika minut vytvořit plně funkční fakturační systém, ze kterého vývojář může vycházet.

12.1 Zadání

Jsme malá firma, zabývající se tvorbou webových stránek a aplikací. Naším cílem je vytvořit jednoduchý fakturační systém spolu se správou zákazníků a k nim přiřazených projektů.

12.1.1 Základní entity

Náš příklad bude obsahovat tyto základní entity, které mezi sebou budou navzájem provázané:

- Zákazník (customer)
- Kontaktní osoby na zákazníka (contact_person)
- Realizované projekty (project)
- Faktury (invoice)
- Způsoby platby, číselník pro faktury (payment_type)

12.1.2 Vztahy mezi entitami

Nyní si nastíníme vztahy mezi jednotlivými entitami. K Zákazníkům máme přiřazené kontaktní osoby. Každý projekt, který realizujeme patří ke konkrétnímu zákazníkovi. A nakonec zákazníkovi vystavujeme faktury, které mohou být zaplacený nějakým způsobem platby. Toto jednoduché schéma je patrné z obrázku č. 28 níže.



Obrázek 28: Schéma zadání případové studie

12.2 Implementace

12.2.1 Konfigurace aplikace

Konfigurace aplikace spočívá v nastavení hlavního definičního souboru app.json. Nastavíme název aplikace, logo a hlavní menu. Logo a název pro účely demonstrace změníme z „AnyAdmin“ na „MyAdmin.“ Hlavní definiční soubor tedy může vypadat takto:

```
{
  "app_name": "MyAdmin",
  "logo": "img/logo-my-admin.png",
  "favicon": "img/favicon.png",
  "home_page": "customers",
  "menu": [
    {"target": "customers"},
    {"target": "projects"},
    {"target": "invoices"}
  ]
}
```

Výpis 20: Případová studie – Hlavní definiční soubor app.json

12.2.2 Nadefinování jednotlivých stránek

Nyní si nadefinujeme jednotlivé stránky, tedy sekce, na které odkazují položky menu z hlavního definičního souboru. Nebudu zde uvádět všechny stránky, pro příklad stačí ukázka definičního souboru se stránkou zákazníků. Přiřadíme stránce název a ikonu, dále položky menu s komponentou tabulky, kde se mají zákazníci zobrazit, a nakonec tlačítko pro přidání nového zákazníka. Jak takový definiční soubor vytvořit, jsme si představili v kapitole 3.4 Definiční soubory stránek. Definiční soubor stránky se zákazníky tedy bude vypadat takto:

```
{
  "title": "Customers",
  "icon": "fas fa-user-tie",
  "menu": [
    {
      "title": "All customers",
      "icon": "",
      "action": {
        "type": "component",
        "target": "table",
        "data": {
          "entity": "customer"
        }
      }
    }
  ],
  "buttons": [
    {
      "title": "New customer",
      "icon": "fa fa-plus-square",
      "bs_color": "success",
      "action": {
        "type": "component",
        "target": "form",
        "data": {
          "entity": "customer"
        }
      }
    }
  ]
}
```

Výpis 21: Případová studie – Definiční soubor stránky se zákazníky

12.2.3 Nadefinování jednotlivých entit a jejich vztahů

Nyní zbývá poslední krok, vytvoříme definiční soubory pro jednotlivé entity ze zadání. Nejprve si promyslíme, jaké data budeme, u které entity vyžadovat a následně je ve formě datových vstupů zapíšeme do definičního souboru, tak jak jsme si ukázali v kapitole 3.3 Definiční soubory entit. Jednotlivé definiční soubory tedy budou obsahovat tyto datové vstupy:

Zákazník (customer.json)

- **Základní informace (sekce formuláře)**
 - Název (name)
 - Adresa sídla (address)
 - IČ (ident_number)
 - DIČ (vat_id)
- **Kontaktní informace (sekce formuláře)**
 - Seznam kontaktních osob (contact_person - vztah s entitou **contact_person**)

Projekt (project.json)

- Název (name)
- URL adresa webové stránky (url)
- Realizováno pro zákazníka (customer - vztah s entitou **customer**)
- Popis (description)
- Soubory k projektu, dokumentace (files – uploader souborů)

Faktura (invoice.json)

- Číslo faktury (no)
- Zákazník (customer – vztah s entitou **customer**)
- Název odběratele (name)
- Adresa sídla odběratele (address)
- IČ (ident_number)
- DIČ (vat_id)
- Datum vystavení (invoice_date)
- Datum splatnosti (due_date)
- Částka (amount)
- Způsob platby platby (payment_type – vztah s entitou **payment_type**)

