# Informační systém pro správu souhlasů se zpracováním osobních údajů

Adam Ulrich

Bakalářská práce
2019



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

| | |
|---|---|
| Jméno a příjmení: | **Adam Ulrich** |
| Osobní číslo: | **A16750** |
| Studijní program: | **B3902 Inženýrská informatika** |
| Studijní obor: | **Softwarové inženýrství** |
| Forma studia: | **prezenční** |
| Téma práce: | **Informační systém pro správu souhlasů se zpracováním osobních údajů** |
| Téma anglicky: | **An Information System for Managing Consents to the Processing of Personal Data** |

Zásady pro vypracování:

1. **V teoretické části popište vývojový framework Laravel, zaměřte se na nástroje pro práci s databází, ORM Model Eloqent, architekturu MVC a další nástroje pro vývojáře.**
2. **Získejte a přehledně vypište funkční a nefunkční požadavky na informační systém pro správu souhlasů se zpracováním osobních údajů.**
3. **Na základě výše uvedených požadavků navrhněte model databáze a pomocí nástrojů FW Laravel připravte databázové tabulky ve formě migračních souborů.**
4. **Implementujte dále pomocí návrhového vzoru MVC víceuživatelský klient-server informační systém pro správu souhlasů.**
5. **Uveďte také softwarové nároky na nasazení systému na produkční server a věnujte se základním krokům zabezpečení.**

Rozsah bakalářské práce:

Rozsah příloh:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:


1. **STAUFFER, M. (2016). Laravel: Up and Running: A Framework for Building Modern PHP Apps: O'Reilly Media.**
2. **DOCKINS, K. (2016). Design Patterns in PHP and Laravel: Apress.**
3. **Laravel Official Documentation. Laravel.com [online]. N/A: OTWELL, 2018 [cit. 2018-11-26]. Dostupné z: https://laravel.com/docs/5.7**
4. **MySQL Documentation. MySQL Documentation [online]. N/A: Oracle Corporation, 2018 [cit. 2018-11-26]. Dostupné z: https://dev.mysql.com/doc/**
5. **Pusher API. Pusher API Documentation [online]. London: Pusher, 2018 [cit. 2018-11-26]. Dostupné z: https://pusher.com/docs**


Vedoucí bakalářské práce: **Ing. Radek Vala, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce: **3. prosince 2018**

Termín odevzdání bakalářské práce: **15. května 2019**

Ve Zlíně dne 7. prosince 2018


doc. Mgr. Milan Adámek, Ph.D.
*děkan*

prof. Mgr. Roman Jašek, Ph.D.
*garant oboru*

**Prohlašuji, že**

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

**Prohlašuji,**

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 15.5. 2019 Adam Ulrich, v. r.

## ABSTRAKT

Bakalářská práce se zabývá problematikou PHP *frameworku* Laravel. Hlavním cílem práce je vytvořit softwarový systém pro správu souhlasů v rámci GDPR. Cílem teoretické části je poskytnout náhled do funkcí a metod v tomto *frameworku.* Popsat způsob tvorby aplikace, doporučené praktiky, strukturu projektu, způsoby komunikace s databází, možnosti manipulace s daty a zabezpečení aplikace, včetně autentizace. Praktická část práce má za úkol seznámení čtenáře s průběhem vývoje aplikace v *Laravelu*. Sesbírat požadavky, navrhnout databázi, vzhled aplikace a věnuje se i nasazení a zabezpečení hotové aplikace.



Klíčová slova: Laravel, PHP, Systém pro správu GDPR



## ABSTRACT

The bachelor thesis deals with the issue of PHP framework Laravel. The main goal of the thesis is to create a software system for the management of agreements within GDPR. The aim of the theoretical part is a preview of functions and methods of this *framework*. To describe the approaches of creating the application, to describe recommended practices, project structure, ways to communicate with a database, data processing capabilities and application security, including authentication. The practical part of the thesis is aimed at acquainting the reader with the development of the application in *Laravel*. Collect requirements, design the database, design the application appearance, deployment and security of the finished application.



Keywords: Laravel, PHP, GDPR Software System

Zvláštní poděkování patří vedoucímu mé práce Ing. Radkovi Valovi Ph.D. za vedení, cenné rady a příjemnou spolupráci jak při psaní práce, tak při tvorbě softwarového systému. Dále bych chtěl poděkovat také paní Ing. Jitce Jaškové, za specifikaci požadavků na systém a přiblížení tématiky GDPR.

Prohlašuji, že odevzdaná verze bakalářské/diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Největším plodem studia a plným úspěchem práce je schopnost improvizovat.

*Marcus Fabius Quintilianus*

# OBSAH

# ÚVOD

V poslední době dochází k velkému rozmachu aplikačních *frameworků*. *Frameworky* tvoří nástavbu nad samotným programovacím jazykem. Díky jejich zabudované architektuře a nástrojům mohou programátoři a softwarové společnosti produkovat velké softwarové systémy výrazně rychleji, a přitom stále zachovat strukturu a přehlednost kódu. Jedním s takových *frameworků* je i Laravel, který je postavený na jazyce PHP. Práce si dala za úkol využít tohoto *frameworku* k vytvoření systému na správu souhlasů GDPR a demonstrovat na něm problematiku *Laravelu*, jeho metody a možnosti při tvorbě takového projektu. V současnosti je *Laravel* jedním z nejvíce používaných PHP *frameworků*. V teoretické části se práce zpočátku zaměří na definici samotného *Laravelu*, jeho historií a instalací – kde demonstruje, jak jednoduché je s *frameworkem* začít. Následuje popis architektury, kterou *Laravel* využívá, seznámení se základními pojmy architektury a jejich reálnou implementací. Popíše, jak komunikuje s databází, jaké může programátor použít metody při manipulaci s daty v databázi a v neposlední řadě seznámí čtenáře s dalšími nástroji, které může při vývoji použít a mohou mu přijít vhod. Informace v teoretické části se opírají o dokumentaci *frameworku*, publikace, které se *Laravelem* zabývají nebo – a to zejména proto, že *Laravel* je moderní *framework* a spoustu poznatků si uživatelé sdílejí skrz internet – různé blogy technologů, kteří *Laravel* používají pro svůj vývoj. V praktické části má práce za úkol implementovat samotný systém pro správu souhlasů. Nejprve sesbírá požadavky na systém, navrhne databázový model a v hlavní části popíše vzniklý systém, který byl naprogramován v rámci této bakalářské práce, jeho manipulaci jak uživatelem, tak administrátorem. U vybraných řešení popíše i důvod zvoleného způsobu implementace. Nakonec popíše nasazení na produkční server a požadavky na jeho nároky.

# I. TEORETICKÁ ČÁST

# 1 SEZNÁMENÍ S FRAMEWORKEM LARAVEL

V této kapitole bakalářská práce provede prvním seznámením s *Laravelem*, jeho historií, architekturou a též instalací. Popíše adresářovou strukturu a důležité pojmy, které se budou v průběhu práce vyskytovat a je proto nezbytné, aby čtenář věděl, co znamenají.

## 1.1 Co je Laravel?

Laravel je open-source *framework* (vývojová platforma) pro webové aplikace založená na jazyce PHP, licencovaná pod MIT. Jeho vznik podnítil Taylor Otwell, pod záminkou vylepšit již existující PHP *frameworky*, zejména pak *CodeIgniter* (https://www.codeigniter.com/). V roce 2011 *CodeIgniter* neobsahoval funkce, které PHP programátor denně používal, jako byly například autentizace uživatele nebo "*closure routing*", což je jednoduchý způsob, jak specifikovat "routy", čili cesty, pomocí nichž se orientujeme v aplikaci. Používá se pro tvorbu webových aplikací, jak na serverové straně, tak na straně klienta. [1] [2]

## 1.2 Co je PHP?

PHP je open-source skriptovací jazyk vytvořený pro webový vývoj a úzce spolupracující s HTML. Na rozdíl od jiných jazyků (*JavaScript*, C#…) je jeho kód vykonáván na serveru, kde generuje HTML, které je poté posláno klientovi. To znamená, že se PHP kód používá tam, kde pracujeme s daty v databázi. PHP je počítačový jazyk, což znamená, že v základu neobsahuje žádnou architekturu, ani pokročilé nástroje, které pomáhají programátorovi s každodenním vývojem a které budou popsány dále. [3]

## 1.3 Historie Laravelu

První *betaverze Laravelu* se objevila 9. června 2011. Obsahovala již zmíněnou autentizaci uživatele, Eloquent ORM (viz kapitolu Eloquent) pro práci s databází, modely, vazby mezi databázovými tabulkami, *routing*, *kešování*, *sessions*, *views* a mnoho knihoven, které usnadňovaly denní práci. Přestože Laravel v této verzi nebyl postavený na architektuře MVC, jak je tomu dnes, rostoucí popularita dala do šesti měsíců vznik MVC verze *Laravelu*, Laravel 2. [1]

Laravel 2 již obsahoval všechny náležitosti, které zajistily, že mohl být považován za plnohodnotný MVC *framework*. *Controllers*, modely, *views* úzce spolupracující s nástroji zabudovanými přímo ve *frameworku*. [1]

Laravel 3, verze, která vyšla 22.2. 2012 se kterou přišla mimo jiné nová webová stránka projektu, obsahovala integraci takzvaných "*bundles*", později nazývaných "*packages*" (balíčky). Vznikaly užitečné návody, jako například známý výukový nástroj Laravel blog, který naučí uživatele vytvořit jednoduchý blog pro vkládání příspěvků a jejich komentáře. [1]

Laravel 4 představil tzv. DB *Seeding*, což je nástroj, který nám umožní naplnit databázi skriptem, ihned po vytvoření tabulky, podporu mailů, fronty a mnohé další. [1]

Nyní, v únoru roku 2019 je aktuální verze 5, o které je pojednává tato práce. Tato verze přinesla novinky jako "*Middlewares*", pomocí nichž specifikujeme, které cesty jsou komu přístupné, "*Event Handlery*", "*File* Systems" a další pokročilé funkce, přesahující úroveň této práce. [4]

## 1.4 Instalace Laravelu

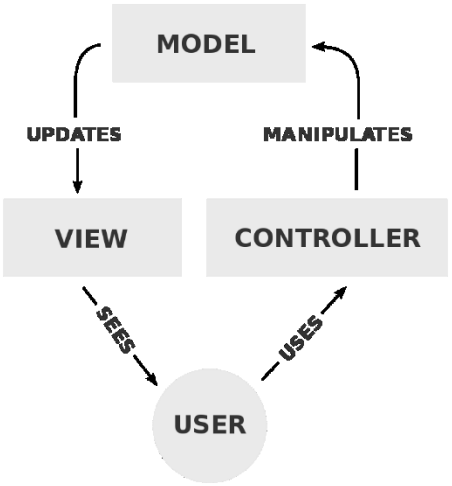
Instalace *Laravelu* probíhá v pouhých dvou krocích: Instalace *Composeru* a instalace *Laravelu*. Laravel využívá správce balíčků *Composer*. *Composer* je skript, který díky speciálnímu souboru (*composer.json*) je schopný nainstalovat veškeré závislosti, které jsou specifikované v tomto souboru, ale i závislosti těch závislostí. Tím je docílena maximální přenositelnost projektů. V praxi to znamená, že kdykoli chceme pracovat s jakkoli velkým projektem v *Laravelu*, který obsahuje množství balíčků, na kterých je závislý, dokud jsou specifikovány v *Composer* souboru, stačí spustit skript a *Composer* se postará, aby v projektu na konci skriptu byly všechny potřebné balíčky. Jelikož i Laravel jakožto *framework* je tvořen z mnoha balíčků, je *Composer* odpovědný za samotnou instalaci *Laravelu*. Svou funkcí se podobá jiným správcům balíčku jako je například *npm* (https://www.npmjs.com/). Na ukázce kódu 1 vidíme demonstraci instalace *Laravelu* pomocí *Composeru*. [5] [16]

```
composer global require "laravel/installer"
```

*Kód 1. Instalace Laravelu [5]*

## 1.5 Architektura Model-View-Controller

*Model-View-Controller* je architektonický vzor používaný v mnoha webových aplikacích a podporovaný nejpoužívanějšími *frameworky* dnešní doby (*Laravel*, *Angular*...). Rozděluje aplikaci na 3 hlavní logické komponenty (reprezentace formy informací) - Model, *View* a *Controller*, podle toho, jak jsou dané informace prezentovány pro uživatele nebo od uživatele akceptovány. V roce 1979 tuto architekturu formuloval Norský vědec Trygve Reenskaug. [18]




Obr. 1. MVC Architektura. [17]

Popis jednotlivých komponent dle zprávy Trygveho Reenskauga z 10.12. 1979

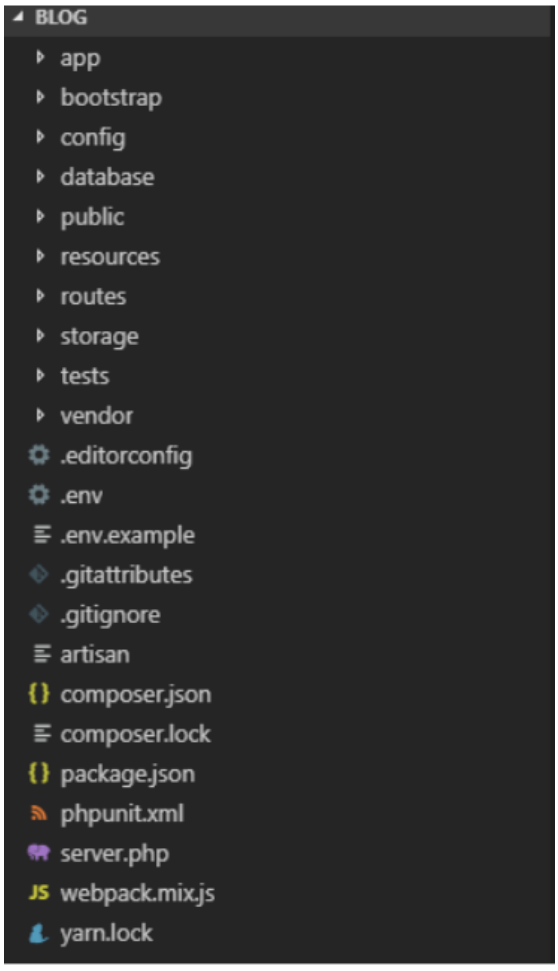
*Model* – reprezentuje vědomosti. Může to být jeden objekt nebo struktura objektů. Mezi modelem a jeho částmi by měla existovat vzájemná korespondence na straně jedné a zastoupenému světu vnímavému majitelem modelu na straně druhé. [18]

*View* (pohled) - je vizuální reprezentace jeho modelu. Je přiložen k modelu (nebo modelové části) a získává data potřebná pro prezentaci z modelu pokládáním otázek. Může také aktualizovat model zasláním vhodných zpráv. [18]

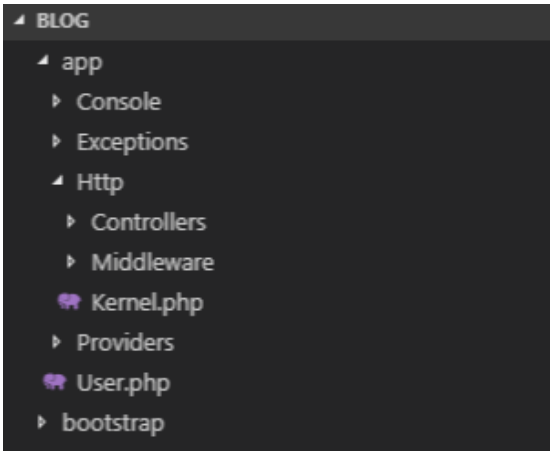
*Controller* (řídící jednotka) - je spojení mezi uživatelem a systémem. Poskytuje uživateli vstup pomocí uspořádání relevantních pohledů, aby se prezentovali na vhodných místech na obrazovce. [18]

## 1.6 Adresářová struktura

Adresářová struktura *Laravelu* logicky souvisí s popsaným architektonickým modelem a je doplněna o adresáře pro nejčastěji používané scénáře a adresáře obsahující soubory, které buď využívá Laravel nebo jsou nástrojem pro jeho konfiguraci. Následující obrázky jsou vizualizací adresářové struktury projektu v *Laravelu* verze 5.6, ve stavu po vygenerování nové aplikace příkazem. Rozumět adresářové struktuře je esenciální pro správný vývoj ve *frameworku Laravel*, jelikož jeho architektura je přímo postavená tak, abychom strukturovali naše soubory do předem připravených souborů.




Obr. 2. Adresářová struktura nového projektu “Blog”.




