Webová aplikace pro párování předmětů pro studium v zahraničí

Kristýna Tomanová

Bakalářská práce
2021

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2020/2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Kristýna Tomanová**
Osobní číslo:	**A18512**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Webová aplikace pro párování předmětů pro studium v zahraničí**
Téma práce anglicky:	**A Web Application for Pairing Subjects for Study Abroad**

Zásady pro vypracování

1. Nastudujte a stručně popište technologie používané v rámci klient-server webových aplikací.
2. Popište základní strukturu a součásti vývojového rámce Laravel, který bude využit k praktické implementaci aplikace.
3. Sesbírejte a specifikujte požadavky na webovou aplikaci pro párování předmětů pro studium v zahraničí.
4. V rámci praktické části popište strukturu projektu webové aplikace, strukturu databáze a dále také implementačně zajímavé části.
5. Věnujte se také zabezpečení webové aplikace.

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

Seznam doporučené literatury:

1. UNHELKAR, Bhuvan. *Software Engineering with UML* [online]. Boca Raton: CRC Press, 2018 [cit. 2020-09-23]. ISBN 9781138297432. Dostupné z: databáze eBook Collection
2. Laravel Documentation. *Laravel: The PHP Framework For Web Artisans* [online]. Laravel, c2011–2020 [cit. 2020-09-23]. Dostupné z: https://laravel.com/docs/8.x
3. ALLANWOOD, Gavin a Peter BEARE. *User Experience Design: A practical introduction* [online]. Second Edition. London: Bloomsbury Visual Arts, 2019 [cit. 2020-09-23]. ISBN 978-1-3500-2172-3. Dostupné z: DOI:10.5040/9781350021723
4. STAUFFER, Matt. *Laravel: Up and Running.* Sebastopol: O'Reilly Media, 2016. ISBN 978-1-491-93608-5.
5. PIPINELLIS, Achilleas. *GitHub Essentials*. Birmingham: Packt Publishing, 2015. ISBN 978-1-78355-371-6.
6. EELES, Peter a Peter CRIPPS. *Architektura softwaru*. Brno: Computer Press, 2011. ISBN 978-802-5130-360.
7. SOMMERVILLE, Ian a Peter CRIPPS. *Softwarové inženýrství*. Brno: Computer Press, 2013. ISBN 978-802-5138-267.
8. MCDONALD, Malcolm. *Web Security for Developers: Real Threats, Practical Defense*. San Francisco: No Starch Press, 2020. ISBN 978-159-3279-943.

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

Datum zadání bakalářské práce: **15. ledna 2021**
Termín odevzdání bakalářské práce: **17. května 2021**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN
L.S.

doc. Mgr. Milan Adámek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D. v.r.
ředitel ústavu

Ve Zlíně dne 15. ledna 2021

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 bakalářské práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky. Univerzity Tomáše Bati ve Zlíně;
- 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 Univerzita Tomáše Bati 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 – 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 14. 5. 2021 Kristýna Tomanová v. r.

podpis studenta

ABSTRAKT

Bakalářská práce je zaměřená na vývoj webové aplikace pro Univerzitu Tomáše Bati, která pomáhá studentům se zájmem o studijní pobyt v zahraničí při výběru vhodné destinace. Aplikace vyhledává spárované předměty zahraničních univerzit, které již byly vyzkoušeny studenty univerzity, čímž zájemci umožní získat přehled o dostupných možnostech výjezdu. Studenti, kteří výjezd již absolvovali, mají možnost absolvované zahraniční předměty ohodnotit a pomoci ostatním zájemcům s rozhodováním.

Klíčová slova: Webová aplikace, výjezd, zahraniční studijní pobyt, párování předmětů, univerzity, PHP framework, Laravel.

ABSTRACT

The bachelor thesis deals with the development of a web application for Tomas Bata University that helps students interested in exchange study with deciding on the eligible destination. Application searches for matching courses from previous mobilities, which helps with an overview of possible options where to go. Students that already experienced the exchange can rate the subjects they undertook during the mobility to help the others with their choice.

Keywords: Web application, study exchange, learning agreement, courses, universities, PHP framework, Laravel.

Tímto bych chtěla poděkovat Ing. Radku Valovi Ph.D. za cenné rady, ochotu a trpělivost během vedení mé bakalářské práce. Dále také Bc. Ing. Pavlu Býčkovi a Mgr. Haně Toncrové za jejich podporu a spolupráci při návrhu a MgA. Aliaksandře Laurove za pomoc s designem aplikace.

OBSAH

ÚVOD

Zahraniční pobyty jsou velmi váženou zkušeností v životě. Rozhodnutí pro výjezd je jedno z nejdůležitějších rozhodnutí pro vysokoškolského studenta. Z průzkumů mezinárodního oddělení vyplývá [1], že všichni dotazovaní vyjíždějící studenti doporučují ostatním se studijního pobytu v zahraničí zúčastnit (85 % dokonce i v době pandemie).

Nicméně každá cenná zkušenost stojí hodně úsilí a komplikací, které často studenty odradí. Z průzkumu také vyplývá, že 24 % podaných přihlášek se ve skutečnosti neuskuteční. To znamená, že každý čtvrtý student ze sta, který pobyt plánoval, si nakonec výjezd rozmyslí.

Vyřizování výjezdu je poměrně komplikovaná záležitost, kdy je nutné vyřešit studijní smlouvy *Learning Agreement*, kde student rozhoduje o předmětech, které bude v zahraničí absolvovat. Univerzity většinou publikují seznam předmětů až v polovině předchozího semestru, což už je v době, kdy zájemce má podanou přihlášku. Student se tedy rozhoduje podle předmětů z předchozího roku, které ale nemusí být vypsány na semestr jeho výjezdu.

Další nepříjemné zkušenosti mohou vyplynout ze samotného pobytu. 30 % studentů není celkově spokojeno se svým výjezdem. To může způsobit špatná komunikace se zahraniční univerzitou, samotný průběh výjezdu, atp.

Dále 32 % studentů oznámilo zkušenost s nevhodným spárováním předmětů. K tomuto jevu dochází z několika možných důvodů. Student si svůj rozvrh volí až po příjezdu do zahraničí a může se tedy stát, že rozvrhové akce budou kolidovat a student si musí zvolit pouze jeden z těchto plánovaných předmětů. Dalším důvodem můžou být povinné prerekvizity kurzu, které student nesplňoval, nebo v případě bakalářského studia je předmět určen pouze pro studenty studia magisterského. Občas dochází k tomu, že kurz není otevřený z důvodu nenaplnění kapacity, nebo dalších důvodů, které určí univerzita. K tomuto slouží další smlouva *Changing Learning Agreement*, kterou je možné odeslat do jednoho měsíce od počátku výjezdu.

Pro vylepšení zážitku z pobytu zahraničí vznikl ve spolupráci s mezinárodním oddělením projekt **Matchversity**, který usnadní výběr univerzity pro zájemce o výjezd do zahraničí.

Matchversity je webová aplikace, která využívá data o mobilitách uložená v informačním systému STAG, a zpřístupňuje tak zkušenosti studentů, co se pobytu v zahraničí účastnili. Díky tomu může zájemce získat přehled o univerzitách, které s UTB navázaly spolupráci a studenti je využili k výjezdu. Student se tedy může rozhodnout pro již ověřenou instituci a mít tak větší jistotu úspěšného výjezdu.

Aplikace obsahuje informace i o uskutečněných či zrušených spárování předmětů

univerzity s předměty UTB. Díky tomu zájemce získá větší přehled o dostupných předmětech a zda je předmět statisticky úspěšně spárován či nikoliv.

Zároveň poskytuje odkaz na webové stránky univerzity. Ty často vypovídají o schopnosti komunikace se zahraničními studenty podle úrovně dostupných informací mezinárodního oddělení. Pokud tedy má instituce těžko dohledatelné informace a omezenou komunikaci se zájemcem o studium, lze z toho vyvodit horší spolupráci během studia.