Opět si myslím, že není nutné vypisovat podobu všech definičních souborů, ale na následujícím výpisu z programu č. 22 si ukážeme pouze definiční soubor entity zákazníků.

```
{
  "title": "Customer",
  "icon": "fa fa-user-tie",
  "title_field": "name",
  "input_form": {
    "main_info": {
      "title": "Main Information",
      "icon": "fa fa-user-tie",
      "inputs": {
        "name": {
          "title": "Name",
          "type": "string",
          "rules": "required"
        },
        "address": {
          "title": "Address",
          "type": "text",
          "rules": "required"
        },
        "ident_number": {
          "title": "Identification Number",
          "type": "string"
        },
        "vat_id": {
          "title": "VAT ID",
          "type": "string"
        }
      }
    },
    "contact_info": {
      "title": "Contact Information",
      "classes": "fa fa-address-card",
      "inputs": {
        "contact_persons": {
          "title": "Contact persons",
          "type": "multiselect:contact_person"
        }
      }
    }
  }
}
```

Výpis 22: Případová studie – Definiční soubor entity zákazníků

Jak již bylo vysvětleno v kapitole 9 (Generátor a databáze), framework automaticky rozpozná relace podle typu datového vstupu, jelikož datový vstup „contact_person“ je typu multiselect, generátor ví, že jde o relaci typu N:N a automaticky vytvoří příslušnou pomocnou tabulku.

12.2.4 Nastavení databáze

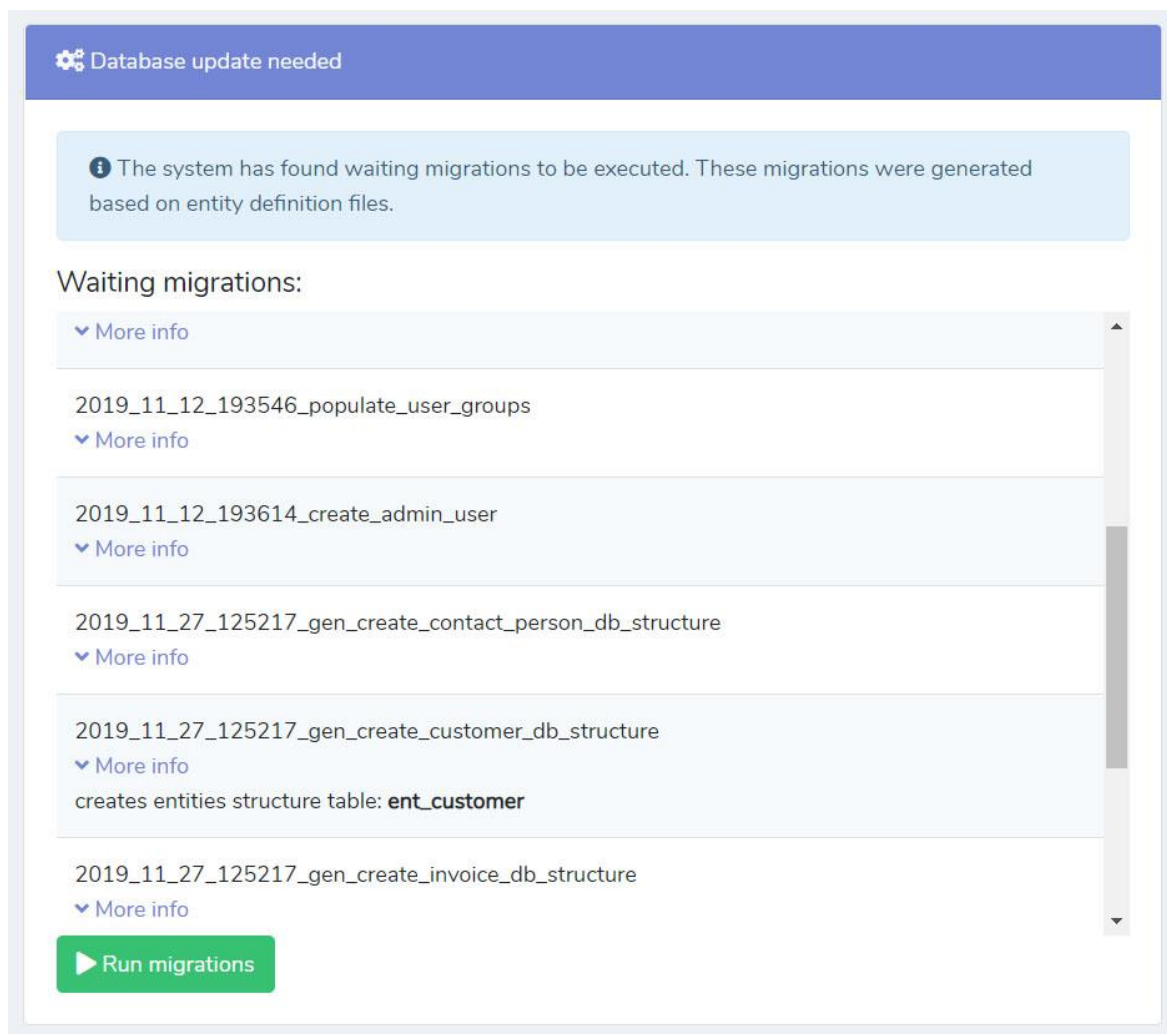
Poslední věcí, kterou musíme udělat, je říci frameworku, kam má ukládat data, proto nastavíme připojení k databázi ve frameworku Laravel. Otevřeme konfigurační soubor „.env“ v kořenovém adresáři aplikace a nastavíme databázi, jako na výpisu z programu č. 23 níže.