Obr. 3. Adresářová struktura adresáře app.

Adresář *app* obsahuje takřka všechny třídy, které programátor vytvoří. Bude popsán detailně níže.

Adresář *bootstrap* obsahuje soubor *app.php*, který se stará o načtení *frameworku* Laravel a adresář *cache*, který je uchovává optimalizační soubory vygenerované *frameworkem* (*route* a *cache* soubory).

Adresář *config* obsahuje konfigurační soubory, které uživatel může upravovat. Může zde upravit například výchozí driver pro šifrování, manuálně specifikovat připojení k databázi, cesty k lokálnímu úložišti (více níže v kapitole o souborech) a mnohé další.

Adresář database je významný tím, že obsahuje funkce, díky kterým framework usnadní práci s databází. Jedná se o migrační soubory, *seedovací* soubory a *factories*. Všechny tyto pojmy budou popsány níže v příslušných kapitolách.

Adresář public obsahuje *index.php*, což je vstupní soubor, pro všechny požadavky, nejen v *Laravelu* ale v celém PHP jazyce. Zastřešuje též soubory potřebné pro *frontendovou* část *Laravelu* (.css soubory pro stylování aplikace, *JavaScript* soubory pro funkci *frontendových* skriptů). V neposlední řadě se zde nachází soubor *.htaccess.* Použijeme ho při nasazování naší aplikace na server a využívá ho například *Apache* HTTP Server (https://httpd.apache.org/).

Adresář *resources* obsahuje *views*, na tento adresář se budeme odkazovat, když budeme chtít najít konkrétní *view*, do kterého posíláme data.

Adresář *routes* obsahuje 4 soubory, které se starají o definici cest a hlavně akcí, které se mají provést po přístupu na danou cestu.

- Soubor *web.php*, který obsahuje cesty, které přímo používáme ve *views* a podporuje proto uchování stavu session CSRF ochranu (viz kapitolu *frontend*)
- Soubor *api.php* obsahuje cesty, které budou autorizovány pomocí tokenů a budou tedy základním kamenem tzv. *RESTful* API.
- Soubor *console.php*, sloužící pro příkazy z konzole.
- Soubor *channels.php*, kde registrujeme kanály, které potřebují zachycovat *eventy* (například *broadcast* kanálu pro *realtime* chatování)

Adresář *storage* obsahuje zkompilované *Blade* soubory (viz níže) a soubory vygenerované aplikací. Jsou zde uloženy též soubory nahrané uživatelem (obrázky, soubory, které nahrál, profilové fotky...), pokud vytvoříme symbolický link (kód 2) a tuto možnost umožníme.

```
php artisan storage: link
```

*Kód 2. Vytvoření symbolického linku.*

Adresář *tests* obsahuje automatizované testy. Budou popsány v kapitole s testováním.

Většina aplikace je uložená v adresáři *app*.

Kořen adresáře *app* obsahuje soubory, které často vytvoří přímo uživatel na základě daného scénáře, a to ručně nebo pomocí příkazů make, které umožňuje *Artisan* (viz níže). Přímo v adresáři *app* se nacházejí třídy reprezentující modely.

Podadresář *broadcasting* obsahuje třídy přenosových kanálů aplikace. V základu není dostupná, ale lze vygenerovat příkazem *Artisanu* (kód 3).

```
php artisan make:channel
```

*Kód 3. Vytvoření adresáře broadcasting pomocí nástroje Artisan.*

Podadresář *console* obsahuje soubor *kernel.php*. V tomto souboru definujeme vlastní příkazy pro *Artisan* make, také se zde definují plánované úlohy (úlohy, které má kernel vykonat v daném čase).

Podadresář *events* stejně jako *broadcasting* není v základu přítomný, lze vygenerovat pomocí nástroje *Artisan*. V tomto adresáři se nacházejí třídy událostí. Události mohou informovat části aplikace o tom, že se udála nějaká určitá událost.

Podadresář *exceptions* obsahuje *handler* (manipulátor) pro výjimky a je též místem, kde uchováváme výjimky, které produkuje (vyvolává) naše aplikace.

Podadresář http obsahuje *controllers* (řídící jednotky). Uchováme zde všechny třídy, které budou spravovat vnější HTTP požadavky.

Podadresář *providers* obsahuje poskytovatele služeb (*service providers*). V základu zde najdeme již například poskytovatele pro autentizaci (*AuthServiceProvider*) nebo poskytovatele pro správu aplikačních cest (*routes – RouteServiceProvider*).

Následující podadresáře jsou volitelné, nejsou obsaženy v adresářové struktuře po vygenerování aplikace, ale je možné je vygenerovat pomocí nástroje *Artisan*. Jedná se o adresáře, do kterých ukládáme soubory, které využijeme při pokročilých scénářích.

Podadresář *jobs* by obsahoval úlohy, které mají schopnost řadit se do fronty.

Podadresář *listeners* by obsahoval třídy, které naslouchají událostem. Například *SendWelcomeEmail* by mohl být *listener* (naslouchač k události), mohl by naslouchat k události *UserRegistered* a tak by po každé registraci uživatele poslal uvítací email.

Podadresář mail by obsahoval třídy, které reprezentují emaily ve formě objektů.

Podadresář *notifications* by obsahoval třídy odpovědné za odesílání notifikací přes různé ovladače (email, SMS …)

Podadresář *policies* by obsahoval různé autorizační brány, které používáme například pro zjištění, zda daný uživatel má právo na danou akci.

Podadresář *rules* by obsahoval volitelná validační pravidla aplikace. Jedná se o velmi pokročilý scénář. [19] [6]

# 2 DATABÁZE A ELOQUENT

*Laravel* je *framework*, který je vytvořen pro používání na straně serveru. Jeho metody jsou tudíž úzce spjaty s databází. Je proto nezbytná znalost databází alespoň na úrovni návrhu.

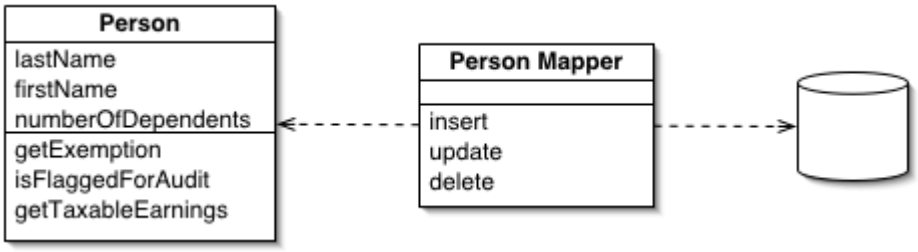
## 2.1 Eloquent

Laravel poskytuje nástroj pro práci s databází, který slouží jako vrstva nad skutečnou databází, díky které jsme odstíněni od databázových příkazů, jazyka SQL a dalších technologií, které dříve musel programátor ovládat. Eloquent je vlastnost Laravelu, díky které si získal velkou popularitu. Je to ukázka toho, jak se Laravel liší od ostatních konkurenčních *frameworků*.

### 2.1.1 Úvod k objektům

Eloquent v *Laravelu* je založen na principu ORM (*object-relational-mapper*).

ORM je vrstva softwaru, která odděluje objekty v paměti od databáze. Jejím úkolem je přenášet data mezi oběma a také je od sebe izolovat. S ORM objekty v paměti nemusí vědět, že je přítomna databáze. Nepotřebují žádný kód rozhraní SQL a s tím související znalosti schématu databáze. (Schéma databáze je vždy odstíněno od objektů, které ji používají.) Protože je to forma *Mapperu*, samotný ORM dokonce nezná ani doménová vrstva. [21]




Obr. 4. Schéma mapování objektu Person do databáze. [22]

Objekt Person je v *Eloquentu* nazýván model. Model je entita, která má své vlastní atributy, které ho popisují. Každá tabulka v databázi má korespondující model, který je použit k interakci s touto tabulkou. Model se používá pro vytažení dat z tabulek nebo vkládání nových řádků do tabulky. K tomu, aby Laravel správně namapoval model na tabulku, musí být specifikován typ a vlastnosti databáze, se kterou má model propojit. Toho je docíleno v již zmíněném adresáři *config*, v souboru *database.php*. Zde jako první můžeme specifiko-

vat výchozí připojení do databáze (např. Připojení k databázi *MySQL* - https://www.mysql.com/).

```
'default' => env('DB_CONNECTION', 'mysql');
```

*Kód 4. Konfigurace výchozího databázového připojení.*

Dále jsou zde specifikovány údaje pro připojení k dané databázi, kde většina z nich je vyčtena přímo ze souboru "*env*". Soubor "*env*" (zkratka pro *environment = prostředí)* je soubor, ze kterého Laravel čte informace o konfiguraci prostředí. Specifikuje ho uživatel a může se lišit pro každý projekt. Obsahuje položky jako název aplikace, připojení k databázi, ale také heslo k databázi, SMTP serveru a další informace týkající se daného prostředí použitého v aplikaci.

### 2.1.2 Propojení databázového objektu a objektu v Laravelu

V *Laravelu* je objekt reprezentován jako třída, která rozšiřuje základní třídu Model *(Illuminate\Database\Eloquent\Model)*. ORM se pak postará o to, že nově vytvořený model „*namapuje*" na tabulku v databázi včetně příslušných atributů. Konvence radí psát model v jednotném čísle. Eloquent poté sám přidá množné číslo do názvu tabulky při mapování (například model Car bude „namapován" do tabulky *cars* v databázi, ale dokonce model *Mouse* bude „namapován" na tabulku *mice*). Pokud chceme tuhle konvenci přepsat, stačí k modelu připsat atribut table a přiřadit mu řetězec s hodnotou názvu tabulky, jaký si přejeme. [20]

```
protected $table = 'custom_table_name';
```

*Kód 5. Ukázka změny názvu tabulky.*

Další užitečný atribut je *fillable*, kde specifikujeme pole atributů, ke kterým bude moci aplikace přistupovat pod záminkou změny. To znamená, že pokud chceme, aby uživatel byl schopný přiřadit našemu objektu jméno, je potřeba do pole *fillable* jméno připsat. [20]

```
protected $fillable = ['name'];
```

*Kód 6. Ukázka atributu fillable.*

V neposlední řadě atribut *hidden*, kde specifikujeme atributy, které nechceme, aby byly přístupné při výpisu atributů modelu. Typicky zde patří hesla či tokeny. [20]

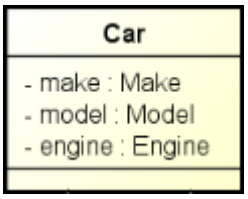
```
protected $hidden = ['password'];
```

*Kód 7. Ukázka atributu hidden.*

Každá třída, kterou vytvoříme má vlastní *namespace*. To je cesta k souboru, která slouží pro jednoznačnou identifikaci dané třídy. Pokud vytvoříme model dle konvence *Laravelu*, v kořeni složky *App*, bude výchozí *namespace App*. [20]

### 2.1.3 Práce s objektem v Laravelu

Eloquent je díky „*namapování*“ na tabulky schopen poskytnout nástroj pro generování SQL příkazů z jeho vazeb a metod. Což významně usnadňuje práci s tabulkami. Mějme následující model Car (auto), reprezentující tabulku *cars*. Model specifikujeme ručně jako třídu, je zde také možnost vygenerovat ji nástrojem *Artisan*.




Obr. 5. Model Car.

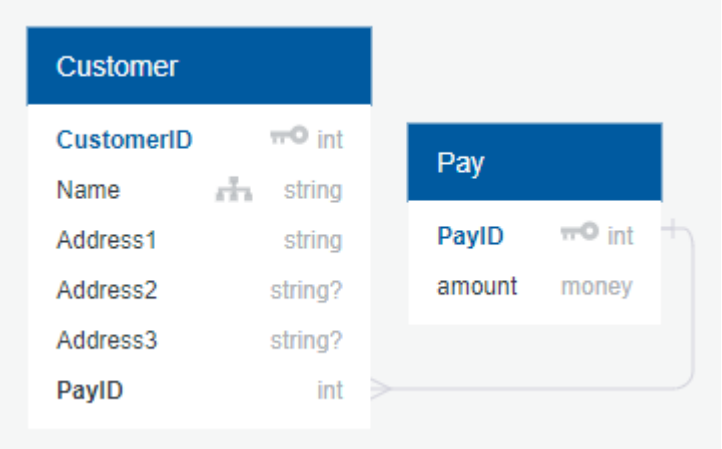
Model Car, pokud je jeho třída vytvořena v adresáři *app*, má výchozí *namespace App\Car*. Tento *namespace* používáme přímo ve volání třídy, můžeme též specifikovat alias klíčovým slovem “as” (*use App\Car* as Auto).

```php
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Car extends Model
{

}
```

*Kód 8. Ukázka reprezentace modelu Car.*

### 2.1.4 Relační vazby

Jednou z klíčových vlastností *Eloquentu* v *Laravelu* je správa tabulkových vazeb. SQL definuje v základu 3 typy vazeb. Jsou to vazby 1:1 (*One-to-one*), 1:M (*One-to-many*) a M:N (Many-to-many). Vazba 1:1 není úplně specifická, jelikož většinou není důvod atributy tabulky jedné vytěsnit do tabulky druhé, nicméně někdy, z bezpečnostního hlediska nebo hlediska větší přehlednosti, je využití této možnosti nezbytné. Schémata byla vytvořena pomocí nástroje *Quick Database Diagrams (app.quickdatabasediagrams.com).*



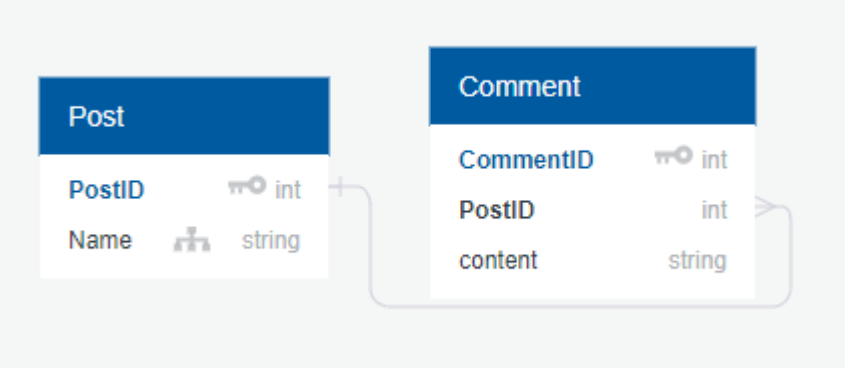

Obr. 6. Databázové schéma relace 1:1.

V *Laravelu* specifikujeme tuto vazbu následujícím způsobem. Uvažujme objekty *Customer* a *Pay*.

```php
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Customer extends Model
{
    public function pay()
    {
        return $this->hasOne('App\Pay');
    }
}
```

*Kód 9. Model Customer.*

Vazba 1:M (nebo i naopak M:1) je nejběžnější typ vazby. Říká, že řádek v tabulce první, může mít mnoho instancí tabulky druhé. Příkladem může být příspěvek, který má přiřazeno mnoho komentářů. Může tedy existovat mnoho komentářů, které patří jednomu jedinému příspěvku, jako v reálném blogu.




Obr. 7. Databázové schéma relace 1:M.