Matchversity tedy již sama o sobě poskytuje důležité informace pro zájemce k výjezdu. Může být ale rozšířená o hodnocení studentů, kteří se studijního pobytu zúčastnili. Ti mohou poskytnout hodnocení samotných spárování a jejich spokojenosti s nimi. V případě, že bylo spárování zrušeno, mohou poskytnout důvod, proč k tomu došlo a tím upozornit zájemce na možnou špatnou zkušenost s předmětem. Informace poskytnuté studentem jsou však dobrovolné a jejich motivace je pouze sdílení svého zážitku s ostatními pomocí anonymního hodnocení kurzů.

Velkým přínosem aplikace je její jedinečnost. Globální vyhledávače univerzit nejsou schopny studentovi poskytnout vhodné předměty, které by na univerzitách mohl studovat. Tím, že aplikace pracuje přímo s daty mobilit UTB, jsou data pro studenta mnohem relevantnější a tím i přínosnější.

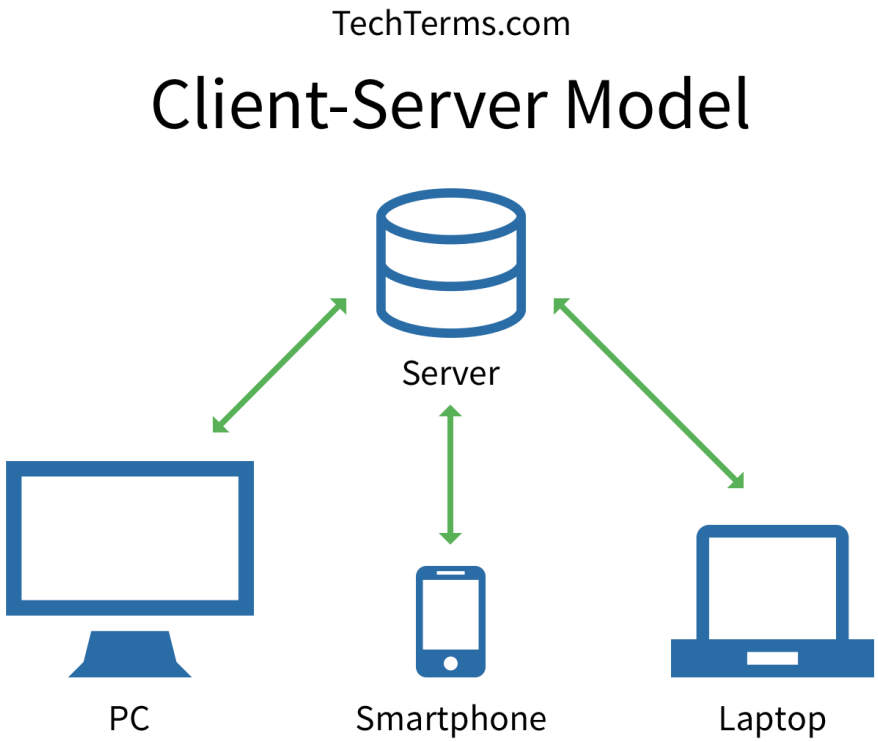
Očekávané výsledky tohoto projektu jsou, že aplikace bude prospěšná pro studenty plánující výjezd, ale také pro ty, kteří si nejsou jisti, zda studijní pobyt vyzkoušet. Spousta lidí má totiž obavy z neznáma a aplikace jim tedy může poskytnout stabilní zázemí, které pomůže získat větší jistotu a zkusit se přihlásit. Proto si slibujeme nárůst počtu přihlášek i uskutečněných výjezdů. Zároveň díky informacím o spárování a univerzitě věříme, že aplikace pozitivně ovlivní zkušenosti s výjezdy a jejich spárování.

I. TEORETICKÁ ČÁST

1 KLIENT-SERVER WEBOVÉ APLIKACE

Klient-server aplikace patří mezi distribuované aplikační struktury, které rozdělují úlohy mezi poskytovatele služby (tj. server) a žadatele o službu (tj. klient). Klientem je tedy myšleno přímo elektronické zařízení, které odesílá požadavek aplikaci. Serverem je vzdálený počítač, který poskytuje data či přístup ke službě [2].

Obrázek 1.1 znázorňuje vztah mezi serverem a klientem. Klient posílá požadavky (žádosti o službu) a server na ně odpovídá.



Obrázek 1.1 Model klient-server aplikace [3]

Webová aplikace se tedy skládá z klientské a server části. Jako klientská část se považuje *frontend* (přední část) softwaru, zatímco část serveru obstarává *backend* (zadní část).

1.1 Backend

Backend je část aplikace, která není viditelná klientovi. Věnuje se správě dat. Všechny skripty jsou vykonávány přímo na serveru a uživatel k nim nemá přístup. Mezi funkce *backend* patří:

- zpracování příchozích požadavků,
- spouštění skriptu ke generování HTML,
- přístup k datům uložených v databázi,
- ukládání nebo změna dat databáze,
- šifrování a dešifrování dat,
- manipulace se stahovanými a nahranými soubory,
- zpracování uživatelského vstupu [4].

1.1.1 PHP

Hypertext Preprocesor neboli PHP je open-source skriptovací jazyk určený pro práci se serverem. Soubory `.php` můžou obsahovat HTML, CSS, Javascript a kód PHP. Poté, co server zpracuje kód, vrátí odpověď v HTML.

Jazyk PHP je multiplatformní - lze jej využívat na všech hlavních operačních systémech. Programování v PHP může být jak procedurálně tak objektově orientované.

Začátek PHP skriptu je definován pomocí tagu `<?php`, za kterým následuje kód, který se vykonává do konce souboru nebo do uzavíracího tagu `?>`.

Jazyk PHP je vhodný pro:

- skripty pro server,
- skripty pro příkazový řádek,
- programování desktopových aplikací [5].

1.1.2 SQL

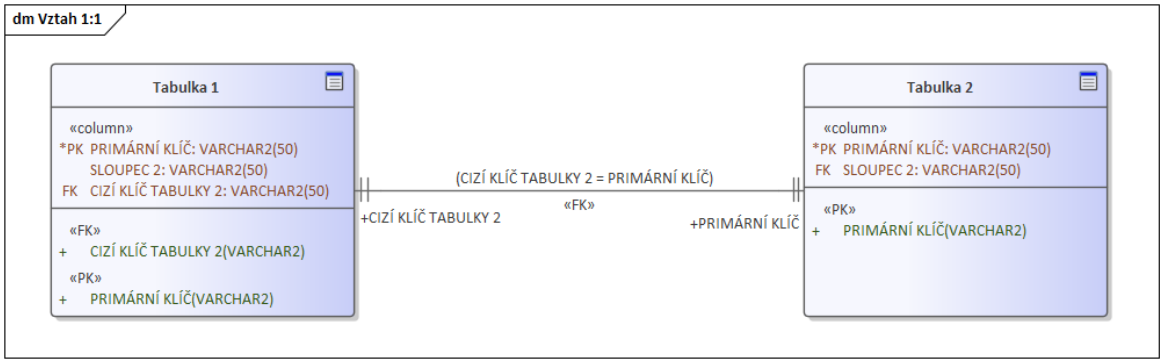
Structured Query Language je jazyk pro práci s relačními databázemi. Slouží k ukládání, změnám, čtení a mazání dat. Jedná se o standardizovaný jazyk pro relační databázové systémy jako je například MySQL, Oracle nebo MSSQL.

Relační databáze organizuje data do tabulek, které se navzájem mohou propojovat. Jejich sloupce určují jednotlivé atributy dat, zatímco každý řádek definuje jeden uložený

záznam. Systém relačních databází vychází z matematického konceptu, se kterým přišel v 70. letech minulého století Edgar F. Codd [6].