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=my_admin
DB_USERNAME=db_user
DB_PASSWORD=user_password
```

Výpis 23: Případová studie – nastavení připojení k databázi

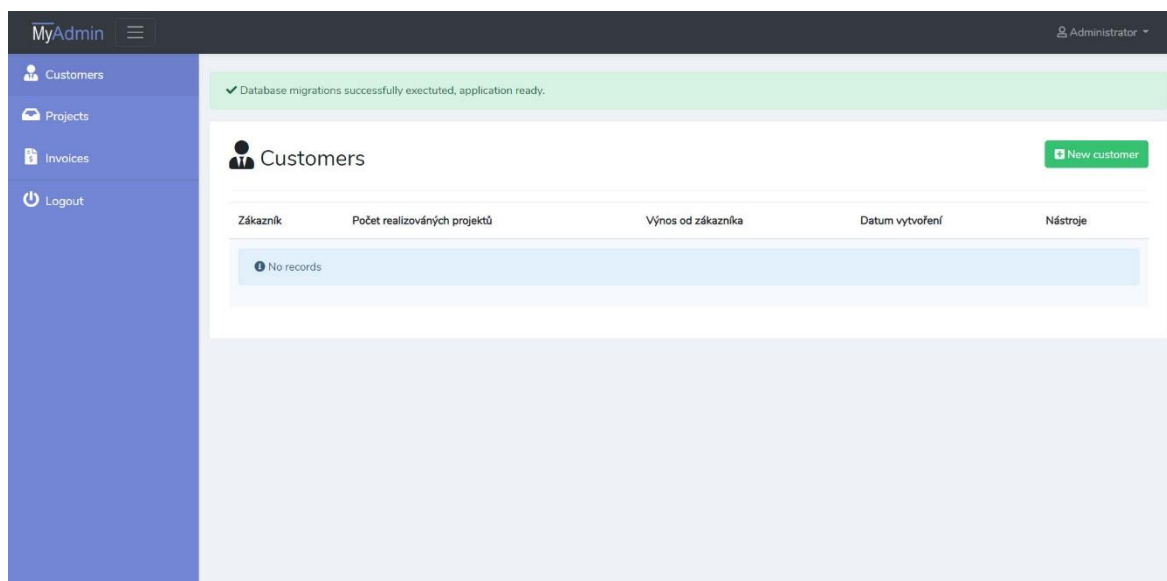
12.2.5 Výsledek

Nyní máme nadefinované vše, co aplikace potřebuje, proto ji navštívíme ve webovém prohlížeči. Na obrázku č. 29 níže je vidět upozornění generátoru, který nám již připravil migrační soubory a vyzívá nás k jejich spuštění. Všimněte si, že kromě entit, které definujeme, automaticky do struktury přibývají tabulky uživatelů a jiná nastavení. I s těmito objekty lze později manipulovat. Jednotlivé migrační soubory lze rozkliknout a podívat se, jak konkrétně strukturu databáze mění.



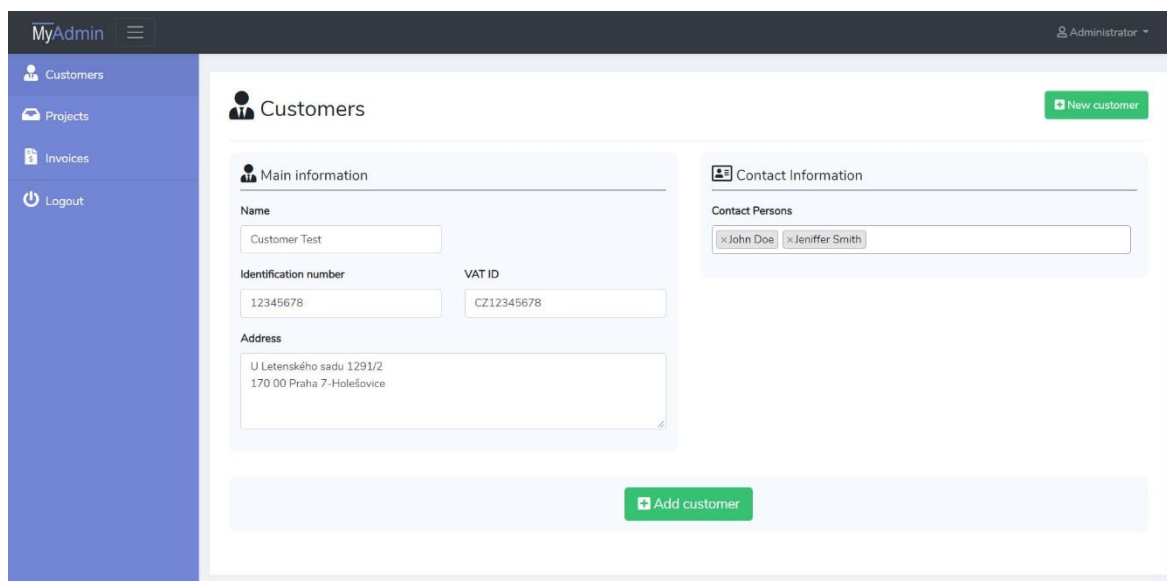
Obrázek 29: Případová studie – výzva generátoru k spuštění migračních souborů

Po úspěšném spuštění migrací je vytvořena kompletní struktura databáze a naše aplikace je připravena k použití, jak je vidět na obrázku č. 30 níže.



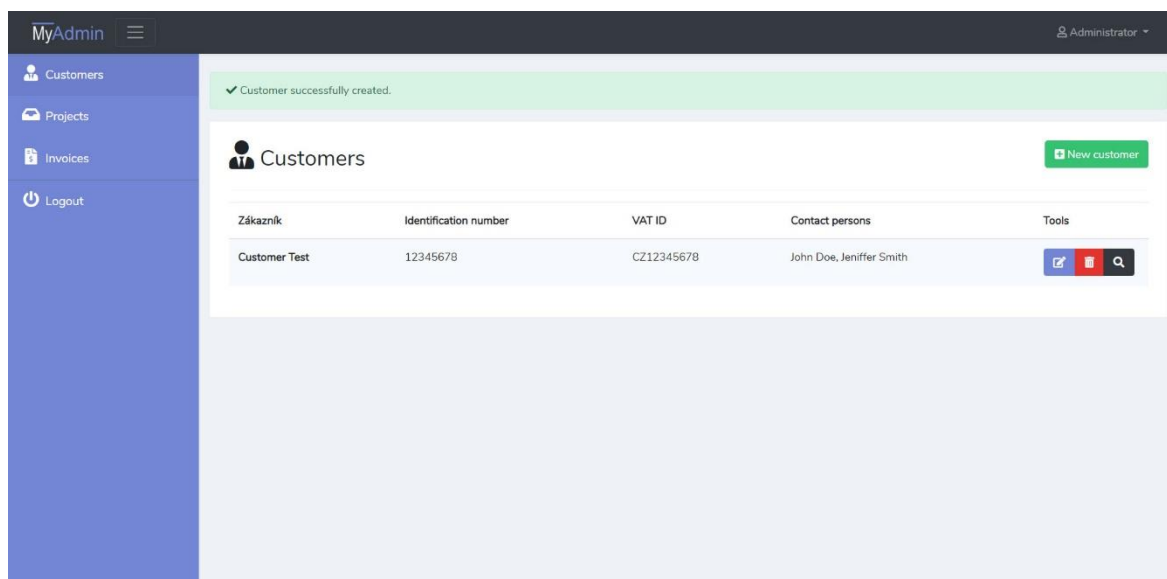
Obrázek 30: Případová studie – aplikace je připravena

Nyní můžeme do aplikace přidat našeho prvního zákazníka. Po kliknutí na „New customer“ vidíme komponentu nového záznamu. Vyplníme údaje, jako na obrázku č. 31 níže a uložíme.



Obrázek 31: Případová studie – přidání nového zákazníka