V *Laravelu* můžeme tuto vazbu specifikovat následovně.

```php
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Post extends Model
{
    public function comments()
    {
        return $this->hasMany('App\Comment');
    }
}
```

*Kód 10. Model Post.*

V aplikaci pak můžeme jednoduše získat komentáře, které patří k danému příspěvku. Následující příkaz najde příspěvek s ID 1 (viz níže metody pro práci s tabulkami) a k němu příslušné komentáře.

```php
$comments = App\Post::find(1)->comments;
```

*Kód 11. Metoda find.*

Tento příkaz může být pomocí jazyka SQL nahrazen následovně.

```sql
SELECT *
FROM Comments
INNER JOIN Posts
ON Comments.PostID=Posts.ID WHERE Posts.ID=1
```

*Kód 12. SQL přepis metody find. [28]*

*Laravel* na pozadí takový kód opravdu vygeneruje. Pokud bychom chtěli vazbu z druhé strany, stačí v modelu Comment specifikovat, že patří entitě *Post*.

```php
<?php
namespace App;

use Illuminate\Database\Eloquent\Model;

class Comment extends Model
{
    public function post()
    {
        return $this->belongsTo('App\Post');
    }
}
```

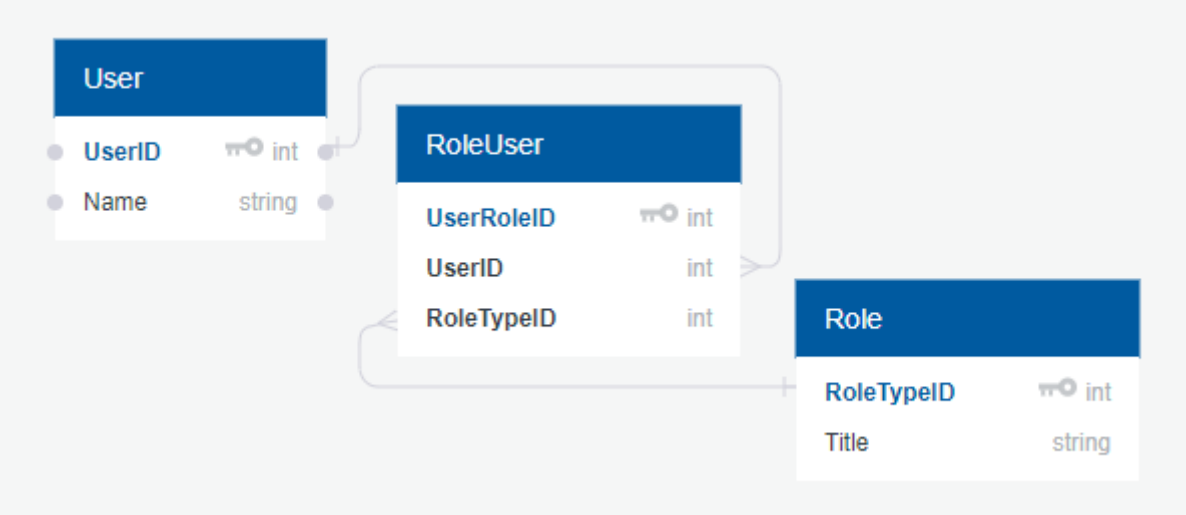
*Kód 13. Relace belongsTo.*

Pomocí této vazby můžeme nalézt příspěvek, ke kterému patří daný komentář, jak je demonstrováno na následující ukázce, která vypíše nadpis příspěvku, u kterého byl daný komentář vložen.

```php
$comment = App\Comment::find(1);

echo $comment->post->title;
```

*Kód 14. Použití relace.*

Vazba M:N (many-to-many) je typická pro použití v případě aplikačních rolí. U těch totiž vyžadujeme, aby více uživatelů mohlo být součástí více rolí. Lze však nahradit dvěma vazbami one-to-many. Této vazby se dá docílit za pomocí tzv. propojovací tabulky („*junction table*“). [23]




Obr. 8. Databázové schéma relace M:N.

V *Laravelu* nemusíme propojovací tabulku vytvářet sami (ačkoli můžeme), ale při použití následujícího kódu, *Laravel* vygeneruje potřebné vazby s propojovací tabulkou sám. Tuto propojovací tabulku pak nazve tak, že zkombinuje názvy dvou tabulek vlevo a vpravo a tyto názvy seřadí podle abecedy. (viz obrázek 8).

```
<?php
namespace App;
use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    /**
     * The roles that belong to the user.
     */
    public function roles()
    {
        return $this->belongsToMany('App\Role');
    }
}
```

*Kód 15. Relace belongsToMany.*

Toto automatické vytvoření však není vždy žádoucí. Často se stává, že chceme věci udělat jinak. Následující kód je ukázka toho, jak jednoduše přepsat pravidlo, které má *Laravel*

zabudované a vytvořit si relaci, používající uživatelem již vytvořenou propojovací tabulku s názvem „*UserRoles*“.

```php
<?php

namespace App;
use Illuminate\Database\Eloquent\Model;

class Role extends Model
{
    /**
     * The users that belong to the role.
     */
    public function users()
    {
        return $this->belongsToMany('App\User')->using('App\UserRoles');
    }
}
```

*Kód 16. Relace belongsToMany s vlastní tabulkou.*

### 2.1.5 Metody pro práci s databázovými tabulkami

Eloquent nabízí metody, ze kterých generuje SQL příkazy pro danou databázi. V následujících ukázkách uvažujme databázi *MySQL*, která je použita i v praktické části projektu.

Metoda *all*, vrací všechny položky z databázové tabulky, ke které je daný model „namapován“, ve formě pole objektů. [19]

```php
$cars = App\Car::all();
```

*Kód 17. Metoda all.*

Ve skutečnosti tato metoda najde „namapovanou“ tabulku Car a provede na ní SQL příkaz příslušný danému *enginu*. V našem případě by to byl příkaz v následující ukázce kódu.

```sql
SELECT * FROM cars;
```

*Kód 18. Metoda all – SQL přepis.*

Metoda *where* vrací pouze položky, které splňují podmínku specifikovanou v její parametrech. Z pravidla očekává dva až tři parametry. V prvním případě, pokud použijeme verzi s dvěma parametry, očekává jako první parametr název sloupce dané tabulky a jako druhý parametr hodnotu, které se má rovnat. [19]

```
$users = DB::table('users')->where('age', 100)->get();
```

*Kód 19. Metoda where – krátká verze.*

Tato verze je přitom jen zkratka pro verzi se třemi argumenty. Ta očekává na prvním místě opět název sloupce, na druhém operátor a na třetím hodnotu. Předchozí případ by šel tedy rozepsat do následujícího kódu. [19]

```
$users = DB::table('users')->where('age','=', 100)->get();
```

*Kód 20. Metoda where – dlouhá verze.*

Obě metody uloží do proměnné *users* pole uživatelů, kteří mají sto let. Můžeme si povšimnout, že na konci příkazu je použita metoda *get*. To proto, že pouze některé metody (jako například metoda *all*) umí automaticky vrátit výsledek, v opačném případě očekává další zřetězení. Pro názornost SQL příkaz by vypadal následovně. [19]

```
SELECT * FROM Users WHERE age=100;
```

*Kód 21. SQL přepis metody where. [28]*

Další zkratkou je hojně používaná metoda *find*. Tato metoda hledá prvek dle primárního klíče. [19]

```
$car = App\Car::find(1);
```

*Kód 22. Metoda find.*

Zajímavá je metoda *pluck*. Mějme následující scénář. Tabulka rolí je složená z ID role, názvu role, hodnoty role a času vytvoření role. Často chceme na určitém místě v aplikaci vypsat dostupné názvy rolí. [19]

```
$titles = DB::table('roles')->pluck('title');
```

*Kód 23. Metoda pluck.*

Proměnná *titles* bude po vykonání příkazu obsahovat pouze názvy (*titles*). Tj. například (*Admin*, *User*, *Manager*). Je to ekvivalence k následujícímu SQL příkazu. [19]

```
SELECT title FROM Roles;
```

*Kód 24. SQL přepis metody pluck. [28]*

### 2.1.6 Migrace

Další z vlastností *Laravelu*, na kterou uživatel narazí hned zpočátku jsou migrace. Ty jsou součástí *eloquentu*. Je to tudíž další součást odstínění uživatele od psaní SQL kódu. Migrace jsou *PHP* kód, který přímo vytváří tabulky, datové typy a relace. V *Laravelu* jsou dostupné pod tzv. migračními soubory. Migrační soubor lze vytvořit dvěma způsoby. První je, vytvořit migraci při vytváření modelu, kde Laravel sám zvolí název tabulky a sloupce. Druhý, více kontrolovaný je vytvořit vlastní migraci ručně. Toho lze docílit opět pomocí nástroje *Artisan*, v příkazové řádce projektu.

```
php artisan make:migration create_users_table
```

*Kód 25. Tvorba migrace s vlastním názvem tabulky.*

Výše uvedený příkaz vytvoří soubor, v jehož názvu je obsažen čas vytvoření souboru a název tabulky která se tvoří. Čas vytvoření je obzvlášť důležitý, jelikož Laravel při migrování tabulek postupuje od nejstarší po nejnovější a je tudíž velmi důležité, aby se tyto soubory daly časově odlišit. V *SQL* je nezbytné, aby tabulka s cizím klíčem vznikla až po vytvoření tabulky s klíčem primárním, na který cizí klíč ukazuje, pokud chceme, aby byla zachována integrita *SQL*. Migrační soubor disponuje dvěma metodami. Metodu *up* a metodou *down*. Metoda *up* obsahuje kód, který se provede při modifikování schématu (nebo počátečním vytvoření neexistujícího databázového schéma). Na druhé straně metoda *down* obsahuje kód, který se provede tehdy, kdy je potřeba vrátit změny zpět. Typicky při úplném obnovení databáze (tzv. *fresh* migraci). V metodě *up* můžeme tedy typicky psát kód pro vytvoření databázového schématu včetně všech jeho sloupců s datovými typy. Na následující ukázce lze vidět migrační soubor s oběma metodami, kde v metodě *up* proběhne vytvoření tabulky *„flights“*.

```php
<?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateFlightsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('flights', function (Blueprint $table) {
            $table->increments('id');
            $table->string('name');
            $table->string('airline');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::drop('flights');
    }
}
```

*Kód 26. Migrační soubor.*

Metoda up má za úkol vytvořit schéma pro daný databázový *engine* (specifikovaný v konfiguračním souboru) a s ním dané sloupce. Laravel poskytuje pro každý databázový datový typ vlastní přepis. Těchto přepisů je 56. V ukázce výše vidíme datový typ increments, který Laravel převede do datového typu *UNSIGNED INTEGER* s automatickým inkrementováním, sloužícím pro primární klíč. Typ *string* převede Laravel na *SQL* datový typ VARCHAR. V neposlední řadě Laravel radí vytvořit i sloupec s funkcí *timestamps*. Tato funkce vytvoří nulovatelný sloupec *created_at* a sloupec *updated_at* s datovými typy

*TIMESTAMP*. V neposlední řadě můžeme do funkce up přidat vlastní kód, který chceme, aby se provedl při vytvoření databázové tabulky. V praktické části této práce přidávám při migraci tabulky uživatelů ukázkového uživatele pro potřeby testování. V metodě *down* pak je veškerý kód, který se provede při změně nebo smazání tabulek. Metoda *drop* provede *SQL* příkaz *DROP TABLE*. [7]

Tento soubor pak slouží jako zdroj, který používá *Artisan* pro vytvoření či upravení tabulek v databázi. Následují ukázky takové práce s migračními soubory pomocí nástroje *Artisan*.

*Migrate* je první z příkazů a nejvíc běžný. Používá se pro prvotní migraci všech migračních souborů – tabulek, do reálné databáze (informace o databázi včetně přístupových údajů jsou uloženy v souboru *env*, viz kapitolu o adresářové struktuře). To znamená, že jakmile se uživatel rozhodne, že jeho model v *PHP* a v migračních souborech je v pořádku, spustí *migrate* a změny se okamžitě projeví v jeho databázi. [7]

```
php artisan migrate
```

*Kód 27. Artisan příkaz pro spuštění migrace.*

*Rollback* je další z velmi běžných příkazů. Lidi dělají chyby, a proto se často stane, že se chceme vrátit o migraci zpátky, podívat se, jak vypadaly tabulky před poslední změnou. K tomu slouží příkaz *rollback*, který vrátí databázi do stavu o jednu nebo i více migrací zpět. [7]

```
php artisan migrate:rollback
```

*Kód 28. Ukázka příkazu rollback.*

*Refresh* je příkaz, který vrátí všechny migrace do počátečního stavu a poté spustí příkaz *migrate*, bez toho, aniž by mazal tabulky. [7]

```
php artisan migrate:refresh
```

*Kód 29. Ukázka příkazu refresh.*

*Fresh* je velmi podobný příkazu *refresh*, ale před samotnými migracemi vymaže všechny existující tabulky (SQL příkaz DROP). [7]

```
php artisan migrate:fresh
```

*Kód 30. Ukázka příkazu fresh.*

### 2.1.7 Seeding

Často se stane, že při vytváření aplikace, chceme tabulky naplnit daty. Tato data mohou být za účelem testování, mohou mít povahu představených hodnot jako jsou například role uživatelů nebo prvků, které očekáváme v aplikaci zabudované a pevně dané při jejím prvním spuštěním. K tomuto účelu nám slouží v *Laravelu* třída *Seeder*. Tuto třídu můžeme rozšířit vlastním definovaným *seederem*. Programátorem definovaný *seeder* je třída, která musí obsahovat metodu run. V této metodě pak specifikujeme data, která se mají naplnit do dané tabulky. Následující příklad demonstruje *seeder*, který naplní tabulku *players* fotbalovými hráči. [20]

```
class PlayersTableSeeder extends Seeder {
    public function run() {
        DB::table('players')->insert([
        ['id' => 1, 'name' => "Messi"],
        ['id' => 2, 'name' => "Ronaldo"],
        ['id' => 3, 'name' => "Ibrahimovic"],
        ['id' => 4, 'name' => "Henry"],
        ]);
    }
}
```

*Kód 31. Ukázka seederu.*

Existuje možnost zde též použít funkce jazyka *PHP*, jako jsou cykly, knihovny a načíst zde soubory z přiloženého tabulkového souboru (*MS Excel*, *OpenOffice Calc*). *Seeder* lze též vygenerovat pomocí nástroje *Artisan*. [8]

```
php artisan make:seeder PlayersTableSeeder
```

*Kód 32. Artisan příkaz pro tvorbu seederu.*

Pro provedení kódu v *seederu* je nutno použít následující skript. [8]

```
php artisan db:seed
```

*Kód 33. Artisan příkaz pro spuštění seederu.*

Po vykonaní příkazu obsahuje tabulka *players* 4 nové hráče.

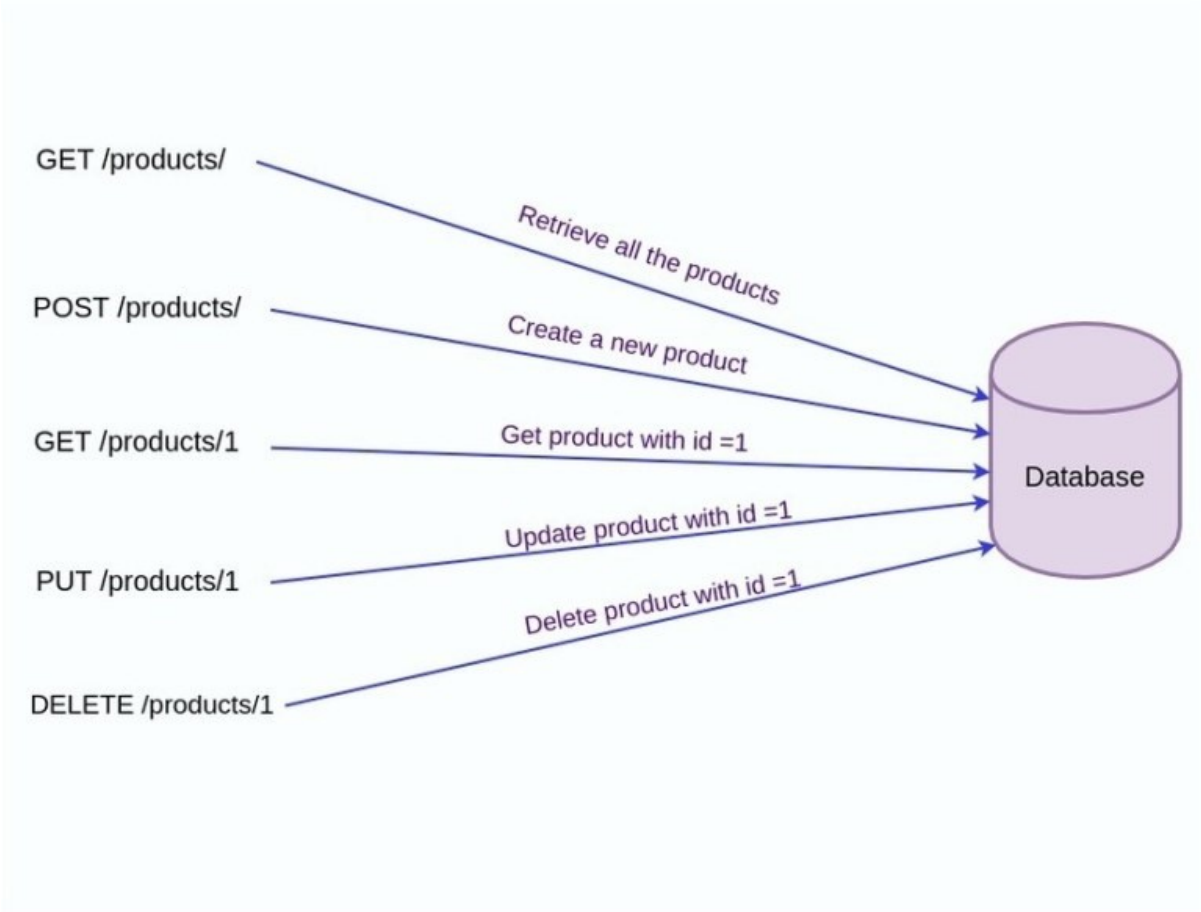
# 3 ROUTING

*Routovaní* tvoří pomyslnou mapu naší aplikace. Abychom správně pochopili, jak funguje *routování*, musíme pochopit, jak se tvoří požadavky a jak na ně Laravel reaguje. Laravel pracuje s HTTP požadavky. HTTP je protokol, který udává způsob síťové komunikace, s WWW servery. Definuje devět základních metod. Nyní popíšeme ty, které v aplikaci využíváme na denním pořádku.

| Metoda | Popis |
|---|---|
| GET | Požadavek na uvedený objekt se zasláním případných dat. |
| HEAD | Metoda podobná GET, avšak nepředává data |
| POST | Odesílá uživatelská data na server. Používá se například při odesílání formuláře na webu. |
| PUT | Nahraje data na server. Objekt je jméno vytvářeného souboru. |
| DELETE | Smaže uvedený objekt ze serveru. |
| CONNECT | Spojí se s uvedeným objektem přes uvedený port. Používá se při průchodu skrze proxy pro ustanovení kanálu SSL. |
| OPTIONS | Dotaz na server, jaké podporuje metody. |
| TRACE | Odešle kopii obdrženého požadavku zpět odesílateli. |
| PATCH | Používá se pro provedení malých změn. |

Obr. 9. HTTP metody. [24]

Největší zastoupení mají v aplikaci metody GET a POST. Metoda GET se volá kdykoli se například zobrazí nová stránka, metoda POST pak kdykoli chceme něco odeslat a uložit (například vytvořit uživatele, přidat auto do e-shopu a podobně). Routa je pak cesta (z anglického „*route*“ - cesta, kterou Laravel poskytne. Následující obrázek demonstruje metodu, *URL*, na které daná metoda reaguje a akci, která se provede.