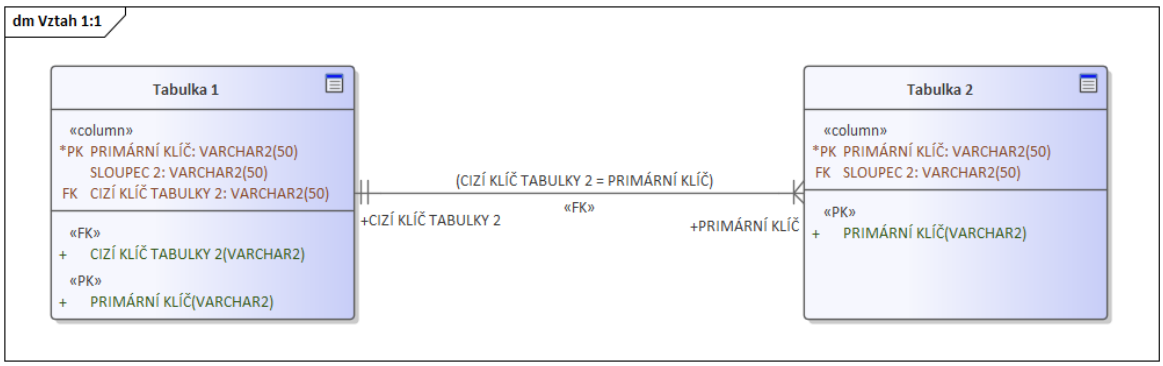
Tabulky jsou minimalizovány tak, aby řádky obsahovali jedinečná data. Propojení je definováno podle vztahů, které určují jejich vazbu.

Vztah 1 : 1 Jeden záznam v tabulce je propojený s právě jedním záznamem v tabulce druhé, jak znázorňuje obrázek 1.2.



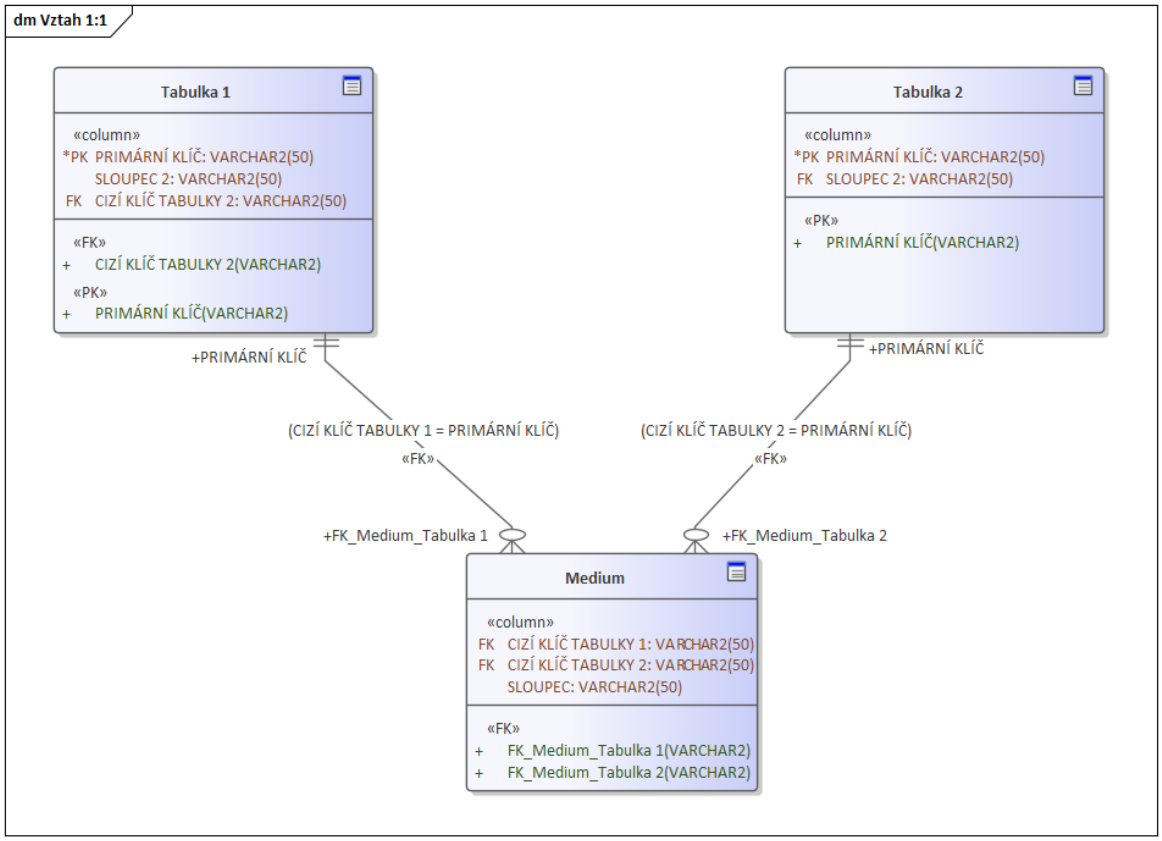
Obrázek 1.2 Vztah 1:1 (vlastní zpracování)

Vztah 1 : n Jeden záznam v tabulce může být propojený s více záznamy druhé tabulky. Tento vztah je znázorněn na obrázku níže (Obr. 1.3).



Obrázek 1.3 Vztah 1:n (vlastní zpracování)

Vztah m : n Každý záznam jedné tabulky může být propojený s více záznamy tabulky druhé jako na následujícím obrázku (Obr. 1.4).

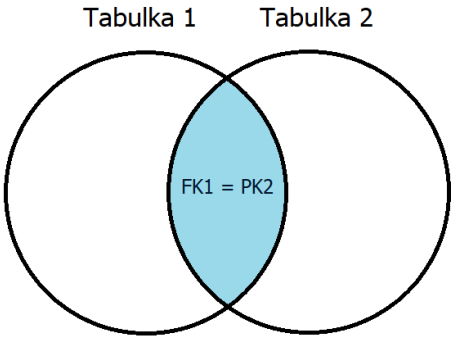


Obrázek 1.4 Vztah m:n (vlastní zpracování)

Každý záznam je identifikován pomocí jednoznačného identifikátoru, kterému se říká primární klíč. Pokud záznam spojujeme s další tabulkou, je potřeba jej identifikovat pomocí jeho identifikátoru, kterému se říká cizí klíč.

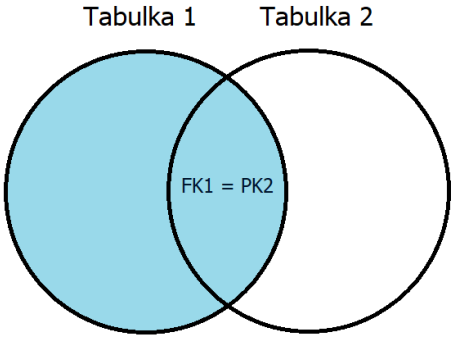
Při dotazu na data můžeme využít spojení tabulek právě díky těmto vztahům ke kterým slouží příkaz `JOIN`.

Inner join Jedná se o nejběžněji používaný `JOIN`. Při použití příkazu `SELECT` dostaneme jen ty záznamy, které jsou spolu ve vztahu přes určený cizí klíč. Na obrázku (Obr. 1.5) je znázorněný diagram výběru ze dvou tabulek.



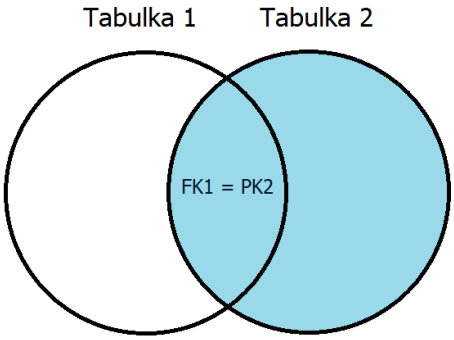
Obrázek 1.5 Inner join (vlastní zpracování)

Left join Pokud použijeme příkaz `SELECT` s `LEFT JOIN`, dostaneme všechny záznamy z první tabulky a z druhé tabulky jen záznamy, které jsou ve vztahu přes určený cizí klíč. Tento výběr zobrazuje i obrázek 1.6.