Po kliknutí na tlačítko „Add customer“ se záznam úspěšně zapsal do databáze a na světě je první instance naší entity zákazníků „Customer test“ v tabulce na obrázku č. 32 níže.



Obrázek 32: Případová studie – zákazník vytvořen

12.3 Závěr

Myslím, že z uvedené případové studie je zřejmé, kolik hodin práce si vývojář ušetřil implementací všech funkcí jako je přihlašování, vytváření databáze, modelů entit, controllerů a pohledů. Všechno je pro něj nyní připraveno a pokud nějaká část nevyhovuje či chybí může ji v souladu s architekturou frameworku doprogramovat jako novou komponentu či funkci.

13 Živá ukázka aplikace

Živá ukázka aplikace s vypracovanou případovou studií je dostupná na následujícím odkazu. Je zde také FTP přístup, pokud by zájemce chtěl zkoušet měnit definiční soubory. Před obhajobou diplomové práce bude vše navraceno do původní podoby.

URL odkaz na živou ukázku frameworku:

<http://preview.any-admin.website/>

Přihlašovací údaje do vytvořené aplikace:

Uživatel: admin

Heslo: admin

FTP přístup (přístup ke zdrojovému kódu a definičním souborům):

Server: ftp.any-admin.website

Uživatel: any_admin@preview.any-admin.website

Heslo: 6JcH)b54IMz*

14 SW nároky a zprovoznění frameworku

Pro úspěšné zprovoznění aplikace je zapotřebí mít na přístroji, na kterém aplikace poběží správně nainstalovaný a nakonfigurovaný server spolu se všemi podpůrnými programy, jak je popsáno níže.

14.1 SW nároky

14.1.1 Operační systém