Obr. 10. Laravel Routing. [25]

Aby Laravel věděl, která akce, má být provedena, je nutno specifikovat tento vztah v jednom ze dvou souborů k tomu určeným. Jedná se o soubory *web* a *api.* Do souboru *api* vepisujeme routy, které vedou ven z aplikace a jsou zpravidla používány aplikacemi třetích stran jako takzvané „*REST API*“ (*Representational State Transfer Application Programming Interface*). Soubor *web* na druhé straně uchovává routy, které používá přímo naše webová aplikace (pokud není čistě REST, ale má svou vizuální část, jako aplikace v praktické části práce). Následující ukázka kódu demonstruje soubor *web.php*, který obsahuje tři routy. První routa, když zaznamená URL ve tvaru "/", což bývá zpravidla první ("kořenová") stránka aplikace (může být však serverem specifikována jiná), odešle uživateli stránku s názvem *welcome.* Druhá *routa,* zaznamená URL ve tvaru „/druha“ a vykoná kód specifikovaný v závorkách pod ní. V tomto případě zobrazí uživateli HTML stránku s názvem „*druha*“. Třetí metoda demonstruje použití parametrů a *Controlleru*. Často se totiž stává, že akce, kterou chceme vykonat bývá složitější než jen zobrazení stránky. Můžeme například chtít uložit uživatele, kontrolovat heslo, stahovat či nahrávat soubory a podobně. Takový kód je dobré vyčlenit do zvláštního souboru, tzv. *Controlleru* (bude

popsán v následující kapitole). Ve třetím případě používáme tzv. parametr. Je to dynamický parametr *id*. Laravel za něj sám doplní číslo, které se vyskytne v URL. To je obzvlášť užitečné v případě, že chceme například zobrazit osobní stránku každého uživatele. Bez této funkce bychom museli mít tolik řádků cest, kolik máme uživatelů. Díky dynamickým parametrům nám stačí jeden řádek.

```
Route::get('/', function () {
    return view('welcome');
 });


Route::get('/druha', function () {

    return view('druha');

});

Route::get('/treti/{id}', 'SampleController@zobraz');
```

*Kód 34. Routovací soubor.*

Stejně jako metody *get* zde fungují i ostatní metody popsané výše.

## 3.1 Controller a Request

V kapitole o *routovaní* jsme se již s názvem *Controller* setkali a víme tedy, k čemu nám slouží. *Controller* je třída obsahující metody, kterým říkáme akce. K těmto akcím mapujeme cesty v *routovacích* souborech. *Artisan* nabízí možnost, jak vygenerovat takovou třídu, a to buď čistou nebo přímo se všemi metodami, které korespondují s *http* metodami. Jedná se o metody, které nám pomohou naplnit požadavky CRUD. CRUD je zkratka pro čtyři základní funkce pro manipulaci s daty ve světe webového vývoje – *Create*, *Read*, Update a *Delete*. Jedná se o metody, díky kterým jsou data vytvářena, čtena, upravována a mazána. Takový *Controller* bude vygenerován následujícím příkazem. [26]

```
php artisan make:controller CarController --resource
```

*Kód 35. Artisan příkaz pro tvorbu „CRUD“ Controlleru.*

Takto vygenerovaný *Controller* lze vidět na následující ukázce. Jedná se *Controller* pro práci s modelem Car (=Auto). V Laravelu je nezbytné, aby programátorem vytvořený *Controller* rozšiřoval základní třídu *Controlleru,* tzv. „*BaseController*“.

```
class CarController extends BaseController {

    public function index()
    {
        // zde získáme všechna auta
    }

    public function create()
    {
        // zde poskytneme formulář pro vytvoření auta
    }

    public function store(Request $request)
    {
        // zde ověříme a uložíme auto do databáze
    }

    public function show($id)
    {
        // zde získáme auto se zadaným ID
    }

    public function edit($id)
    {
        // zde je místo pro zobrazení formuláře k úpravě zadaného auta
    }

    public function update($id)
    {
        // zde získáme auto a upravíme
    }

    public function destroy($id)
    {
        // zde je místo pro smazání auta z databáze
    }
}
```

*Kód 36. Ukázka „CRUD“ controlleru.*

Metody CRUD, jsou metody „*store*“, „*show*“, „*edit*“ a „*destroy*“, v tomto pořadí. Ostatní metody, které *Artisan* vygeneroval jsou pro poskytnutí formulářů. Formulář pro vytvoření entity (metoda *create)* a pro upravení existující entity (metoda *edit*). V kódu výše si můžeme všimnout, že metoda *store* očekává parametr *request*, datového typu *Request.* [26]

*Request* se do metody store (potažmo do celé třídy *Controlleru)* připojí za pomoci *Dependency Injection* (technika, díky které může třída získat data, na kterých je závislá například z jiné třídy). Laravel má tuto techniku obsaženou v nadřazené entitě, tzv. *Service Containeru.* Pro tvoření aplikace v *Laravelu* není však podrobná znalost této třídy nutná, bakalářská práce se jí nebude zabývat. O *Service Containeru* je možné dozvědět se více v dokumentaci *Laravelu*. Proměnná *request* obsahuje data, která se uchovají při odeslání formuláře. S těmito daty pak můžeme v metodě pracovat a kontrolovat jejich vlastnosti, ověřovat, a nakonec je schvalovat a ukládat do databáze. [9] [27]

Metody s argumenty „*id“* získá *Laravel* díky dynamickým parametrům z *routy*, popsaných v kapitole o *routingu.*

# 4 DALŠÍ UŽITEČNÉ FUNKCE

*Laravel* poskytuje spoustu nástrojů a funkcí, které může programátor využívat při tvorbě aplikace.

## 4.1 Artisan

Nástroj *Artisan* poskytuje funkce pro vytvoření všech popsaných součástí MVC architektury. Nástroj se ovládá z prostředí příkazové řádky a do *Laravelu* byl implementován, aby byla práce programátora ještě více zpříjemněna a zrychlena. Důkazem toho jsou metody, které lze vidět v ukázkách kódu výše v této práci. *Artisan* je možno použít i pro ovládání databáze a dalších periferií. Veškeré dostupné příkazy nástroje *Artisan* lze vyčíst po zadání následujícího příkazu. [10]

```
php artisan list
```

*Kód 37. Ukázka výpisu příkazů nástroje Artisan.*

## 4.2 Facades

*Facade* je návrhový vzor který se často používá v objektově orientovaném programování. *Facade* je vlastně třída, obalující komplexní knihovnu, která má za úkol poskytnout dané knihovně jednodušší rozhraní. V *Laravelu* pojmem *Facade* rozumíme třídu, která poskytuje statické rozhraní službám (*Services*) uvnitř *IoC* (*Inversion of Control*) kontejneru. Příkladem takové *Facade* je *Auth*, což je *Facade* poskytující metody pro práci s přihlášeným uživatelem. [11] [32]

```
$loggedUser = Auth::user();
if($loggedUser->role != 'admin') {
    return 'You do not have permissions to do that';
}
```

*Kód 38. Použití Facade "Auth" pro kontrolu oprávnění.*

## 4.3 Práce se soubory

Často se stane, že v aplikaci je nutno pracovat se soubory, nahrávat fotky, dokumenty a ukládat je na server, aby mohly být později poskytnuty jiným uživatelům například skrz

grafické rozhraní. V praktické části práce si toho můžeme všimnout například u profilových fotek či dokumentů, které se připojují k souhlasu. *Laravel* poskytuje *Facade* pro práci se soubory s názvem *Storage*. V konfiguračním souboru si programátor může definovat disk, se kterým bude pracovat. Tento disk jsou ve své podstatě jen cesty, na kterých na serveru nalezneme soubory, které jsme skrz aplikaci uložili. [20]

```
use Illuminate\Support\Facades\Storage;

Storage::disk('images')->put('images/1', $content);

$image = Storage::get('image.jpg');
```

*Kód 39. Ukázka uložení a stažení souboru pomocí Storage.*

## 4.4 Autentizace

*Laravel* poskytuje metody pro práci s uživateli skrz *Facade* s názvem *Auth*. Často se stává, že programátor bude chtít ve své PHP aplikaci skrz uživatelské účty chránit data svých zákazníků. Stává se to dokonce tak často, že *Laravel* implementoval skrz nástroj *Artisan* možnost vygenerovat základní vzor pro autentizace uživatelů.

```
php artisan make:auth
```

*Kód 40. Vygenerování metod a tabulek pro autentizaci.*

*Facade Auth* poskytuje také základní metody pro práci s přihlášenými uživateli. Je to například metoda *user,* která poskytne referenci na právě přihlášeného uživatele, metoda *login,* která vytvoří *session* pro daného uživatele a dále metoda *logout,* která tuto *session* zničí. [15]

## 4.5 Frontend a CSRF

*Laravel* disponuje nástrojem *Blade*. *Blade* je engine pro šablony (jako například HTML), který zajistí kompilaci šablony do PHP souboru. Tento nástroj vnáší do šablon nové metody, které nám usnadní práci při zobrazování dat, které jsme pomocí *Laravelu* získali. Jednou z nejdůležitějších vlastností je výpis proměnných do šablony. Tento výpis se provádí pomocí složených závorek. *Laravel* na pozadí zavolá PHP funkci *htmlspecialchars*, čímž chrání před XSS útoky (*Cross-Site Scripting*). Další funkce *Blade* jsou například *foreach*

(cyklus pro výpis polí) a dále funkce, které usnadňují rozdělení pohledů do více souborů: *extends, section, yield, component* a *slot.* [12]

## 4.6 Testování

Psaní testů pro aplikaci je skvělou praxí při vývoji softwaru. Často je to první úloha, která se má provést ještě před vytvořením funkcí samotné aplikace. Zpočátku jsou testy, které napíšeme neúspěšné, protože nemáme žádný kód, který by vyhověl požadavkům testu. Jakmile naše testy selžou, můžeme si sestavit naši aplikaci, abychom mohli projít našimi testy. Tato praxe zajišťuje, že náš kód splňuje stanovené softwarové požadavky. Slouží také jako vodítko při rozšiřování naší aplikace nebo při refaktoringu. Tato praxe je běžně známá jako „Test-driven development“ (TDD). Testovací soubor lze vytvořit pomocí nástroje *Artisan* následujícím příkazem. [29]

```
php artisan make:test ProductTest --unit
```

*Kód 41. Vygenerování třídy pro testování.*

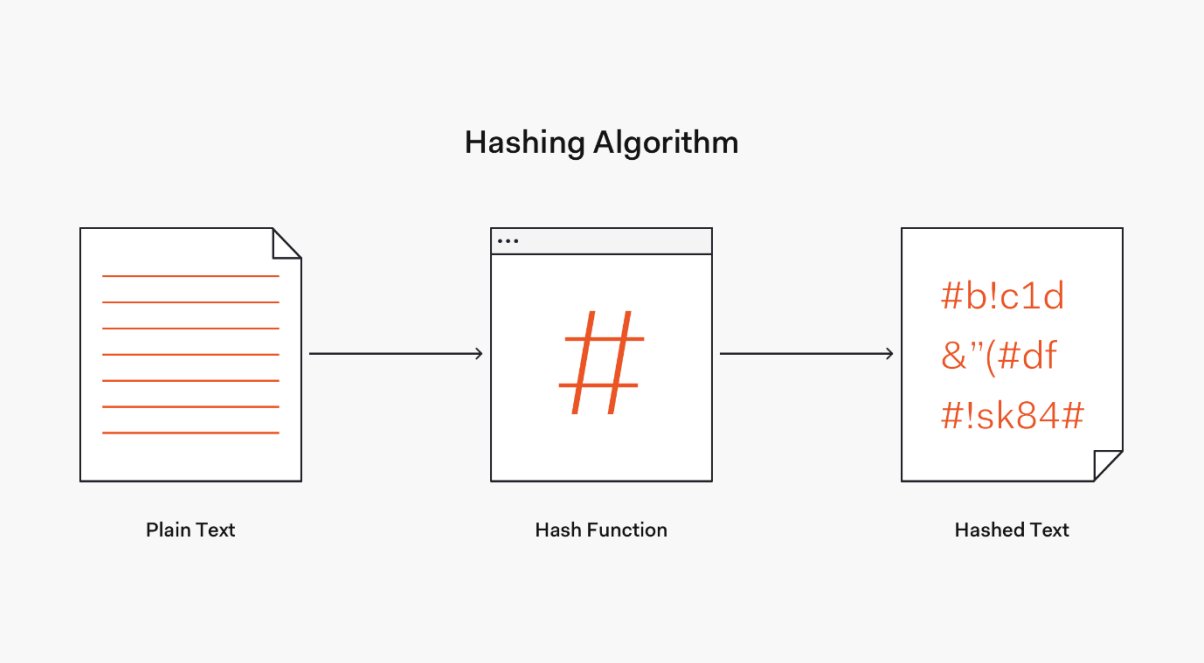
Takto vygenerovaný soubor bude obsahovat třídu *ProductTest*, ve které můžeme definovat metody pro testovaní jednotlivých jednotek. V takových metodách bude zvlášť užitečná funkce *assertJson*, kterou voláme na *response* a která kontroluje shodnost dat. [29]

*Faker* (https://github.com/fzaninotto/Faker) je open-source knihovna licencovaná pod MIT pro PHP, upravená pro Laravel a kterou Laravel na svých stránkách doporučuje používat. Je to databáze měst, jmen a jiných entit z reálného světa, které programátor potřebuje ve své databázi často ukládat a poskytuje tedy testovací prostředí bez nutnosti vymýšlet či shánět velká množství dat. [13]

## 4.7 Zabezpečení

Laravel jakožto Framework vyvíjen v době, kde se na bezpečnost klade opravdu velký důraz, disponuje spoustou funkcí a vzorů, které zaručí, že aplikace, kterou vytvoříme bude konkurenceschopná mezi ostatními svého druhu na poli bezpečnosti a zabezpečení. Hesla uživatelů je možné šifrovat pomocí *Facade Hash*. Toto zabezpečení funguje na principu kontroly tzv. *hashe* (zašifrované hodnoty hesel, které se zpětně dají přiřadit uživateli). *Fa-*

*cade Hash* používá metodu *Bcrypt*. Bcrypt je nástroj pro šifrování souborů na různých platformách. Šifrované soubory jsou přenosné ve všech podporovaných operačních systémech a procesorech. Hesla musí být v rozsahu 8 až 56 znaků a interně jsou *hashovány* na 448 bitový klíč. Všechny dodané znaky jsou však významné. Čím silnější je vaše přístupová fráze, tím bezpečnější jsou vaše data. [14] [30]




Obr. 11. Schéma *hashovací* funkce. [31]

# II. PRAKTICKÁ ČÁST

# 5 SBĚR POŽADAVKŮ

Jelikož je problematika praktické části práce spojena s tvorbou softwarové aplikace, je její prvotní a nevyhnutelnou částí sběr a analýza požadavků. Zde existuje několik metod, které můžeme použít. V tomto případě byla zvolena komunikace se zadavatelem. Znamená to, že programátor či projektový analytik si domluví přímo schůzku s člověkem, který buď vývoj aplikace zadal, nebo – ještě lépe – bude aplikaci přímo používat.