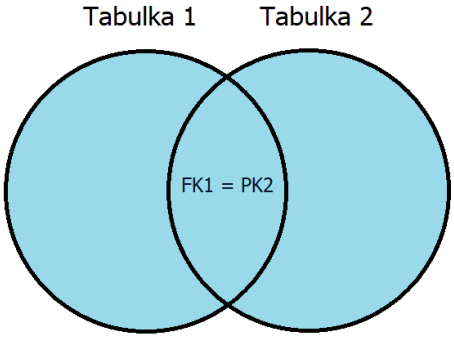
Obrázek 1.6 Left join (vlastní zpracování)

Right join Na obrázku 1.7 je zobrazen výsledek při použití příkazu `SELECT`, kdy dostaneme všechny záznamy z druhé tabulky a z první tabulky jen záznamy, které jsou ve vztahu přes určený cizí klíč.



Obrázek 1.7 Right join (vlastní zpracování)

Full outer join Při použití příkazu `SELECT` dostaneme všechny záznamy z obou tabulek. Toto je znázorněno na obrázku 1.8.



Obrázek 1.8 Full outer join (vlastní zpracování)

1.2 Frontend

Frontend je prezentační část aplikace, která je předváděna uživateli. Skripty jsou klientovi přístupné a může je na své straně měnit.

1.2.1 HTML

HyperText Markup Language je standard pro vytváření dokumentů zobrazovaných na WWW (*World Wide Web*). Byl vymyšlen zakladatelem WWW Tim Berners-Lee v roce 1990 [7]. Vycházel z *Standard Generalized Markup Language* SGML, který je určen k organizaci prvků dokumentu. Obsahuje následující části:

1. Deklarace – určuje které znaky a oddělovače se budou v dokumentu používat,
2. *Document Type Definition* DTD – popisuje syntax markupu,

3. Specifikace sémantik,

4. Obsah dokumentu [8].

HTML pomocí elementů popisuje strukturu stránky a způsob, jak zobrazit její obsah. Elementy jsou definovány pomocí takzvaných tagů. Element může existovat i bez použití tagu (například `<head>`) a také element může být také zanořený v dalším elementu. Element má svoje vlastnosti, které mohou být popsány atributy v tagu [9]. Názornou ukázku demonstruje příklad (Kód 1.1). Řádek 2 obsahuje úvodní tag párového elementu `html` a jeho koncový tag je na řádku 11. Příklad nepárového tagu, který neobsahuje koncový tag, je element `meta` na řádku 5, který obsahuje i atribut `charset`.

```
1   <!DOCTYPE html>
2   <html>
3     <head>
4       <title>Ukazkovy kod</title>
5       <meta charset="UTF-8" />
6     </head>
7     <body>
8       <h1>Ukazkovy kod HTML</h1>
9       <span>Toto je Ukazkovy kod pro bakalarskou praci.</span>
10    </body>
11  </html>
12
```

Kód 1.1 Příklad HTML kódu (vlastní zpracování)

1.2.2 CSS

Cascading Style Sheets jsou doplňkem pro HTML, který popisuje vzhled elementů stránky. CSS umožňuje rozšířit styl elementů podle jejich názvů, identifikátorů nebo tříd, které jsou definovány selektory. Dokáže určit defaultní styly během elementů i během událostí nebo po nich. Příklad kódu je znázorněn v kódu 1.2. Řádek 1-8 ukazuje definici stylu pro elementy `body` a `h1`, zatímco řádky 10-14 znázorňují přiřazení stylu pro všechny elementy ve třídě `product`.

```
1   body {
2     background-color: white;
3   }
4
5   h1 {
6     color: blue;
7     text-align: center;
8   }
9
10  .product {
11    color: red;
12    margin-left: 10px;
13  }
14
```

Kód 1.2 Příklad CSS kódu (vlastní zpracování)

1.2.3 JavaScript

JavaScript je skriptovací jazyk, který se používá k dynamickým změnám elementů dokumentu, definovaného HTML. Zatímco CSS se věnuje designu jednotlivých elementů, JavaScript dokáže kromě toho definovat také jejich chování, či elementy přidávat nebo mazat. Kód 1.3 ukazuje příklad základní syntaxe. Řádek 1 přiděluje proměnné `result` element z daného HTML dokumentu, který je definován ID `result`. Na dalším řádku je pak volána funkce `changeColor()`, která je definována na řádku 4-6. Tato funkce mění barvu textu na černou.

```
1     let result = document.getElementById('result');
2     changeColor(result);
3
4     function changeColor(element) {
5       element.style.color = 'black';
6     }
7
8
```

Kód 1.3 Příklad JavaScript kódu (vlastní zpracování)

2 LARAVEL FRAMEWORK

Laravel je open-source framework pro webové aplikace, který využívá programovací jazyk PHP. Založen byl v roce 2011 Taylorem Otwellem jako reakce na PHP framework CodeIgniter z roku 2006, který postrádal funkce pro autentizace a autorizaci uživatele [10].

2.1 Návrhové vzory

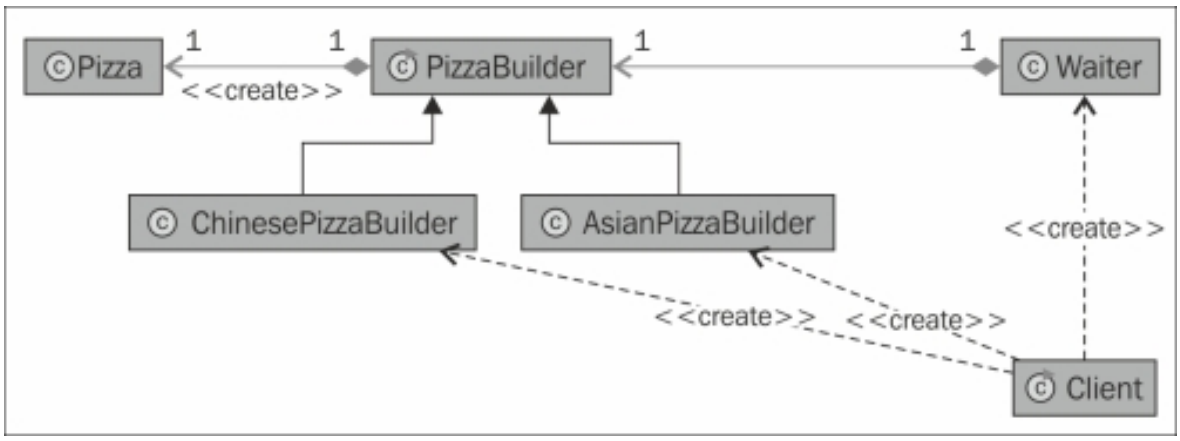
Laravel přinesl jednoduchost a čitelnost díky podpoře několika následujících návrhových vzorů.

2.1.1 Builder pattern

Builder pattern je návrhový vzor, který se zabývá vytvářením objektů v objektově orientovaném programování. Pomáhá vybudovat komplexnější objekty složené z jednoduchých objektů a je nezávislý na dalších objektech. Funguje na principu vytvoření abstraktní třídy `Builder`, která určuje základní vlastnosti objektu.

Laravel používá tento vzor u třídy `AuthManager`, která je potomkem třídy `Manager`, pro přihlašování uživatele. Narozdíl od rodiče však tato třída potřebuje manipulovat se *sessions* a *cookies*, ve kterých bývají uložené autentizační tokeny [11].