Framework je na operačním systému závislý do té doby, dokud je na operační systém možné nainstalovat všechny potřebný software uvedený níže. Obojí určitě splňují operační systémy na bázi Linuxu i Windows.

14.1.2 Server

Podmínkou zprovoznění aplikace, je na přístroji nutné mít nainstalovaný:

- **Apache server** (v2.4 a vyšší)
- **MySQL databáze** (v5.5 a vyšší)
- **PHP** (v7 a vyšší)

Vše výše uvedené integruje například distribuční instalace XAMPP, ke stažení zde:

<https://www.apachefriends.org/download.html>

14.1.3 Git

Ke stažení frameworku využijeme softwaru Git, který je ke stažení zde:

<https://git-scm.com/downloads>

14.1.4 Dependency managers

Jelikož framework využívá řady pluginů na back-endové i front-endové části webu, je nutné mít nainstalované tyto dependency managery, pomocí kterých pluginy stáhneme. Pluginy totiž nejsou uloženy na GitLabu spolu s frameworkem, je na ně u uložen pouze odkaz.

- **Composer**

Ke stažení zde: <https://getcomposer.org/>

- **NodeJS**

Ke stažení zde: <https://nodejs.org/en/>

14.2 Postup zprovoznění frameworku (instalace)

K zprovoznění frameworku potřebujeme pouze splnit SW nároky z předchozí kapitoly a přístup k internetu, postupujeme takto:

1. Otevřeme si příkazový řádek a navigujeme do složky, kam si přejme framework umístit.
2. Naklonujeme (stáhneme) si repositář frameworku z projektu z GitLabu pomocí softwaru git:

```
> git clone https://gitlab.com/FeldCode/any-admin.git
```

3. V adresáři se objeví nová složka „any-admin“ se soubory frameworku, přepneme se do ní:

```
> cd any-admin
```

4. Stáhneme si back-endové technologie, na kterých framework závisí pomocí programu composer:

```
> composer install
```

5. Stáhneme si front-endové technologie, na kterých framework závisí pomocí programu node.js (npm):

```
> npm install
```

6. V kořenovém adresáři frameworku přejmenujeme „.env.example“ na „.env“

```
> cp .env.example .env
```

7. Vygenerujeme nový klíč aplikace Laravel

```
> php artisan key:generate
```

8. Otevřeme .env a vyplníme údaje k databázi:

```
DB_CONNECTION=mysql  
DB_HOST=127.0.0.1  
DB_PORT=3306  
DB_DATABASE=any_admin  
DB_USERNAME=db_user  
DB_PASSWORD=user_password
```

9. Nyní vytvoříme definiční soubory a navštívíme aplikaci v prohlížeči, stejným způsobem jako v případové studii v kapitole 12. Generátor nám sám vytvoří potřebnou databázovou strukturu a aplikace je připravena k použití.

15 Závěr

V úvodu práce jsme si vytyčili dvě hlavní zásady:

- Co nejvíce vývojáři **usnadnit práci** a co nejméně **zasahovat do jeho možností**.
- Umožnit vývojáři snadno se **v kódu zorientovat**.

Myslím, že z uvedené případové studie vyplívá, že první zásada byla dodržena a že se podařilo vytvořit framework pomocí kterého běžný webový vývojář pracující s frameworkem Laravel dokáže **v řádu několika minut až hodin vytvořit plně funkční základ administrativní aplikace webového typu**. V druhé fázi, pokud je to nutné si vývojář může upravit či doprogramovat **funkce a komponenty k jakémukoli účelu**, tím je zajištěna stoprocentní kontrola nad funkcionalitou a podobou výsledné aplikace.

Snadnou orientaci v kódu zajistil vývoj frameworku jako nadstavbu frameworku **Laravel**, který je obecně komunitou webových vývojářů nejběžněji používán.

Lze tedy prohlásit, že se podařilo vytvořit framework, ve kterém se běžný webový vývojář velmi snadno zorientuje, a že rovnováha mezi usnadněním práce a limitováním možností vývojáře byla úspěšně nalezena.

Seznam literatury

- [1] Pravidla českého pravopisu s dodatkem Ministerstva školství, mládeže a tělovýchovy České republiky. Academia 1998 ISBN 978-0-692-35334-9.
- [2] JOLLY, R.: Systems thinking for business: capitalize on structures hidden in plain sight. Portland: Systems Solutions Press, 2015 ISBN 978-0-692-35334-9.
- [3] LARAVEL.COM: *Documentation* [Cit. 18. 9.2019]
Dostupné z: <https://laravel.com/docs>
- [4] GETBOOTSTRAP.COM: *Documentation* [Cit. 18. 9.2019]
Dostupné z: <https://getbootstrap.com/docs/4.0/getting-started/introduction/>
- [5] A Project Guide to UX Design : For user experience designers in the field or in the making. New Riders 2012 ISBN 978-01-3293-172-4.
- [6] WIKIKNIHOVNA: UX design [Cit. 06.04.2017]
Dostupné z: http://wiki.knihovna.cz/index.php/UX_design
- [7] MEDIUM.COM: Model-View-Controller – Data Driven Investor [Cit. 21.11.2019]
Dostupné z: <https://medium.com/datadriveninvestor/model-view-controller-mvc-75bcb0103d66>
- [8] RESEARCHGATE.NET: HTTP request/response model [Cit. 21.11.2019]
Dostupné z: https://www.researchgate.net/figure/HTTP-request-response-model_fig3_311571526
- [9] WIKIPEDIA.ORG: Ajax (programming) [Cit. 25.11.2019]
Dostupné z: [https://en.wikipedia.org/wiki/Ajax_\(programming\)](https://en.wikipedia.org/wiki/Ajax_(programming))
- [10] LARAVEL.COM: Middleware [Cit. 25.11.2019]
Dostupné z: <https://laravel.com/docs/master/middleware>
- [11] WIKIPEDIA.ORG: PHP [Cit. 26.11.2019]
Dostupné z: <https://cs.wikipedia.org/wiki/PHP>
- [12] WIKIPEDIA.ORG: MySQL [Cit. 26.11.2019]