## 5.1 Problematika aplikace

Před samotnou funkcionalitou aplikace je potřeba mít přehled o problematice, kterou má softwarový systém řešit. V našem případě je problematika spojena s GDPR. GDPR je zkratka pro *General Data Protection Regulation* a jedná se zákon o protekci dat, týkající se všech občanů Evropské Unie. Tento zákon mimo jiné udává povinnost správců a zpracovatelů údajů bez ohledu na jejich velikost nebo počet zaměstnanců zavést technická opatření, která doloží, že fyzická osoba udělila souhlas se zpracováním svých údajů. [33]

## 5.2 Prvotní myšlenka a prototyp

Prvotní myšlenka fungování aplikace vznikla už dříve, při vytvoření tohoto tématu zadavatelem bakalářské práce, realizace této myšlenky však vyžadovala důkladnou analýzu. Vývoj začal vytvořením prototypu. To znamená, že byl vytvořen prvotní návrh aplikace s nastíněnou logikou. Již zde byla cítit velká výhoda použití *frameworku*. Funkční základ byl vytvořen za relativně krátkou dobu. Databáze prototypu obsahovala pouze tabulku uživatele a souhlasu, které byly nutné k demonstraci základního konceptu – uložit uživatele a jeho přiřazený souhlas. Co se týče samotné aplikace, ta obsahovala základní pohled, tabulku uživatelů a možnost přiřazení souhlasu. Tento základ byl následně prezentován před DPO univerzity, se kterou bylo nutné zhodnotit funkcionalitu. Z rozhovoru vzešly funkční a nefunkční požadavky.

## 5.3 Funkční požadavky

Funkční požadavky tvoří požadavky na funkcionalitu aplikace. Identifikují problémy, které musí umět daná aplikace řešit. Funkční požadavky vycházejí z jazyka UML (*Unified Modeling Language*) a jsou číslovány. Jelikož systém funguje ve dvou rolích, dělí se zde požadavky na tři části – požadavky obecné, požadavky pro administrátora a požadavky pro uživatele.

### 5.3.1 Obecné funkční požadavky

Následuje výčet získaných obecných požadavků.

Tab. 1. Obecné funkční požadavky.

| | |
|---|---|
| 1 | Systém bude umožňovat přihlášení uživatelů. |
| 2 | Systém bude umožňovat odhlášení uživatelů. |
| 3 | Systém bude umožňovat správu uživatele. |
| 4 | Systém bude umožňovat navigaci. |
| 5 | Systém bude umožňovat vizuálně rozlišit roli uživatele |
| 6 | Systém bude disponovat dvěma rolemi. |
| 7 | Systém bude disponovat seznamem kontaktů. |
| 8 | Systém bude umožňovat ukládat externí uživatele a externí požadavky. |

Nyní budou popsány výše uvedené požadavky.

Požadavek č.1: Systém nabídne přihlašovací formulář, kde uživatel vepíše své údaje a po autorizaci bude vpuštěn do aplikace.

Požadavek č.2: Systém umožní odhlásit uživatele a tím dovolí přihlášení jinému uživateli v podobě přihlašovací obrazovky.

Požadavek č.3: Systém umožní spravovat uživatele – například změnu profilového obrázku, zobrazení svých údajů.

Požadavek č.4: Systém po přihlášení zobrazí základní obrazovku s navigací do příslušných sekcí – dle role.

Požadavek č.5: Sytém umožní uživateli vizuálně rozlišit, zda je v roli administrátora nebo uživatele.

Požadavek č.6: Systém umožní rozlišit dvě role – administrátor a uživatel.

Požadavek č.7: Systém umožní uživateli zobrazit důležité kontakty související s GDPR nebo aplikací.

Požadavek č.8: Systém umožní komunikaci s aplikací z vnějšku pro externí uživatele. Informace o souhlasu s možností zrušení jim budou poslány emailem.

### 5.3.2 Funkční požadavky interního uživatele

Největší podíl s ohledem přístupu k aplikaci má právě interní uživatel, který zde může přímo spravovat své souhlasy. Je to například student dané univerzity či zaměstnanec korporace, využívající tento systém. Jeho hlavní akce vůči aplikaci spočívají ve správě osobních souhlasů, které mu byly přiřazeny. Patří mu následující požadavky.

Tab. 2. Funkční požadavky interního uživatele.

| | |
|---|---|
| 1 | Sytém bude umožňovat přihlásit se jako uživatel. |
| 2 | Systém bude umožňovat zobrazit jemu přiřazené souhlasy. |
| 3 | Systém bude umožňovat zobrazit historii přiřazeného souhlasu. |
| 4 | Systém bude umožňovat odmítnout souhlas. |
| 5 | Systém bude umožňovat hromadné přiřazení souhlasů přihlášenému uživateli. |

Požadavek č.1: Systém nabídne přihlašovací obrazovku, kde uživatel zvolí prostředí přihlášení (*Shibboleth*) a bude vpuštěn do systému jako uživatel.

Požadavek č.2: Systém nabídne uživateli možnost zobrazit seznam jemu přiřazených souhlasů. Tyto souhlasy budou seřazeny v tabulce, ve které bude možnost měnit způsob řazení či vyhledávat.

Požadavek č.3: Systém umožní zobrazit historii přiřazeného souhlasu. Seznam s přesným časem, kdy došlo ke změně stavu souhlasu.

Požadavek č.4: Systém umožní uživateli souhlas odmítnout.

Požadavek č.5: Pokud uživatel souhlas přiřazen nemá (nikdy ho nepřiřadil nebo ho odmítnul), bude v seznamu souhlasů pro hromadné přiřazení, kde uživatel vybere souhlasy, které chce přijmout.

### 5.3.3 Funkční požadavky externího uživatele

Externí uživatel je kdokoli, kdo podává souhlas související s danou institucí, která souhlasy spravuje, ale není jejím přímým členem, tudíž není důvod zřízení uživatelského účtu do systému dané instituce. Takový uživatel uděluje souhlasy skrz jiné webové stránky různých událostí (např. soutěž STOČ – Studentská odborná činnost, kterou spravuje univerzita a kde soutěžící podává souhlas se zpracováním osobních údajů) a s aplikací pro správu

souhlasů komunikuje pomocí tzv. API (Application Programming Interface). Své souhlasy spravuje pomocí emailu, kam mu přijdou veškeré informace a pokyny ke zrušení souhlasu. S tím souvisejí tyto funkční požadavky.

Tab. 3. Funkční požadavky externího uživatele.

| | |
|---|---|
| 1 | Systém bude disponovat API pro přidání souhlasu externímu uživateli. |
| 2 | Po přidání souhlasu systém pošle email externímu uživateli. |

Požadavek č.1: Rozhraní vyžaduje specifikovat email uživatele a souhlas, který přijímá.

Požadavek č.2: Přidání souhlasu vyvolá odeslání emailu. Email se pošle na adresu, kterou uživatel zadal při registraci souhlasu. Tento email poskytuje základní informace o souhlasu a možnost souhlas odmítnout.

### 5.3.4 Funkční požadavky administrátora

Následuje výčet získaných funkčních požadavků pro roli administrátora. Tato role je vlastně rolí pro DPO dané organizace, kde bude aplikace implementována. DPO funguje jako pověřená osoba a může tedy spravovat veškeré souhlasy a jejich vlastníky. Systémová hierarchie rolí využívá dědičnost, což administrátorovi umožňuje provést všechny úkony uživatele a navíc následující.

Tab. 4. Funkční požadavky administrátora.

| | |
|---|---|
| 1 | Systém bude umožňovat přihlásit se jako administrátor. |
| 2 | Systém bude umožňovat zobrazení uživatelů. |
| 3 | Systém bude umožňovat zobrazení souhlasů. |
| 4 | Systém bude umožňovat přidání souhlasů. |
| 5 | Systém bude umožňovat zobrazit informace o uživateli. |
| 6 | Systém bude umožňovat zobrazit historii souhlasů daného uživatele. |
| 7 | Systém bude umožňovat odmítnout souhlas zvoleného uživatele. |
| 8 | Systém bude umožňovat přidat přílohu k souhlasu. |
| 9 | Sytém bude umožňovat zobrazit externí uživatele. |

| | |
|---|---|
| 10 | Sytém bude umožňovat spravovat souhlasy externích uživatelů. |
| 11 | Sytém bude umožňovat hromadné přiřazení souhlasů. |

Požadavek č.1: Systém bude umožňovat přihlášení pro administrátora, po kterém bude mít zpřístupněnou vlastní jedinečnou funkcionalitu.

Požadavek č.2: Sytém bude umožňovat administrátorovi zobrazit všechny interní uživatele, kteří se registrovali do systému.

Požadavek č.3: Sytém bude umožňovat zobrazit všechny souhlasy, které byly vytvořeny, jejich datum vytvoření, popis a přílohu.

Požadavek č.4: Systém bude umožňovat přidání nového typu souhlasu do databáze, specifikovat jeho titulek, popis, přiřadit přílohu.

Požadavek č.5: Systém umožní administrátorovi zobrazit podrobné informace o daném uživateli.

Požadavek č.6: Systém umožní pro daného uživatele zobrazit historii souhlasů včetně data a přesného času kdy došlo ke změně. Seznam bude seřazený a bude možné v něm vyhledávat.

Požadavek č.7: Systém umožní vybranému uživateli odmítnout (popřípadě poté znova přijmout) jakýkoli souhlas, který je danému uživateli přiřazen.

Požadavek č.8: Systém umožní administrátorovi přiřadit textovou přílohu k daném souhlasu.

Požadavek č.9: Systém umožní administrátorovi zobrazit externí uživatele, kteří komunikovali s aplikací pomocí vnějšího aplikačního rozhraní.

Požadavek č.10: Systém umožní administrátorovi spravovat souhlasy, které externí uživatelé potvrdili, sledovat jejich historii apod. (viz zprávu souhlasů interního uživatele)

Požadavek č.11: Sytém umožní administrátorovi hromadně přiřadit souhlas. Toto přiřazení bude možné jak jednotlivým uživatelům – administrátor si zde vybere, kterým uživatelům bude tento souhlas přiřazen – tak jednotlivým pracovištím, kde si administrátor zvolí pracoviště a souhlas bude přiřazen všem uživatelům v daném pracovišti.

## 5.4 Nefunkční požadavky

Při návrhu aplikace se setkáme ještě s jedním typem požadavků. Jsou to požadavky, které neovlivňuje člověk, ale faktory prostředí. Patří sem požadavky na zabezpečení, formáty souborů, spolehlivost aplikace a podobné. V tabulce vidíme výpis nefunkčních požadavků pro systém, který je vyvíjen v praktické části.

Tab. 5. Nefunkční požadavky.

| | |
|---|---|
| 1 | Systém bude multiplatformní a fungovat skrz prohlížeč. |
| 2 | Systém bude dostatečně responzivní a bude nabízet standardní odezvu interakce. |
| 3 | Pro přílohy souhlasů bude systém podporovat vybrané typy textových souborů – viz popis. |
| 4 | Pro přílohy obrázků bude systém podporovat vybrané typy obrázkových souborů. |

Nefunkční požadavek č.1: Sytém bude součástí ekosystému aplikací univerzity, je potřeba aby byl multiplatformní a fungoval v prohlížeči. Tohoto požadavku bylo docíleno vlastní implementací v PHP a *frameworku Laravel*.

Nefunkční požadavek č.2: Systém bude zpracovávat požadavky v čase do 10 sekund.

Nefunkční požadavek č.3: Pro textové přílohy jsou podporovány tyto formáty: *doc, docx, pdf, txt.*

Nefunkční požadavek č.4: Pro obrázkové přílohy jsou podporovány tyto formáty: *jpeg*, *png*, *gif.*

## 5.5 Závěr z požadavků

Požadavky je nyní potřeba posoudit individuálně a navrhnout jejich zapracování do aplikace. Z návrhu požadavků lze vyčíst i podstatná jména, která budou později figurovat jako databázové entity. Jsou to například: souhlas, uživatel, externí uživatel atd. Této metodě se odborně říká analýza podstatných jmen a provádí se pro vytvoření diagramu tříd.

# 6 MODEL DATABÁZE

Návrh modelu databáze je klíčový moment, kdy je potřeba rozhodnout se, jakým způsobem budeme uchovávat a spravovat data, se kterými v aplikaci manipulujeme. V zásadě používáme dva typy databází – bezrelační, tzv. *NoSQL* databáze a relační, *SQL* databáze. V projektu je použita relační *SQL* databáze typu *MySQL (*https://www.mysql.com/). Relační databáze nám dovolují používat relace – vztahy mezi jednotlivými entitami databáze. Tím zajišťují zachování integrity databáze. Nejprve budou popsány modely s jejich atributy zvlášť a poté bude popsáno propojení entit při pohledu na databázi jako na celek. Díky ORM *Laravelu* má každá tabulka automaticky atributy *created_at* a *updated_at*, které se zaplní údajem typu TIMESTAMP, pokud dojde k vytvoření nové entity (*created)* nebo k aktualizaci některého z atributů (*updated*).

## 6.1 Uživatel

Uživatel je reprezentován tabulkou users (konvence udává používat množné číslo v názvu tabulek). Tabulka disponuje následujícími atributy:

Atribut *id,* který slouží jako jednoznačný identifikátor s automatickou inkrementací.

Atribut *name*, který drží křestní jméno uživatele.

Atribut *surname*, který drží příjmení uživatele.

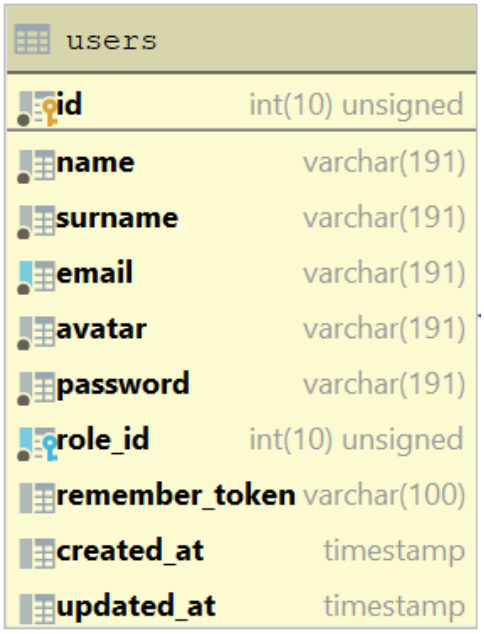
Atribut *email*, který může sloužit jako druhý jednoznačný identifikátor, jelikož neexistují na světě dva stejné emaily.

Atribut *avatar*, který slouží pro uložení cesty k profilovému obrázku, aby byl obrázek vždy přiřazen správnému uživateli.

Atribut *password*, který ukládá “zahashované” heslo uživatele.

Atribut *role_id*, což je cizí klíč k primárnímu klíči tabulky rolí (viz níže).

Atribut *remember_token* se zaplní unikátním tokenem, který se vytvoří, pokud uživatel při přihlášení zvolil volbu “zapamatovat si mě”.



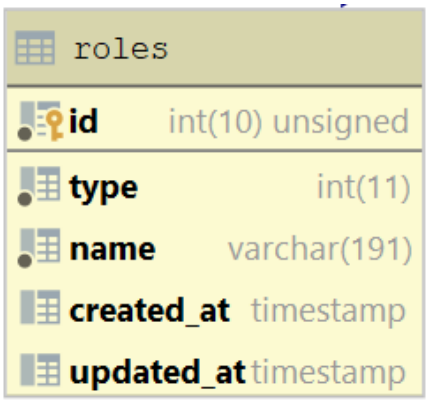

Obr. 12. Tabulka users.

## 6.2 Role

Jedním z požadavků je rozřazení uživatelů do rolí – administrátor a uživatel. Toho je docíleno pomocí tabulky rolí. U role nám stačí udržovat – kromě časových atributů a identifikátoru – pouze dva atributy.

Atribut *type*, který popisuje číselný kód role.

Atribut *name*, který slovem popisuje roli. Je to tzv. deskriptor.




Obr. 13. Tabulka roles.