Diagram (Obr. 2.1) ukazuje princip Builder pattern. Třída `PizzaBuilder` je abstraktní třídou, která definuje vytváření objektu `Pizza`. Její potomci pak přímo specifikují parametry objektu tak, aby odpovídal jejich vlastnostem.



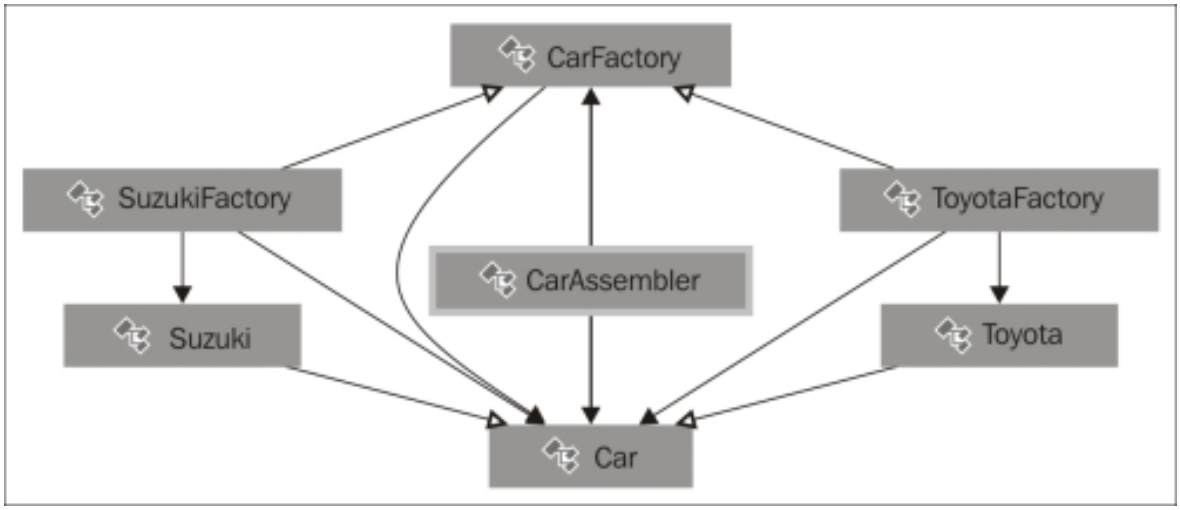
Obrázek 2.1 Builder pattern [11]

2.1.2 Factory pattern

Factory pattern slouží k vytváření šablonových metod, které jsou založené na rodičovské třídě. Umožňuje vytvořit instanci objektu bez explicitního definování jeho třídy.

Tento návrhový vzor je v Laravelu využíván v třídě `Validation` pro psaní vlastních validačních pravidel [12].

Na obrázku 2.2 je znázorněn diagram demonstrující funkci *Factory pattern*. Každý potomek třídy `CarFactory` vytváří potomka třídy `Car`.



Obrázek 2.2 Factory pattern [12]

2.1.3 Repository pattern

Repository pattern umožňuje abstrakci datové vrstvy a centralizování používání objektů.

Repository pattern je vhodná pro použití při práci s Eloquent modely. Používá se vytvořením třídy repozitářového rozhraní, které definuje metody pro používání modelu. Implementace tohoto *interface* způsobí lepší zacházení s kontrolerem, který díky tomu nemusí vědět jakým způsobem s daty manipuluje [13].

Na následujícím příkladu kód 2.1 ukazuje vytvoření rozhraní pro model `User`.

```
<?php namespace App\Repository\User;

interface UserRepository {

  public function all();

  public function get();

  public function create($input);

  public function update($input);

  public function delete($input);

  public function find($id);

}
```

Kód 2.1 Vytvoření repository pattern [13]

Třída v kódu 2.2 implementuje třídu `UserRepository` pro manipulaci pomocí Eloquent.

```
<?php namespace App\Repository\User;

use User;

class EloquentUserRepository implements UserRepository {

  public function all()
  {
    return User::all();
  }

  public function get()
  {
    return User::get();
  }

  public function create($input)
  {
    return User::create($input);
  }

  public function update($input)
  {
    return User::update($input);
  }

  public function delete($input)
  {
    return User::delete($input);
  }

  public function find($input)
  {
    return User::find($input);
  }

}
```

Kód 2.2 Implementace rozhraní [13]

2.1.4 Strategy pattern

Strategy pattern řeší problémy výběru vhodného algoritmu za běhu programu. To se řeší pomocí interface, který definuje funkci a třídy jednotlivých strategií ji potom implementují jejich zvoleným algoritmem.

V Laravelu se tohoto vzoru využívá u implementace rozhraní `LoaderInterface`, kde jsou zvoleny metody podle nastavení v souboru `appconfigapp.php` [14].

2.1.5 Provider pattern

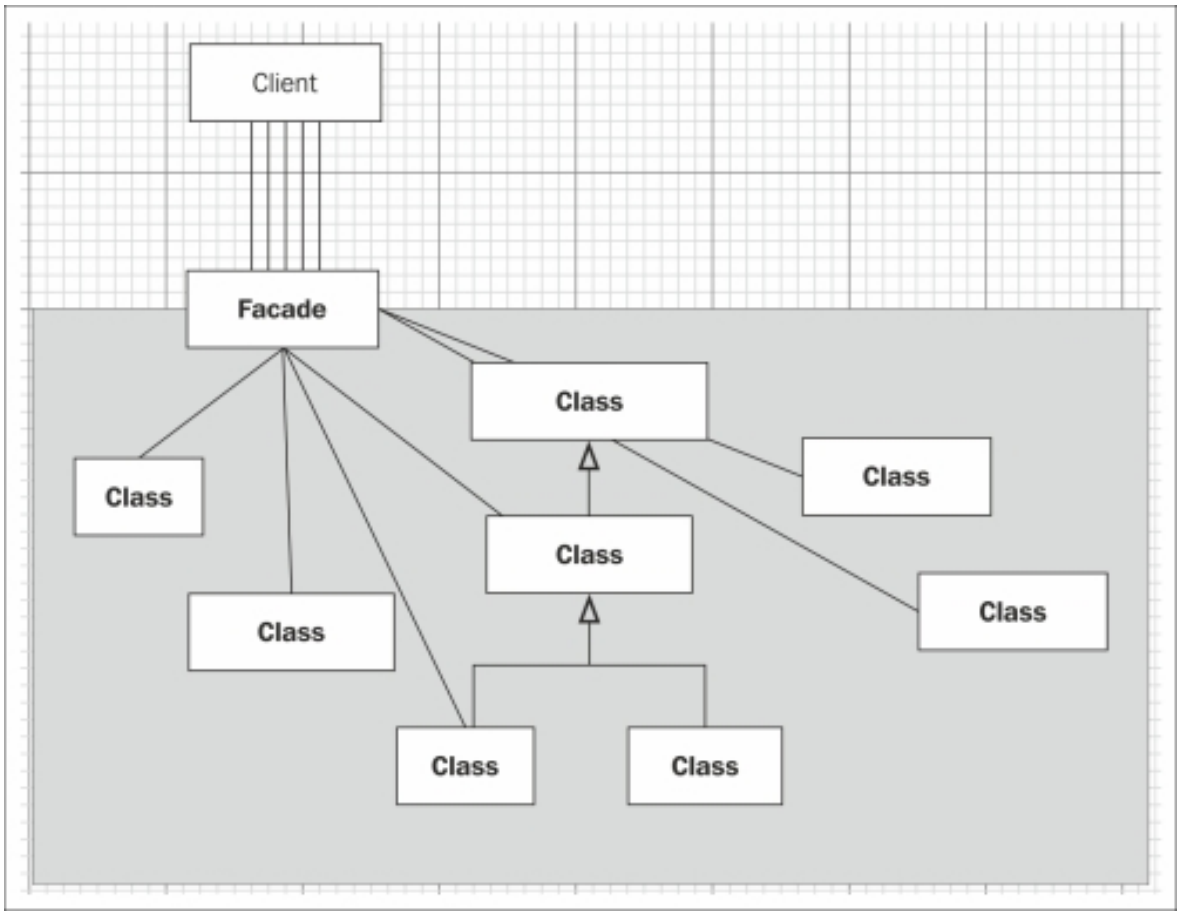
Provider pattern byla představena Microsoftem v *ASP.NET Starter Kits*. Funguje na stejném principu jako *Strategy pattern*, ale k implementaci používá abstraktní třídu a slouží konkrétně jako vrstva mezi třídou API a samotnou abstrakcí dat.