Dostupné z: <https://cs.wikipedia.org/wiki/MySQL>
- [13] GITHUB.COM: i-rocky/eloquent-dynamic-relation: Adds dynamic relationship to eloquent ORM models [Cit. 26.11.2019]
Dostupné z: <https://github.com/i-rocky/eloquent-dynamic-relation>
- [14] GITHUB.COM: barryvdh/laravel-dompdf: A DOMPDF Wrapper for Laravel [Cit. 26.11.2019]
Dostupné z: <https://github.com/barryvdh/laravel-dompdf>
- [15] WIKIPEDIA.ORG: HTML5 [Cit. 26.11.2019]
Dostupné z: <https://cs.wikipedia.org/wiki/HTML5>
- [16] WIKIPEDIA.ORG: Kaskádové styly [Cit. 26.11.2019]
Dostupné z: https://cs.wikipedia.org/wiki/Kask%C3%A1dov%C3%A9_styly
- [17] WIKIPEDIA.ORG: JavaScript [Cit. 26.11.2019]
Dostupné z: <https://cs.wikipedia.org/wiki/JavaScript>

- [18] GETBOOTSTRAP.COM: The most popular HTML, CSS, and JS Library in the world [Cit. 26.11.2019]
Dostupné z: <https://getbootstrap.com/docs/4.3/getting-started/introduction/>
- [19] JQUERY.COM: jQuery [Cit. 26.11.2019]
Dostupné z: <https://jquery.com/>
- [20] WIKIPEDIA.ORG: jQuery [Cit. 26.11.2019]
Dostupné z: <https://cs.wikipedia.org/wiki/JQuery>
- [21] WIKIPEDIA.ORG: AJAX [Cit. 26.11.2019]
Dostupné z: <https://cs.wikipedia.org/wiki/AJAX>
- [22] JEREMYFAGIS.GITHUB.IO: Dropify [Cit. 26.11.2019]
Dostupné z: <https://jeremyfagis.github.io/dropify/>
- [23] DROPZONEJS.COM: Dropzone.js [Cit. 26.11.2019]
Dostupné z: <https://www.dropzonejs.com/>
- [24] SELECT2.ORG: Select2 [Cit. 26.11.2019]
Dostupné z: <https://select2.org/>
- [25] SUMMERNOTE.ORG: Summernote [Cit. 26.11.2019]
Dostupné z: <https://summernote.org/>
- [26] SWEETALERT2.GITHUB.IO: SweetAlert2 [Cit. 26.11.2019]
Dostupné z: <https://sweetalert2.github.io/>
- [27] LABS.ABEAUTIFULSITE.NET: jQuery MiniColors [Cit. 26.11.2019]
Dostupné z: <https://labs.abautifulsite.net/jquery-minicolors/>
- [28] EONASDAN.GITHUB.IO: Bootstrap DateTimePicker [Cit. 26.11.2019]
Dostupné z: <https://eonasdan.github.io/bootstrap-datetimepicker/>
- [29] GITBRENT.GITHUB.IO: Bootstrap 4 Toggle Switch Button [Cit. 26.11.2019]
Dostupné z: <https://gitbrent.github.io/bootstrap4-toggle/>
- [30] SASS: Syntactically Awesome Style Sheets [Cit. 27.11.2019]
Dostupné z: <https://sass-lang.com/>
- [31] GIT-SCM.COM: GIT [Cit. 27.11.2019]
Dostupné z: <https://git-scm.com/>
- [32] GETCOMPOSER.ORG: Composer [Cit. 27.11.2019]
Dostupné z: <https://getcomposer.org/>
- [33] NODEJS.ORG: Node.js [Cit. 27.11.2019]
Dostupné z: <https://nodejs.org/en/>

Terminologický slovník

Termín	Zkratka	Význam [zdroj]
<i>Know-how</i>	-	Soubor znalostí a vědomostí či pracovních postupů
<i>Front-End</i>	-	Klientská část vyvíjené webové aplikace. Toto zahrnuje vše, co se provede na počítači u koncového uživatele. (Např.: Vykreslení stránky, Animace, apod.)
<i>Back-End</i>	-	Serverová část vyvíjené webové aplikace. Toto zahrnuje vše, co se provede na serveru. (Např.: Práce s databází, Přihlašovací logika, apod.)
<i>Framework</i>	-	Podpůrný systém/software při vývoji aplikace.
<i>User Experience</i>	UX	User Experience si můžeme představit jako soubor všech lidských zkušeností se systémem. ^[5]
<i>Open-Source</i>	-	Software s „otevřeným kódem“. Jde o volně dostupnou licenci, která držitele opravňuje k jakýmkoli vlastním úpravám a distribuci upravené verze.
<i>Responzivní design</i>	-	Definování CSS stylů takovým způsobem, aby se prvky na stránce poskládaly úhledně na jakémkoli zařízení, podle velikosti jeho obrazovky.
<i>Dependency Manager</i>	-	Software, který hlídá a spravuje vzájemnou provázanost a závislosti využitých technologií.

Seznam obrázků a výpisů

Seznam obrázků

Obrázek 1: Logo AnyAdmin	7
Obrázek 2: Rozložení prvků na stránce aplikace	11