## 6.3 Informace

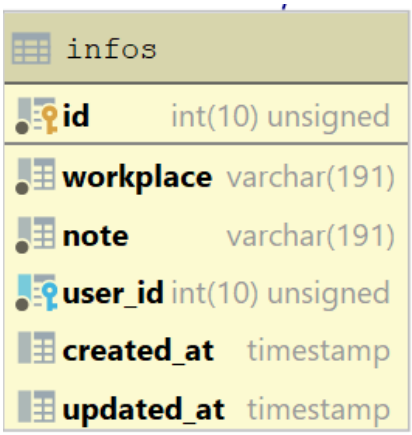
Dalším z požadavků je uchovávat informace o uživateli další informace. Jsou to informace o pracovišti, kterého je součástí a popřípadě poznámka k uživateli (například pokud je to

jeho dočasné pracoviště). Zde je myšleno i na to, že jeden uživatel může být součástí více pracovišť. Toho je docíleno cizím klíčem v této tabulce. V tabulce se nachází 6 atributů. Zajímavé jsou zejména tyto.

Atribut *workplace,* který uchovává informace o pracovišti.

Atribut *note*, který slouží pro uchování poznámky k uživateli.

Atribut *user_id*, který je cizí klíč pro tabulku uživatelů. Díky tomuto cizímu klíči můžeme každému uživateli vytvořit pole informací, tudíž jeden uživatel může mít několik pracovišť a poznámek. Tomuto se říká vazba 1:M.



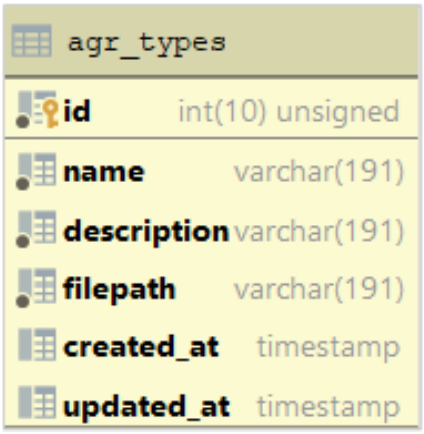

Obr. 14. Tabulka infos.

## 6.4 Souhlas

Souhlas je reprezentován tabulkou *agr_types*, ve které uchováváme typ souhlasu, veškeré informace o něm. Tento souhlas pak přiřazujeme uživatelům.

Atribut *name*, drží název souhlasu a atribut *description* popis souhlasu.

Pro potřeby uložení souboru k souhlasu je podobně jako u profilového obrázku uložena cesta k souboru jako atribut *filepath*.



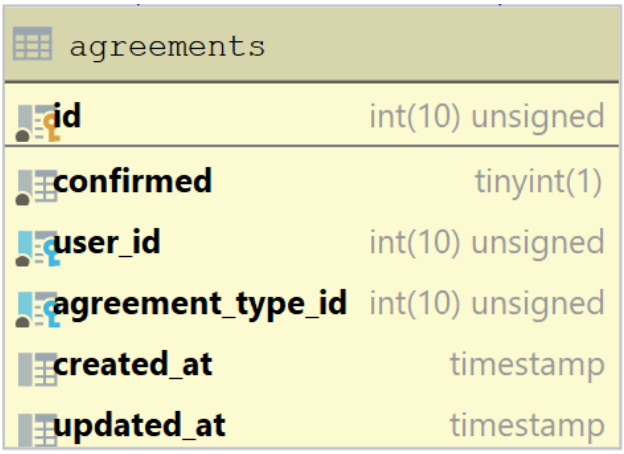

Obr. 15. Tabulka agr_types.

## 6.5 Propojení souhlasu a uživatele

V teoretické části, bakalářská byla popsána skutečnost, že vazby M:N jsou v relačních databázích realizovaný pomocí tzv. propojovací tabulky. Tento fakt můžeme demonstrovat na propojení souhlasu a uživatele. Více uživatelů může mít více souhlasů. Tím zároveň naplníme požadavek: „Systém bude umožňovat zobrazit historii souhlasů daného uživatele.“ Jelikož v propojovací tabulce nebudeme uchovávat jen nové propojení, ale jakoukoli změnu na souhlasu (odmítnutí i přijetí souhlasu se projeví novým řádkem tabulky). Tuto tabulku nazveme *agreements*.

Atribut *confirmed* je typu *„*tinyint*(1)“*, který funguje jako *boolean* hodnota (pravda/nepravda). Používá se v aplikaci pro zjištění, zda byl souhlas přijat či odmítnut.

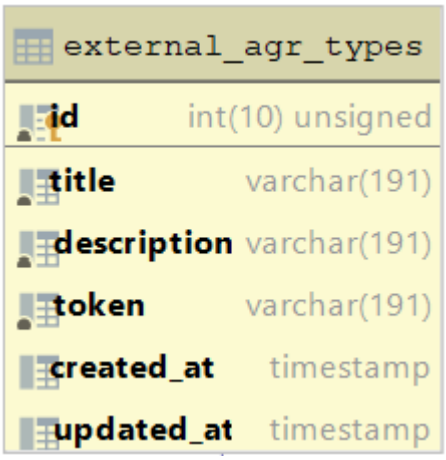
Atributy *user_id* a *agreement_type_id*, jsou cizí klíče pro napojení uživatele a typu souhlasu vazbou M:N. Takovou implementaci by bylo možné nahradit tzv. *compound – složeným* klíčem, avšak Laravel tuto možnost nepodporuje.




Obr. 16. Tabulka agreements.

## 6.6 Externí uživatelé

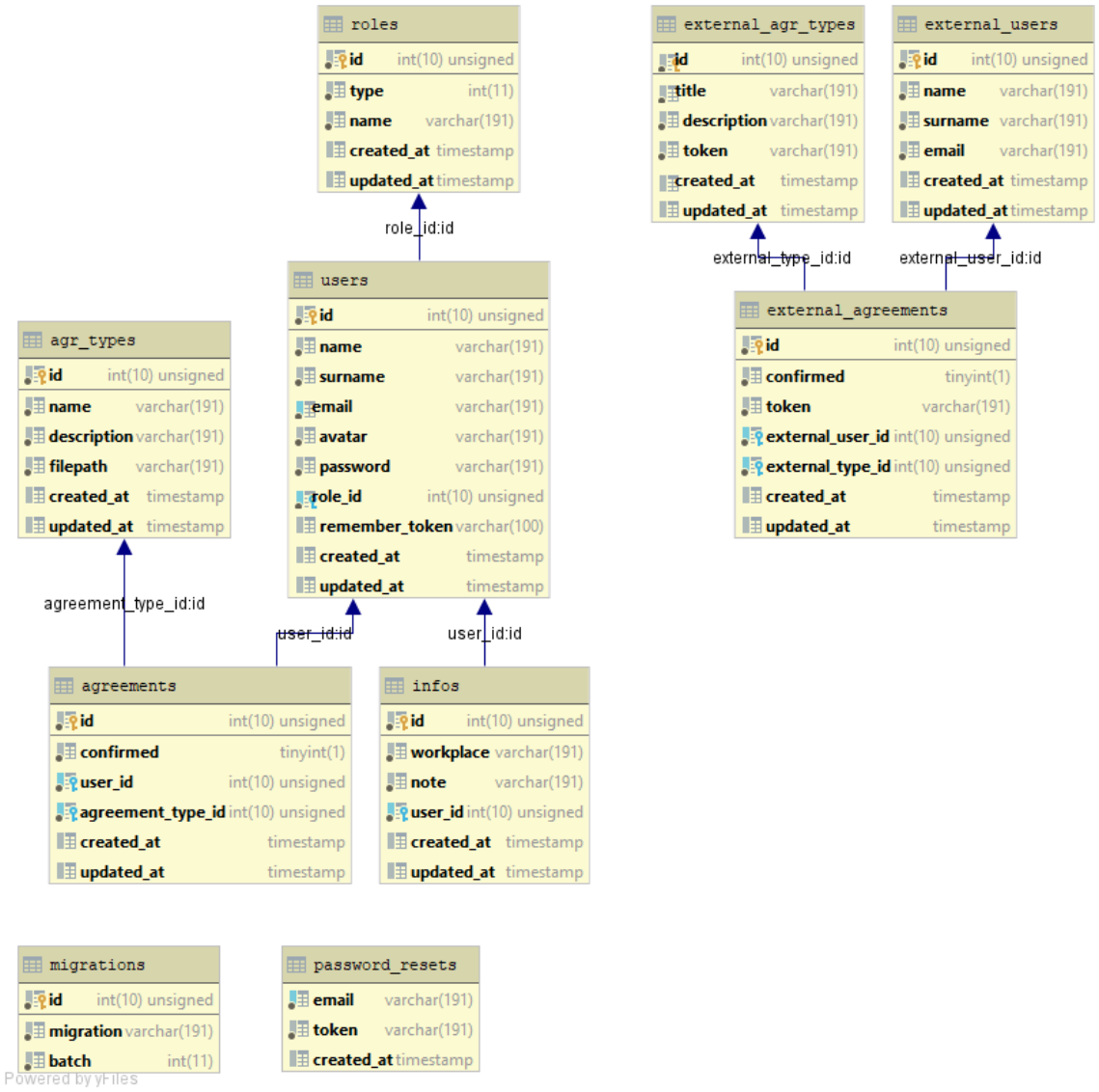
Struktura tabulek pro externí uživatele je totožná se strukturou interních souhlasů až na jeden rozdíl. Přítomnost dalších tokenů. Tabulka *external_agr_types* i tabulka *external_agreements* obsahují token, díky kterým může aplikace skrz aplikační rozhraní rozpoznat existující souhlasy či existující přiřazení. Vyhneme se tak duplicitám při ukládání souhlasů.




Obr. 17. Ukázka tokenu v tabulce external_agr_types.

## 6.7 Kompletní pohled

Následující obrázek demonstruje kompletní schéma databáze. Toto schéma, jakožto i ostatní schémata tabulek, je vytvořeno aplikací *DataGrip* od společnosti *JetBrains*, která mi poskytla studentskou licenci.

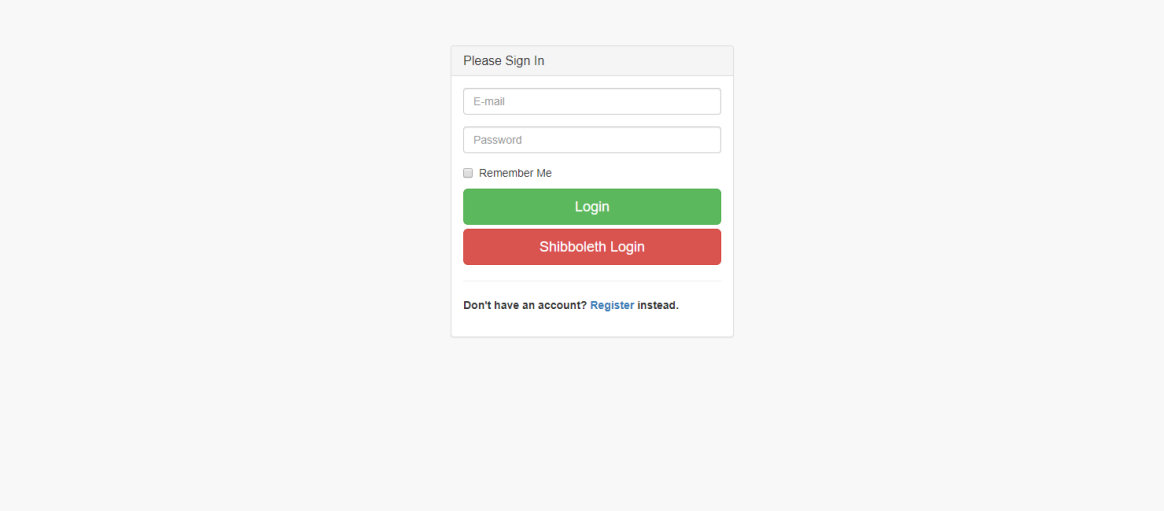



Obr. 18. Vizualizace kompletní databáze pro projekt GDPR.

# 7 IMPLEMENTACE

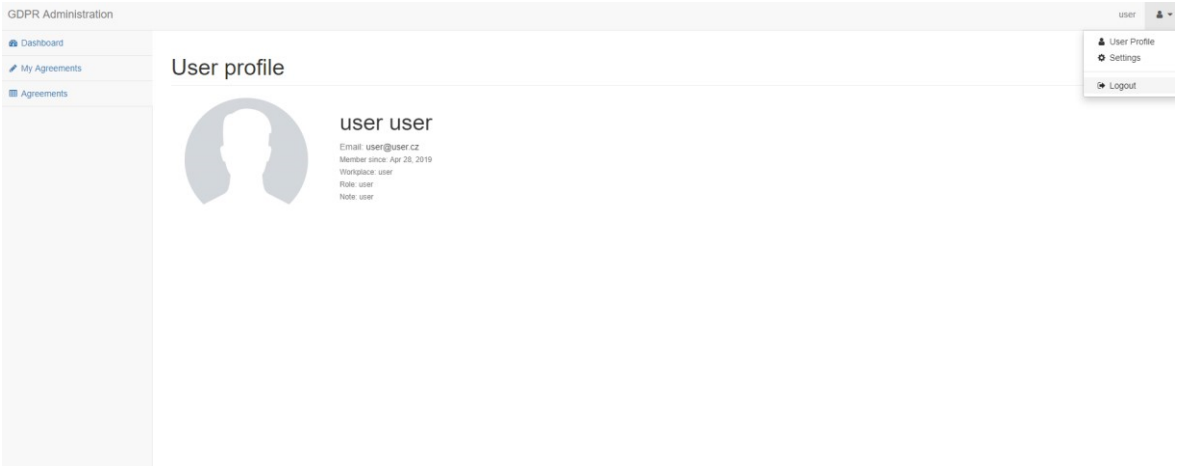
Na hotovou databázi je nyní možné začít stavit logiku aplikace. Výsledkem je několik *views*, které jsou ovládány a volány pomocí metod v *controllerech.*

První *view*, který bude demonstrován je *view* pro přihlašování. V tomto pohledu se přihlašují uživatelé. Mají zde možnost zvolit „*remember me*“, což invokuje vložení *tokenu* do tabulky a zapamatování si přihlášení. Je zde též možnost zvolit jiný typ přihlášení – skrz manažer identit *Shibboleth (*www.shibboleth.net/). Pro design aplikace je použit *Bootstrap template SB Admin* (https://startbootstrap.com/templates/sb-admin/). Tento *template* je zdarma k použití a poskytuje design administrátorským aplikacím, které mají za úkol různou správu. Vývojářům patří velké díky.




Obr. 19. Přihlášení uživatelů.

Po přihlášení může uživatel pomocí menu v pravém horním rohu spravovat svůj účet. Zobrazit své údaje a změnit profilový obrázek. Je zde také odhlášení.



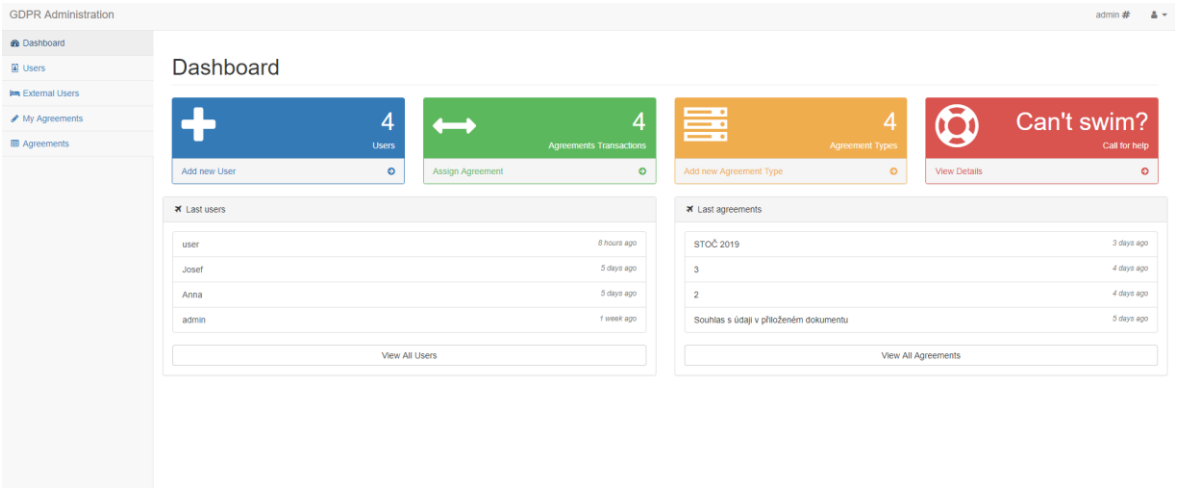

Obr. 20. Informace o profilu

Nyní se začne aplikace lišit v závislosti na roli. Nejprve bude popsána role administrátora a poté též role uživatele.