Provider pattern se v tomto frameworku využívá u třídy `AuthServiceProvider` a `HashServiceProvider`, které jsou potomky `ServiceProvider`. `AuthServiceProvider` umožňuje zvolit si mezi zasláním session nebo cookie přes response nebo přes třídu

`AuthDriver`, zatímco `HashServiceProvider` rozhoduje, jakou operaci provést s hashi zaslanými requestem [15].

2.1.6 Facade pattern

Facade pattern umožňuje sjednotit několik komplikovaných rozhraní do jedné třídy. Schovává tak komplexnost systému, jak je znázorněno na obrázku 2.3.



Obrázek 2.3 Facade pattern [16]

Laravel Facade pattern využívá například ve třídě `URL` [16].

2.2 Architektura

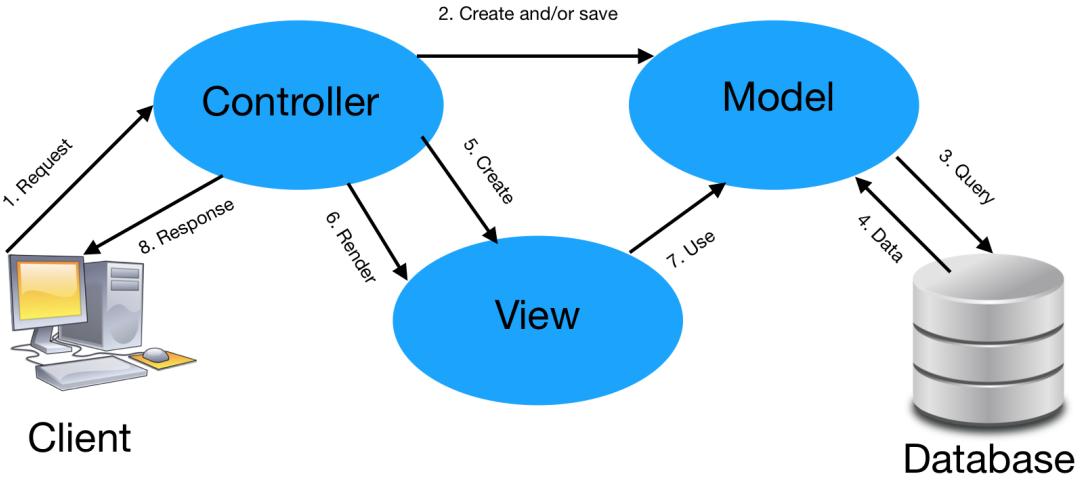
Laravel používá takzvanou MVC (*Model-View-Controller*) architekturu, která rozděluje aplikaci do tří základních vrstev. Tyto vrstvy jsou vázány výhradně na svou sousední vrstvu – v tomto případě vrstva *Model* a *View* spolu komunikují přes vrstvu kontroler, jak je znázorněno na obrázku 2.4.

View představuje klientskou část. Pomocí uživatelského rozhraní zobrazuje klientovi odpovědi na jeho dotazy. *View* je definován HTML kódem, který definuje obsah zobrazované stránky.

Přes *routes* je z *View* poslán do kontroleru požadavek (*request*). Routes určují, jaký kontroler bude mít dotaz na starosti, a která metoda jej obslouží.

Kontroler rozhoduje, jakou odpověď zašle uživateli. V případě, že je požadavek přijat, kontroler vyžádá potřebná data od modelu, která následně zpracuje a odešle v odpovědi zpět do View.

Model obsahuje uložené informace aplikace. Tato data odesílá kontroleru podle jeho požadavku.



Obrázek 2.4 Model-View-Controller architektura [17]

2.3 Součásti

Laravel je doplňován nebo rozšířen dalšími balíčky a nástroji, které rozšiřují jeho funkce.

2.3.1 Composer

Composer slouží pro správu závislostí v PHP. Instaluje a aktualizuje knihovny a balíčky pro daný projekt.

2.3.2 Artisan

Artisan je *Command Line Interface*, který používá Laravel. Pomocí příkazu začínajícím `php artisan` lze generovat soubory a pracovat s frameworkem [18].

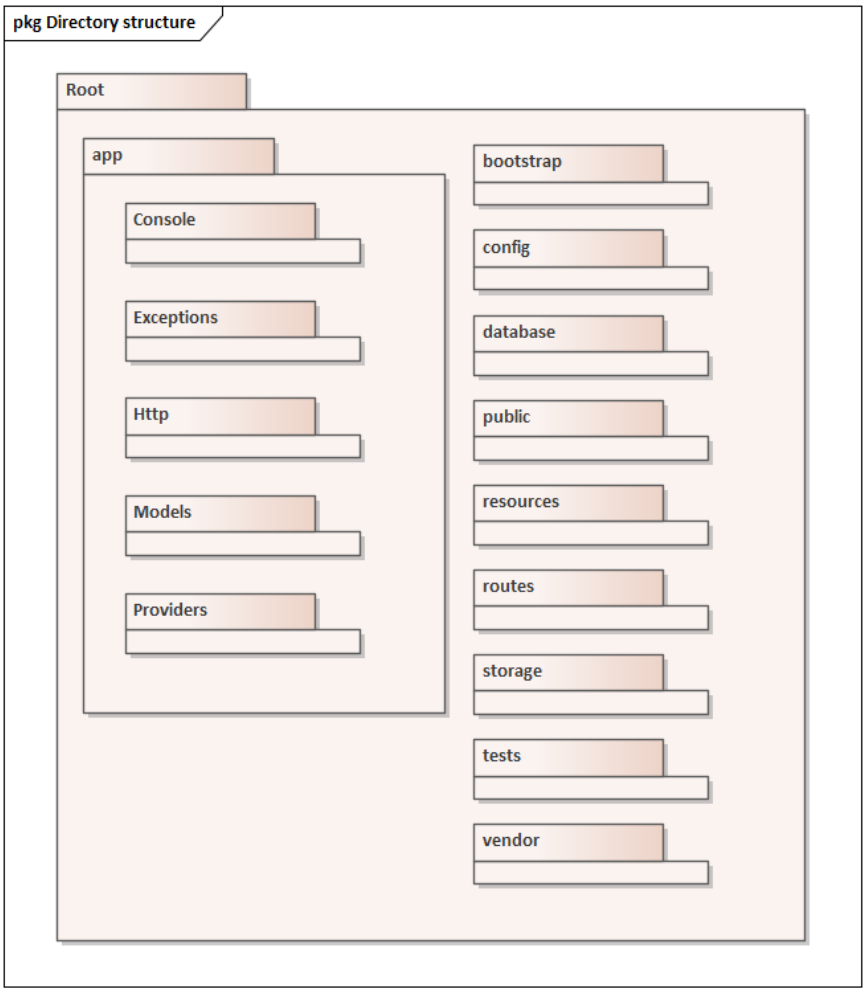
2.3.3 Webpack

Webpack je takzvaný *module bundler* primárně určený pro JavaScript. To znamená, že vypracovává graf závislostí, který mapuje moduly a vytváří kolekce programů [19].

Jako konfigurační vrstva pro Webpack slouží Mix, odkud jsou kompilovány JavaScript a CSS soubory [20].