Obrázek 3: Request-Response model ^[8]	12
Obrázek 4: Architektura MVC (Mode-View-Controller) ^[7]	13
Obrázek 5: Konceptuální schéma funkčních prvků frameworku	14
Obrázek 6: Konceptuální rozložení prvků na stránce	33
Obrázek 7: Zobrazená komponenta tabulky na stránce se zákazníky	38
Obrázek 8: Zobrazená komponenta editačního formuláře entity projektů	39
Obrázek 9: Zobrazená komponenta detailu entity projektu	41
Obrázek 10: Spuštění funkce uživatelem – vyžádání datového vstupu od uživatele	46
Obrázek 11: Spuštění funkce uživatelem – chyba	47
Obrázek 12: Spuštění funkce uživatelem – úspěch	47
Obrázek 13: Upozornění generátoru na potřebné migrace – přidání sloupce	51
Obrázek 14: Nový datový vstup je nyní dostupný ve formuláři	51
Obrázek 15: Upozornění generátoru na potřebné migrace – odstranění sloupce	52
Obrázek 16: Základní návrh designu aplikace	54
Obrázek 17: Výchozí zobrazení HTML tagu „input“ typu „file“	61
Obrázek 18: Zobrazení HTML tagu „input“ typu „file“ pomocí pluginu Dropify	61
Obrázek 19: Výchozí podoba formuláře s několika tagy „input“ typu „file“	62
Obrázek 20: Zobrazení formuláře po aplikaci pluginu Dropzone.js	62
Obrázek 21: Výchozí zobrazení HTML tagu „select“ s atributem „multiple“	63
Obrázek 22: Zobrazení HTML tagu „select“ s atributem „multiple“ – Select2	63
Obrázek 23: Textový editor – Plugin Summernote	64
Obrázek 24: Dialogové okno – plugin SweetAlert2	65
Obrázek 25: HTML tag „input“ s možností výběru barvy – plugin jQuery MiniColors	66
Obrázek 26: HTML tag „input“ s možností výběru datumu – Bootstrap DateTimePicker ...	66
Obrázek 27: Porovnání klasického tagu „input“ typu „checkbox“ a pluginu BS4 Toggle	67
Obrázek 28: Schéma zadání případové studie	70
Obrázek 29: Případová studie – výzva generátoru k spuštění migračních souborů	75
Obrázek 30: Případová studie – aplikace je připravena	76
Obrázek 31: Případová studie – přidání nového zákazníka	76
Obrázek 32: Případová studie – zákazník vytvořen	77

Seznam výpisů z programu

Výpis 1: Ukázka dílčích datových objektů definičních souborů: položky menu	16
Výpis 2: Ukázka dílčího datového objektu definičních souborů: uživatelské omezení	17
Výpis 3: Ukázka dílčího datového objektu definičních souborů: tlačítko (<code>_button</code>)	18
Výpis 4: Ukázka dílčího datového objektu definičních souborů: akce (<code>_action</code>)	20
Výpis 5: Ukázka dílčího datového objektu definičních souborů: sekce formuláře	21
Výpis 6: Ukázka dílčího datového objektu definičních souborů: datový vstup (<code>_input</code>)	25
Výpis 7: Příklad použití definičního souboru <code>app.json</code> (výchozí podoba)	26
Výpis 8: Ukázka definičního souboru entity zákazníka	27
Výpis 9: Ukázka definičního souboru stránky (sekce) se zákazníky	29
Výpis 10: Vstupní data komponenty „table“	36
Výpis 11: Vstupní data komponenty „form“	38
Výpis 12: Vstupní data komponenty „detail“	40
Výpis 13: Ukázka vzorové komponenty (PHP třída)	42
Výpis 14: Zápis globální vývojářem definované funkce	43
Výpis 15: Zápis vývojářem definované funkce do modelu entity	44
Výpis 16: Tlačítko s dotazem na spuštění funkce rozeslání hromadného e-mailu	45
Výpis 17: Odpověď funkce rozeslání hromadného e-mailu, předána zpět FEC	45
Výpis 18: Vložení datového vstupu „Poznámka“ do definičního souboru zákazníků	50
Výpis 19: Soubor s překladem do češtiny	55
Výpis 20: Případová studie – Hlavní definiční soubor <code>app.json</code>	70
Výpis 21: Případová studie – Definiční soubor stránky se zákazníky	71
Výpis 22: Případová studie – Definiční soubor entity zákazníků	73
Výpis 23: Případová studie – nastavení připojení k databázi	74