## 7.1 Interakce s aplikací pro administrátora

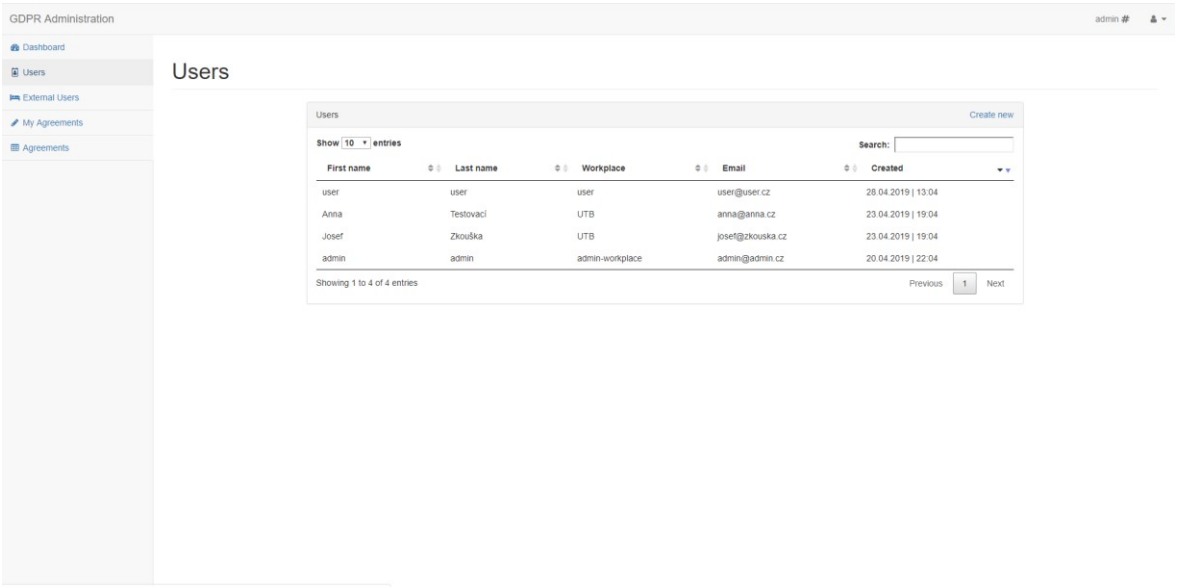
Uživatel v roli administrátora vidí v pravém horním rohu mřížku. To je znamení, že pracuje jako administrátor. Tato značka je známa ze světa operačního systému Unix, kde označuje tzv. *root* uživatele – uživatele s plnými právy v systému.

Administrátor po přihlášení dostane k dispozici hlavní přehled, tzv. *dashboard.* Zde má k dispozici dlaždice s akcemi, které jsou nejčastěji používány. Jsou to akce na přidání uživatele, přirazení souhlasu nebo též stránka s pomocí, která obsahuje kontakty na správce a tvůrce aplikace.



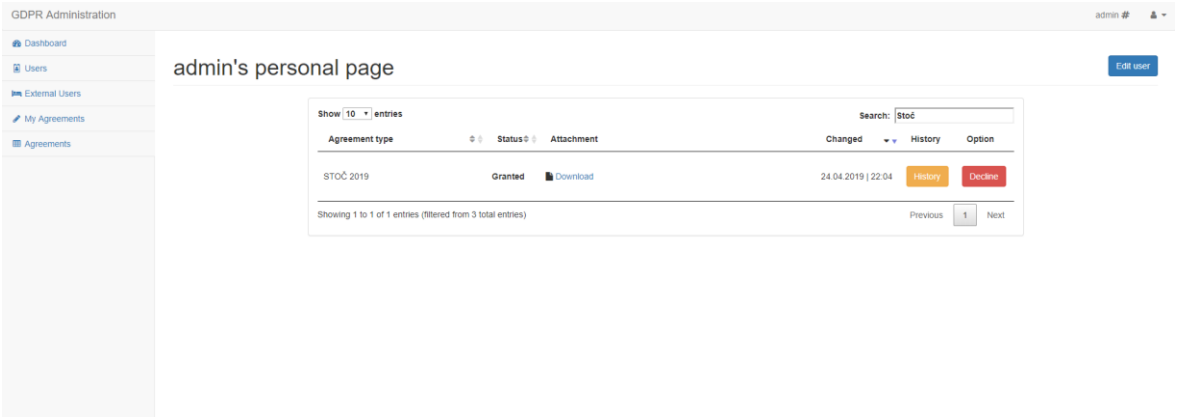

Obr. 21. Hlavní přehled – dashboard.

Pomocí navigace nebo dlaždic se administrátor může dostat k dalšímu pohledu – Uživatelé (*Users*). Tento pohled obsahuje strukturovaný seznam všech uživatelů. V seznamu se dá vyhledávat a dá se též řadit. Tím je splněn další z funkčních požadavků. Tabulka též podporuje stránkování. Implementace tabulky je pomocí služby *Datatables*, což je *plug-in* pro jazyk *JQuerry* (https://datatables.net/). Tabulka obsahuje jméno a příjmení uživatele, jeho pracoviště, email a datum vytvoření jeho účtu. Administrátor může kliknout na uživatele a tím zobrazit jeho souhlasy.



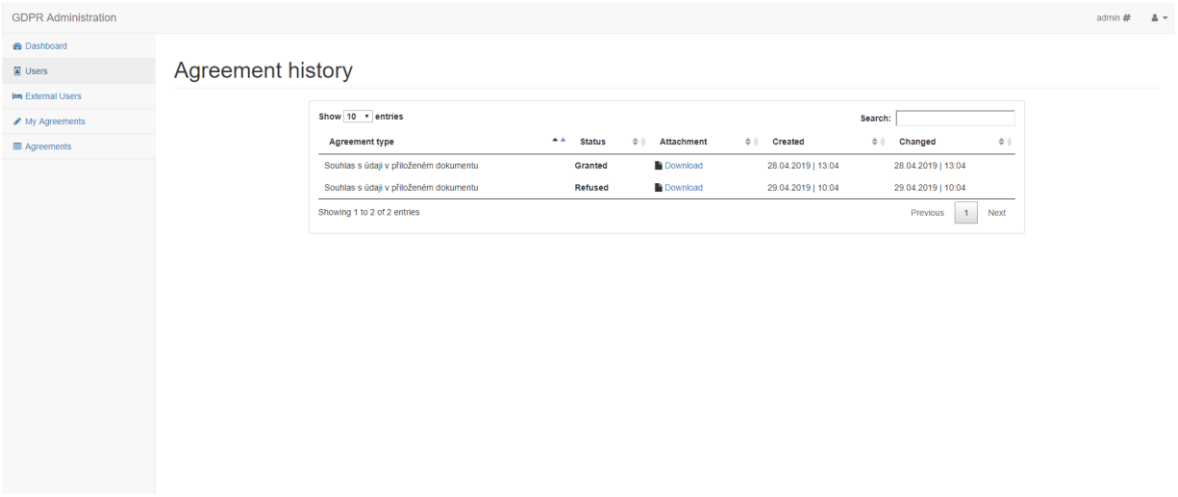

Obr. 22. Tabulka uživatelů.

Administrátor má možnost zvolit jednoho z uživatelů a tím se mu zpřístupní další pohled. Pohled obsahuje seznam všech unikátních typů souhlasů, které byly uživateli kdy přiřazeny. Zobrazuje se vždy poslední změna. Je zde datum přiřazení, respektive odebrání souhlasu, název souhlasu a možnost stažení přílohy. Administrátor má možnost zde přílohu i nahrát. V seznamu je možnost jednoduše vyhledávat ve všech sloupcích.




Obr. 23. Osobní stránka uživatele.

Předchozí pohled kromě schválení či odebrání souhlasu nabízel ještě jednu možnost. Zobrazit historii. Pohled historie souhlasu je velmi důležitý a je to opět další požadavek zadavatele. Historie slouží administrátorovi k vyřešení různých situací, kde potřebuje zjistit, zda v danou chvíli v minulosti uživatel souhlas dal. Je to jeden z klíčových funkčních požadavků v rámci legislativy GDPR.

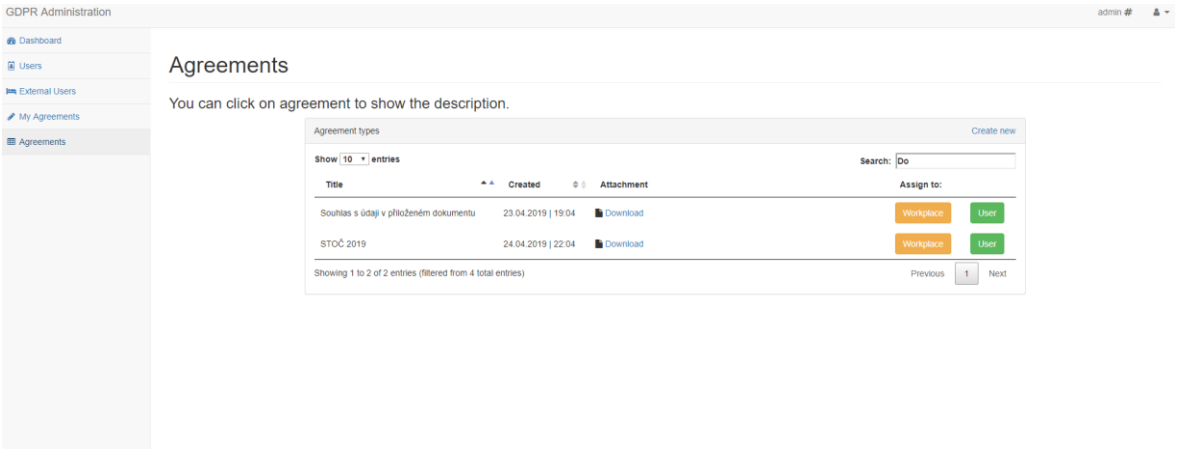



Obr. 24. Historie souhlasů.

Další pohled, který aplikace pro administrátora nabízí je pohled na externí uživatele. Tento pohled je stejný jako pohled na interní uživatele s veškerou funkcionalitou. Historie, správa souhlasů.

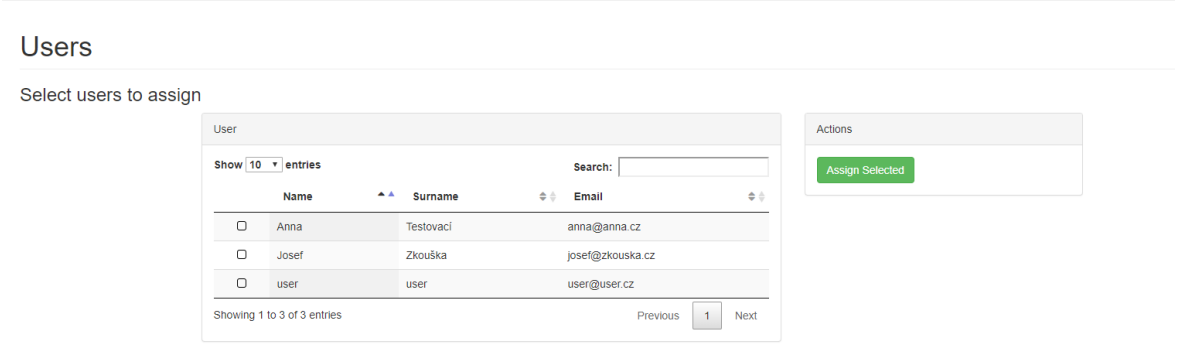
Další z pohledů je osobní stránka přihlášeného uživatele. Ta je klíčová hlavně pro běžného uživatele a bude tedy popsána v sekci interakce pro uživatele.

Poslední pohled v navigační liště je pohled na typy souhlasů, kde má administrátor k dispozici přehled všech typů souhlasů.




Obr. 25. Typy souhlasů.

Tento pohled je unikátní pro administrátora, jelikož zde může zvolit hromadné přiřazení souhlasů. Souhlasy lze hromadně přiřadit buď pracovišti nebo konkrétním uživatelům. Na dalším obrázku je ukázka hromadného přiřazení uživatelům. V tabulce vidíme všechny uživatele, kteří ještě daný souhlas nemají. Stačí jednoduše vybrat uživatele, kterým chceme souhlas přiřadit a kliknout tlačítko „přiřadit vybrané“. Po krátkém obnovení z tabulky zmizí uživatelé, kterým byl souhlas již přiřazen.

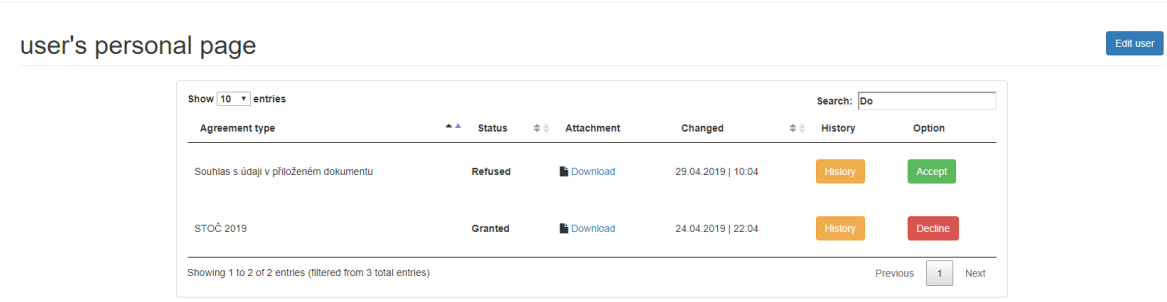



Obr. 26. Hromadné přiřazení uživatelům.

## 7.2 Interakce s aplikací pro uživatele

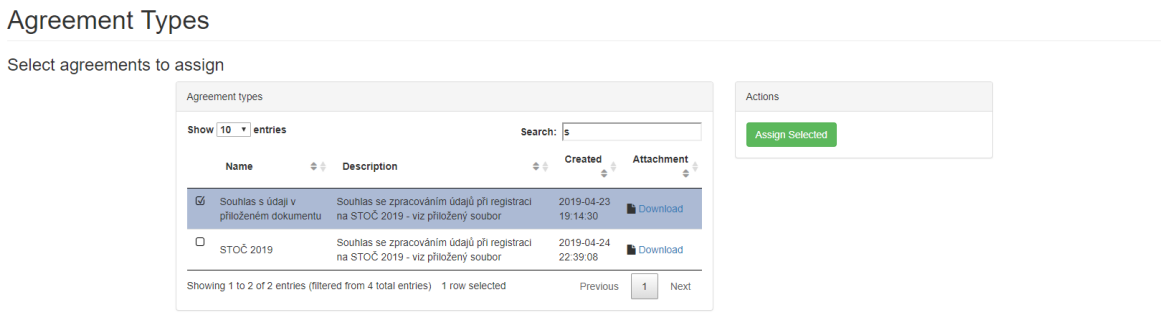
Funkcionalita pro uživatele se značně liší od funkcionality pro administrátora. Vzhled úvodní stránky zůstává stejný, dlaždice však vedou k jiným akcím. Akcím, které jsou tentokrát nejběžnější pro uživatele. Je to například výpis vlastních souhlasů nebo osvojení si

nového souhlasu. V navigaci též ubylo několik odkazů. Jsou tu jen dva. Jeden z nich je položka *„My Agreements“* – moje souhlasy, což je osobní stránka souhlasů přihlášeného uživatele. Klíčová funkcionalita pro interní uživatele, jejíž popis byl v této práci v sekci administrátora záměrně vynechán. Interní uživatel zde vidí své souhlasy, které mu byly v minulosti přiřazeny. Tyto souhlasy může odmítnout či znovu přijmout. Je zde i slovně a barevně odlišeno které souhlasy přijal a které ne, pro účely například tisku. Je zde i možnost prokliku na historii daného souhlasu, která je totožná s historií popsanou u administrátora. Opět je zde vidět snaha zachovat koncepční i vizuální stránku.




Obr. 27. Osobní stránka uživatele.

Poslední stránkou dostupnou pouze pro uživatele, které bude v práci demonstrována, je stránka přiřazení souhlasů. Uživatel si zde může jednoduše vybrat všechny souhlasy, u kterých chce vyjádřit souhlas a kliknout na tlačítko *„Assign Selected“* – přiřadit vybrané souhlasy.




Obr. 28. Přiřazení vybraných souhlasů.