2.4 Základní struktura

Základní adresářová struktura Laravelu poskytuje strukturu pro začátek jakékoliv aplikace. Kořenový adresář je rozdělen do několika podadresářů (Obr. 2.5), které mají své funkce. Podle této struktury bude v následujících kapitolách rozebrán frameworku Laravel.



Obrázek 2.5 Základní struktura adresáře (vlastní zpracování)

Samotný kořenový adresář obsahuje soubory potřebné k základní konfiguraci, jako jsou `.env`, soubory verzovacího systému GIT. Zbylý obsah se pro větší přehlednost řadí do podadresářů.

2.4.1 App

`app` patří mezi nejdůležitější adresáře, který obsahuje hlavní kód naší aplikace. Podrobněji je popsán v kapitole 2.5.

2.4.2 Bootstrap

Adresář `bootstrap` obsahuje soubory, které slouží ke spuštění frameworku – tím je především `app.php`. Tento adresář je nastaven tak, aby nebyla nutná manipulace s jeho informacemi, která by mohla vést k poškození funkčnosti webové aplikace [21].

2.4.3 Config

Adresář `config` obsahuje konfigurační soubory aplikace.

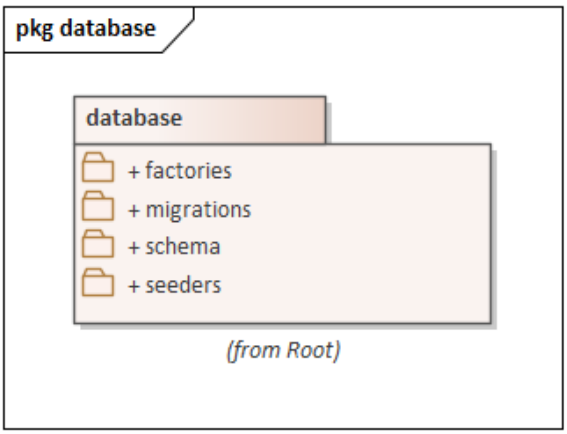
2.4.4 Database

Adresář `database` obsahuje všechny soubory spojené databází.

Laravel podporuje propojení s následujícími databázemi:

- MySQL od verze 5.7,
- PostgreSQL od verze 9.6,
- SQLite od verze 3.8.8,
- SQL Server od verze 2017 [22].

Struktura adresáře na obrázku 2.6 bude rozebrána v následujících kapitolách.



Obrázek 2.6 Adresář database (vlastní zpracování)

Factories `Factory` umožňuje automatické vytváření dat pro testování databáze. To zjednodušuje práci o manuální vkládání dat. V následujícím příkladu (Kód 2.3) je znázorněno vytvoření `UserFactory`, které umožní naplnění tabulky `users` generovanými daty.

```
namespace Database\Factories;

use App\Models\User;
use Carbon\Carbon;
use Illuminate\Database\Eloquent\Factories\Factory;
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Str;

class UserFactory extends Factory
{
  protected $model = User::class;

  public function definition()
  {
    return [
    'name' => $this->faker->name(),
    'email' => $this->faker->unique()->safeEmail(),
    'email_verified_at' => Carbon::now(),
    'password' => Hash::make('Safe password'),
    'remember_token' => Str::random(10),
    ];
  }
}

```

Kód 2.3 Factory [23]

Migrations K vytvoření schématu slouží migrace, které definují jeho tabulky. Tento postup je výhodný pro týmové práce, protože udržují databázi každého uživatele aktuální. Migrace se skládá ze dvou metod `up()`, která slouží pro vytvoření migrace a `down()`, která migraci zruší. Základem těchto metod je tedy jejich protichůdnost, kdy se mají navzájem vyrušit bez ovlivnění zbylých migrací.

Migrace se vytváří pomocí příkazu v artisanu

```
projectDirectory> php artisan make migration create_flights_table
```

Tento příkaz vytvoří třídu, která je potomkem třídy `Migration`. Implementuje již zmíněné metody `up()` a `down()`. Příklad této metody je ukázán v kódu 2.4.

```php
<?php

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

class CreateFlightsTable extends Migration
{
  public function up()
  {
    Schema::create('flights', function (Blueprint $table) {
      $table->id();
      $table->string('name');
      $table->string('airline');
      $table->timestamps();
    });
  }

  public function down()
  {
    Schema::drop('flights');
  }
}

```

Kód 2.4 Migrace [24]

Jakmile jsou migrace definovány, spouští se v artisanu pomocí příkazu

```
projectDirectory> php artisan migrate
```

Pro zrušení posledních migrací slouží příkaz

```
projectDirectory> php artisan migrate:rollback --step=5
```

kde `--step` je nepovinný parametr definující počet kroků k navrácení.

Pokud chceme zrušit všechny migrace, je vhodné použít

```
projectDirectory> php artisan migrate:reset
```

Pro znovuobnovení všech migrací se používá

```
projectDirectory> php artisan migrate:refresh --seed
```

kde `--seed` je nepovinný parametr, který po migracích spustí i seeding databáze.

Seeders Pro naplnění databáze daty slouží seedery. Ty se vytvoří pomocí

```
projectDirectory> php artisan make:seeder UserSeeder
```

Tento příkaz vytvoří potomka třídy `Seeder`, která implementuje metodu `run()`, díky které se naplní databáze. Jako příklad je uveden kód 2.5.

```
<?php

namespace Database\Seeders;

use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Str;

class DatabaseSeeder extends Seeder
{
  public function run()
  {
    DB::table('users')->insert([
    'name' => Str::random(10),
    'email' => Str::random(10).'@gmail.com',
    'password' => Hash::make('password'),
    ]);
  }
}

```

Kód 2.5 Seeder [25]

Spuštění seederu zajistí příkaz

```
projectDirectory> php artisan db:seed
```

který defaultně spustí `DatabaseSeeder.php`, ve kterém jsou volány ostatní seedery.

V případě, že je potřeba spustit pouze jeden seeder, využívá se příkazu

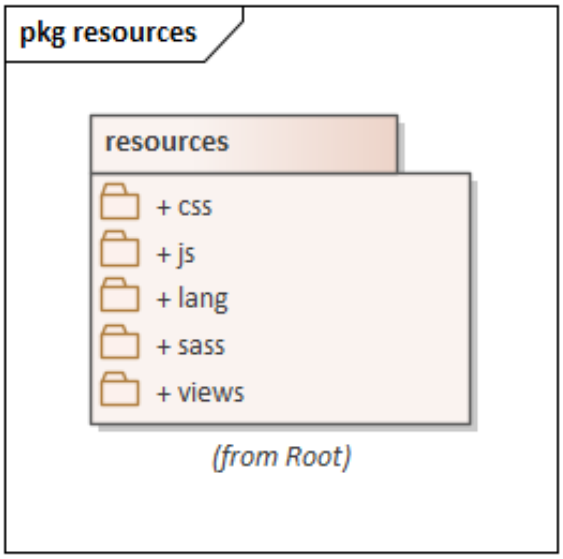
```
projectDirectory> php artisan db:seed --class=JmenoSeederu
```

2.4.5 Public

Adresář `public` obsahuje `index.php`, který slouží jako vstupní bod pro všechny požadavky do aplikace, a dále kompilovaný JavaScript a CSS a obrázky.

2.4.6 Resources

Adresář `resources` obsahuje všechny views, soubory pro lokalizaci a nekompilovaný JavaScript a CSS. Strukturu adresáře popisuje obrázek 2.7.