## 7.3 Interakce externích uživatelů

Externí uživatel je uživatel, který nemá vlastní účet. Nekomunikuje s aplikací jako interní uživatel, skrz grafické rozhraní, ale pomocí aplikačního rozhraní. Webová služba, která poskytuje komunikaci skrz takové rozhraní pak jednoduše odešle údaje o uživateli a souhlasu skrz API (rozhraní) do systému GDPR. Sytém pak takového uživatele uloží a pošle mu výše zmiňovaný email. Email je odeslán prostřednictvím SMTP serveru (*Simple Mail Transfer Protocol*).

# 8 NASAZENÍ A ZABEZPEČENÍ

Kapitola nasazení a zabezpečení jsou sloučeny, jelikož tyto procesy musí jít ruku v ruce. Pokud nasazujeme aplikaci, musíme aplikaci zabezpečit.

## 8.1 Nasazení

Aby mohl být Laravel projekt na server nasazen, musí server splňovat některé požadavky. Jedním z požadavků je nainstalované samotné PHP a některé další PHP balíčky. [5]

- *PHP >= 7.1.3*
- *OpenSSL PHP Extension*
- *PDO PHP Extension*
- *Mbstring PHP Extension*
- *Tokenizer PHP Extension*
- *XML PHP Extension*
- *Ctype PHP Extension*
- *JSON PHP Extension*
- *BCMath PHP Extension*

Dále je potřeba mít instalovaný manažer balíčků *Composer* a webový server (například *Apache* nebo *Nginx*).

Pro vývoj aplikace, může být použit *Laravel Homestead.* Je to speciální obraz systému s předinstalovanými nástroji pro *Laravel* vývoj. Takový obraz byl použit i pro vývoj této aplikace. Obsahuje všechny nástroje, a navíc i *MySQL* databázový server. Server, na kterém je aplikace nasazena poskytla Fakulta aplikované informatiky Univerzity Tomáše Bati ve Zlíně. Jedná se o server s operačním systémem *Debian* a má následující specifikace.

CPU

- Procesor se čtyřmi jádry, každé s frekvencí 2GHz

RAM

- 8 GB fyzické paměti

Pevný disk

- 100 GB, přičemž je využito 8.2 GB

## 8.2 Zabezpečení

Klíčová otázka při tvorbě každého softwarového sytému je otázka ohledně zabezpečení. U projektu, který je vyvíjen v praktické části této práce se můžeme setkat se zabezpečením v několika vrstvách. Můžeme ho rozdělit do dvou skupin. Zabezpečení serveru a zabezpečení aplikace.

### 8.2.1 Zabezpečení serveru

Jedná se o zabezpečení počítače, na kterém jsou aplikace a uživatelská data uloženy. První vrstvou je dostupnost serveru. Server je dostupný pouze uvnitř privátní sítě univerzity. Pro přístup na server z místa mimo privátní síť univerzity je potřeba komunikovat skrz tzv. *VPN (Virtual Private Network)*, která je zabezpečená uživatelským jménem a heslem. Další vrstvou zabezpečení serveru tvoří samotný operační systém serveru. V tomto případě se jedná o Linuxovou distribuci *Debian*. Zde je potřeba mít účet a heslo s právy přístupu k projektu a také k samotnému *MySQL* RDBMS (*Relational Database Management System)*. Poslední vrstvou zabezpečení dat na serveru je nutnost administrátorského přístupu k databázi *MySQL,* který je chráněn heslem.

### 8.2.2 Zabezpečení aplikace

Po zabezpečení serveru je nutno zabezpečit samotnou aplikaci, ke které mají přístup uživatelé skrz své počítače v interní síti. Jednou z vrstev zabezpečení tvoří samotný *framework*, jehož možnosti zabezpečení jsou popsány v teoretické části v kapitole 4.7. Při přístupu do aplikace je vyžadován účet uživatele či administrátora. Z výše uvedených témat je patrné, že k aplikačním datům se může dostat pouze uživatel, jehož účet vytvořil administrátor (a to pouze k přístupu k datům daného uživatele) anebo samotný administrátor, což je osoba pověřená tato data vidět a spravovat a která má vlastní tajné přihlašovací údaje.

Při přihlášení uživatele je vytvořena unikátní *session*, která je po odhlášení zničena. Před uložením hesla v databázi je toto heslo *hashováno* a nikdy není zpět dekryptováno. Používá se tzv. kontrola otisku *hashe.*

Každý externí souhlas má vlastní *API* klíč, což je unikátní klíč pro jednoznačnou identifikaci souhlasu při jeho manipulaci. Tento klíč se posílá například v emailu a díky němu je zajištěno bezpečné spárování daného typu souhlasu s požadavkem na odmítnutí.

Jelikož se jedná o webovou aplikaci a komunikace probíhá skrz HTTPS. HTTPS protokol je protokol, který využívá protokol HTTP spolu s protokolem SSL. Využívá se zde kryptografických algoritmů k ověřování identity a zachování důvěrnosti přenášených dat.

# ZÁVĚR

GDPR je pro velké instituce obrovskou výzvou. Nejen, že je potřeba uchovávat veškeré souhlasy zaměstnanců, ale je též potřeba mít možnost tyto souhlasy zrušit a mít neustálý přehled o všech souhlasech daného zaměstnance.

Teoretický výstup práce popsal, jaké existují možnosti pro implementaci aplikace řešící výše uvedený problém pomocí *frameworku Laravel.* Byl uveden daný *framework*, stručně nastíněna jeho historie, jednoduchost instalace a následovalo seznámení čtenáře se základními pojmy ve světě tvorby aplikací na straně serveru. Byla popsána architektura daného *frameworku* a jeho úzké spojení s databází. V práci bylo též vysvětleno, jak můžeme tento *framework* využít pro tvorbu celé databáze jak pro malé, tak pro velké projekty. Spolu s tím byly uvedeny metody, které *framework* nabízí pro manipulaci s důležitými daty v databázi. Věnovala se též metodám zabezpečení, které *framework* nabízí.

Praktický výstup práce tvoří samotná implementace systému pro správu GDPR v popsaném *frameworku Laravel.* Paralelně s prací byla vyvíjena softwarová aplikace. Opírá se o zaužívané metody tvorby profesionálních softwarových projektů. Byl proveden důkladný sběr funkčních a nefunkčních požadavků na aplikaci, na základě schůzek se zadavatelem. Požadavky byly sepsány do přehledné tabulky a podrobněji popsány. V praktické části byla dále popsána databázová struktura navržená pomocí relační databáze (tabulky a jejich relace), čímž byl čtenáři poskytnut přehled o struktuře dat, uchovávaných v systému. Byla předvedena vizualizace jednotlivých tabulek a následně celého databázového modelu. Následně byly popsány role, které se v aplikaci využívají. Aplikace, která byla dle požadavků navržena a implementovaná byla následně v práci uvedena. Byly popsány jednotlivé funkční celky aplikace jak ze strany administrátora, tak ze strany uživatele. U významných celků byl poskytnut vizuální náhled do řešení. Spolu s tím se práce zaměřila i na důvody jednotlivých řešení. Změřila se i na způsob nasazení takové aplikace na produkční server. Při tomto nasazení byl využit školní server, na němž byla aplikace nejen nasazena, ale i testována.

Poslední kapitola práce se zabývá z určitého pohledu nejvíce důležitým aspektem vývoje jakékoli softwarové aplikace – jejím zabezpečením. Jsou zde popsány veškeré prvky zabezpečení, které byly využity při implementaci. Ať už je to zabezpečení serveru, zabezpečení požadavků na server nebo zabezpečení samotné aplikace.

# SEZNAM POUŽITÉ LITERATURY

[1] SURGUY, Maks, 2013. History of Laravel PHP framework, Eloquence emerging. In: *MAKSIM SURGUY PERSONAL WEBSITE AND BLOG* [online]. MAKSIM SURGUY [cit. 2019-05-08]. Dostupné z: https://maxoffsky.com/code-blog/history-of-laravel-php-framework-eloquence-emerging/

[2] Laravel Routing, 2018. *Laravel Documentation* [online]. TAYLOR OTWELL [cit. 2019 05-08]. Dostupné z: https://laravel.com/docs/5.6/routing

[3] What is PHP?, 2019. *PHP* [online]. The PHP Documentation Group [cit. 2019-05-08]. Dostupné z: https://php.net/manual/en/intro-whatis.php

[4] REHMAN, Noman Ur, 2015. What's New in Laravel 5. In: *Envatotuts+* [online]. [cit. 2019-05-08]. Dostupné z: https://code.tutsplus.com/tutorials/whats-new-in-laravel-5--cms-21842

[5] Laravel Installation, 2018. *Laravel Documentation* [online]. TAYLOR OTWELL [cit. 2019-05-08]. Dostupné z: https://laravel.com/docs/5.6/installation

[6] Laravel Structure, 2018. *Laravel Documentation* [online]. TAYLOR OTWELL [cit. 2019-05-08]. Dostupné z: https://laravel.com/docs/5.6/structure

[7] Laravel Migrations, 2018. *Laravel Documentation* [online]. TAYLOR OTWELL [cit. 2019-05-08]. Dostupné z: https://laravel.com/docs/5.6/migrations

[8] Laravel Seeding, 2018. *Laravel Documentation* [online]. TAYLOR OTWELL [cit. 2019-05-08]. Dostupné z: https://laravel.com/docs/5.6/seeding

[9] Laravel Container, 2018. *Laravel Documentation* [online]. TAYLOR OTWELL [cit. 2019-05-08]. Dostupné z: https://laravel.com/docs/5.6/container

[10] Laravel Artisan, 2018. *Laravel Documentation* [online]. TAYLOR OTWELL [cit. 2019-05-08]. Dostupné z: https://laravel.com/docs/5.6/artisan

[11] Laravel Facades, 2018. *Laravel Documentation* [online]. TAYLOR OTWELL [cit. 2019-05-08]. Dostupné z: https://laravel.com/docs/5.6/facades

[12] Laravel Blade, 2018. *Laravel Documentation* [online]. TAYLOR OTWELL [cit. 2019-05-08]. Dostupné z: https://laravel.com/docs/5.6/blade

[13] Laravel Database Testing, 2018. *Laravel Documentation* [online]. TAYLOR OTWELL [cit. 2019-05-08]. Dostupné z: https://laravel.com/docs/5.6/database-testing

[14] Laravel Database Hashing, 2018. *Laravel Documentation* [online]. TAYLOR OTWELL [cit. 2019-05-08]. Dostupné z: https://laravel.com/docs/5.6/hashing

[15] Laravel Authentication, 2018. *Laravel Documentation* [online]. TAYLOR OTWELL [cit. 2019-05-08]. Dostupné z: https://laravel.com/docs/5.6/authentication

[16] *Comoser Documentation* [online], 2019. Nils Adermann, Jordi Boggiano a komunita [cit. 2019-05-08]. Dostupné z: https://getcomposer.org/doc/

[17] Diagram of interactions within the MVC pattern., 2010. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, RegisFrey [cit. 2019-05-08]. Dostupné z: https://commons.wikimedia.org/wiki/File:MVC-Process.svg

[18] *Zpráva z konference MVC* [online], 1979. Trygve Reenskaug [cit. 2019-05-08]. Dostupné z: http://heim.ifi.uio.no/~trygver/1979/mvc-2/1979-12-MVC.pdf

[19] STAUFFER, Matt, 2016. *Laravel: up and running: a framework for building modern PHP apps*. Sebastopol, CA: O'Reilly Media. ISBN 978-1-491-93608-5.

[20] BEAN, Martin, 2015. *Laravel 5 Essentials*. Livery Place 35 Livery Street Birmingham B3 2PB, UK: Published by Packt Publishing. ISBN 978-1-78528-301-7.

[21] Data Mapper, 2003. In: *Martin Fowler* [online]. Martin Fowler [cit. 2019-05-08]. Dostupné z: https://martinfowler.com/eaaCatalog/dataMapper.html

[22] Data Mapper Sketch, 2003. In: *Martin Fowler* [online]. Martin Fowler [cit. 2019-05-08]. Dostupné z: https://martinfowler.com/eaaCatalog/databaseMapperSketch.gif

[23] DATE, C. J., 2015. *SQL and Relational Theory: How to Write Accurate SQL Code*. 2nd ed. Sebastopol: OReilly Media. ISBN 978-1449316402.

[24] HTTP Methods, 1992. In: *W3* [online]. MIT, ERCIM, Keyo, Beihang [cit. 2019-05-08]. Dostupné z: https://www.w3.org/Protocols/HTTP/Methods.html

[25] The Router, 2017. In: *WorldWideWeb101* [online]. Peakxel Online [cit. 2019-05-08]. Dostupné z: http://worldwideweb101.com/2017/10/10/build-a-react-app-with-a-laravel-restful-back-end-part-1-laravel-5-5-api/

[26] CRUD, 2015. In: *MDN Web Docs* [online]. Mozilla contributors [cit. 2019-05-08]. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/CRUD

[27] $_REQUEST, 2019. *PHP* [online]. The PHP Documentation Group [cit. 2019-05-08]. Dostupné z: https://www.php.net/manual/en/reserved.variables.request.php

[28] *MySQL Reference Manual* [online], 2019. Redwood City, California: Oracle Corporation [cit. 2019-05-08]. Dostupné z: https://dev.mysql.com/doc/refman/8.0/en/

[29] AFOLAYAN, Fisayo, 2018. How to write tests for Laravel applications. In: *Pusher Blog* [online]. FISAYO AFOLAYAN [cit. 2019-05-08]. Dostupné z: https://blog.pusher.com/tests-laravel-applications/

[30] *Bcrypt - Blowfish File Encryption* [online], 2002. Johnny Shelley [cit. 2019-05-08]. Dostupné z: http://bcrypt.sourceforge.net/

[31] ARIAS, Dan, 2018. Hashing Passwords: One-Way Road to Security A strong password storage strategy. In: *Auth0 Blog* [online]. Auth0 Blog [cit. 2019-05-08]. Dostupné z: https://auth0.com/blog/hashing-passwords-one-way-road-to-security/

[32] LAVARIAN, Reza, 2015. How Laravel Facades Work and How to Use Them Elsewhere. In: *Sitepoint* [online]. Sitepoint [cit. 2019-05-08]. Dostupné z: https://www.sitepoint.com/how-laravel-facades-work-and-how-to-use-them-elsewhere/

[33] ŠKORNIČKOVÁ, Eva, 2018. Souhlas se zpracováním osobních údajů. *GDPR.CZ Obecné nařízení o ochraně osobních údajů prakticky* [online]. Mgr. Eva Škorničková [cit. 2019-05-08]. Dostupné z: https://www.gdpr.cz/gdpr/heslo/souhlas-se-zpracovanim-osobnich-udaju/

# SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

HTTP *Hypertext Transfer Protocol.* Internetový protokol určený pro komunikaci s WWW servery.

HTTPS *Hypertext Transfer Protocol Secure*. HTTP protokol se zabezpečením typu SSL.

SSL *Secure Sockets Layer*. Protokol zabezpečující komunikaci dvou stran.

CRUD *Create Read Update Delete*. Zkratka pro metody vytvoření, čtení, aktualizace a mazání.

GDPR *General Data Protection Regulation*. Obecné nařízení o ochraně osobních údajů.

MVC *Model-View-Controller.* Architektonický vzor.

API *Application Programming Interface.*

SQL *Structured Query Language.* Jazyk pro práci s databázemi.

RDBMS *Relational Database Management Systém.* Systém používaný pro správu relačních databází.

SMTP *Simple Mail Transfer Protocol.* Komunikační protokol pro přenos e-mailů.

ORM *Object-relational mapping.* Technika práce s objekty.

DPO *Data Protection Officer.* Osoba pověřená prací s uživatelskými daty v rámci GDPR.

UML *Unified Modeling Language.* Modelovací jazyk pro analýzu *software*.

URL *Uniform Resource Locator.* Reference na webový zdroj v podobě adresy.

HTML *Hypertext Markup Language.* Jazyk pro programování webových stránek.

DB *Database.* Zkratka pro databázi ze softwarového hlediska. Databáze uchovává data.

CPU *Central Processing Unit.* Součást počítače, která vykonává instrukce.

VPN *Virtual Private Network.* Virtuální propojení počítačů do jedné sítě.

# SEZNAM OBRÁZKŮ

# SEZNAM TABULEK

# SEZNAM PŘÍLOH

# PŘÍLOHA P I. PŘILOŽENÉ CD – OBSAH

- Bakalářská práce ve formátu PDF
- Zdrojové kódy realizované softwarové aplikace
- Textový soubor s návodem pro instalaci