Obrázek 2.7 Adresář resources (vlastní zpracování)

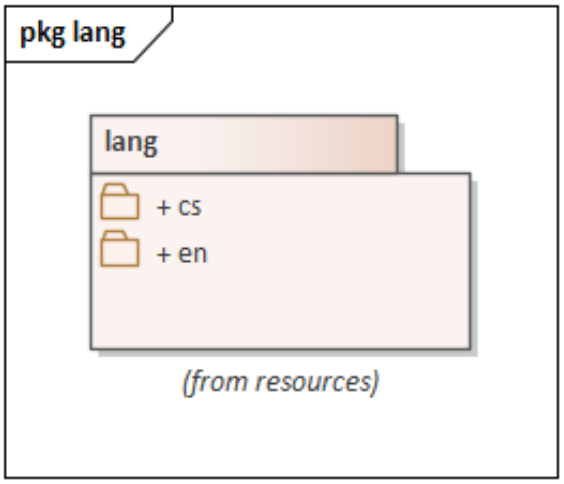
Views View je soubor, který obsahuje HTML kód pro zobrazení obsahu uživateli. Laravel k tomuto používá soubory `.blade.php`, které obsahují takzvanou Blade template. Ta nabízí efektivní řešení, jak propojit HTML s PHP kódem. Díky tomu můžeme předávat datový obsah z kontroleru do view a zobrazit ho bez vkládání čistého PHP kódu do dokumentu. Tomu dopomáhají directivy (Kód 2.6), které umožňují jednoduše definovat části dokumentu, které jsou závislé na obsahu.

```
<span>
  @if (count($records) === 1)
    I have one record!
    @elseif (count($records) > 1)
      I have multiple records!
  @else
    I don't have any records!
  @endif
</span>

@foreach ($users as $user)
  <p>This is user {{ $user->id }}</p>
@endforeach
```

Kód 2.6 Direktivy [26]

Lokalizace Lokalizace umožňuje přizpůsobit obsah dokumentů jazyku uživatele. Laravel tento text získává buď ze souborů `.php` a nebo `.json`. Struktura adresáře je popsána na obrázku 2.8.



Obrázek 2.8 Adresář lang (vlastní zpracování)

PHP soubory jsou uloženy v podadresářích podle kódu jazyka. V tomto případě je text definován názvem souboru a klíčovým slovem, jak je uvedeno v kódu 2.8. Záznam překladu je definován v poli jako je to v příkladu 2.7.

```
<?php

return [
  'welcome' => 'Welcome to our application!',
];

```

Kód 2.7 Lokalizace v php souboru [27]

```
<span>{{ __('messages.welcome') }}</span>

```

Kód 2.8 Příklad volání překladu v blade [27]

V případě použití JSON souboru je tento soubor uložen přímo v adresáři lang a pojmenován podle kódu jazyka. Text je pak převáděn podle parametru volané funkce (Kód 2.10) jako klíče v JSON souboru (Kód 2.9). V případě, že takovýto klíč nenajde, je ponechán text zadaný v parametru.

```
{
  "I love programming.": "Miluji programovani."
}

```

Kód 2.9 Příklad lokalizace v JSON souboru[27]

```
<span>{{ __('I love programming.') }}</span>
```

Kód 2.10 Příklad volání JSON překladu v bladu [27]

2.4.7 Routes

`routes` slouží k definování cest v naší aplikaci. Laravel je defaultně rozděluje na routy webové `web.php` a na routes API `api.php`. Routes přiřazují URI jejich metody nebo kontrolery, které mají na starosti vyřízení příchozího požadavku. Dostupné metody routeru jsou znázorněny v kódu 2.11.

```
Route::get($uri, $callback);
Route::post($uri, $callback);
Route::put($uri, $callback);
Route::patch($uri, $callback);
Route::delete($uri, $callback);
Route::options($uri, $callback);
```

Kód 2.11 Metody routeru [28]

Pokud má route parametr, do URI je vložen jeho název ve složených závorkách a v případě volitelného parametru se za název přidá otazník. Příklad je znázorněn v kódu 2.12.

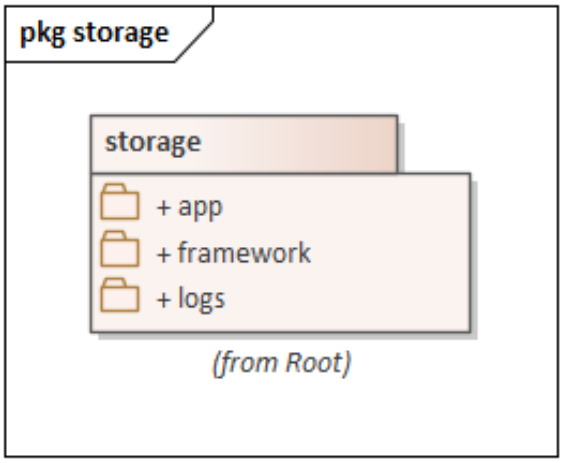
```
Route::get('/user/{id}', function ($id) {
  return 'User '.$id;
});

Route::get('/user/{name?}', function ($name = 'John') {
  return $name;
});
```

Kód 2.12 URI s parametrem [28]

2.4.8 Storage

Adresář `storage` obsahuje logovací soubory a kompilované Blade šablony a další soubory generované frameworkem. Podadresář app slouží k uchování souborů generovaných aplikací, framework je pro cache a soubory generované frameworkem a logs obsahuje logy aplikace. Strukturu adresáře popisuje obrázek 2.9.

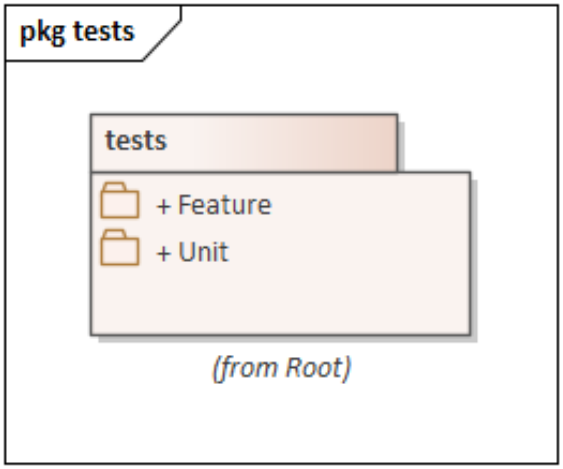


Obrázek 2.9 Adresář storage (vlastní zpracování)

2.4.9 Tests

Adresář `tests` slouží pro automatizované testy. Patří sem unit testy PHPUnit a feature testy. Pro spuštění testů slouží

`projectDirectory> php artisan test` . Struktura adresáře je znázorněna na obrázku 2.10.



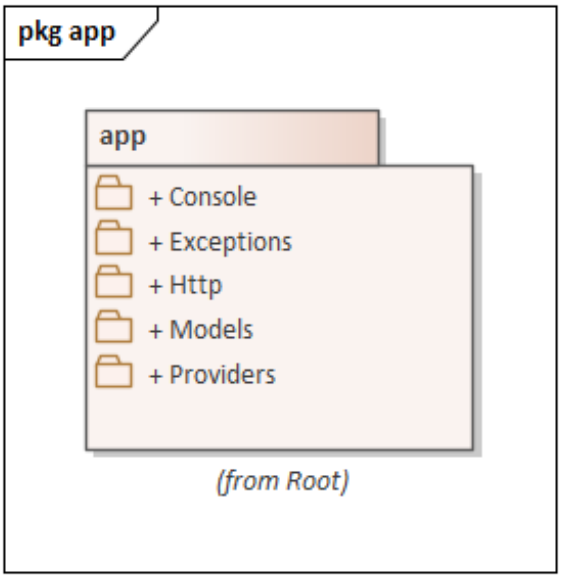
Obrázek 2.10 Adresář tests (vlastní zpracování)

2.4.10 Vendor

Adresář `vendor` obsahuje všechny závislosti generované composerem.

2.5 App

Adresář `app` obsahuje hlavní kódy aplikace. Jeho struktura je popsána na obrázku 2.11 a v následujících podkapitolách.



Obrázek 2.11 Adresář app (vlastní zpracování)

2.5.1 Console

Adresář `Console` obsahuje všechny Artisan příkazy aplikace. Ty mohou být generovány pomocí

```
projectDirectory> php artisan make:command
```

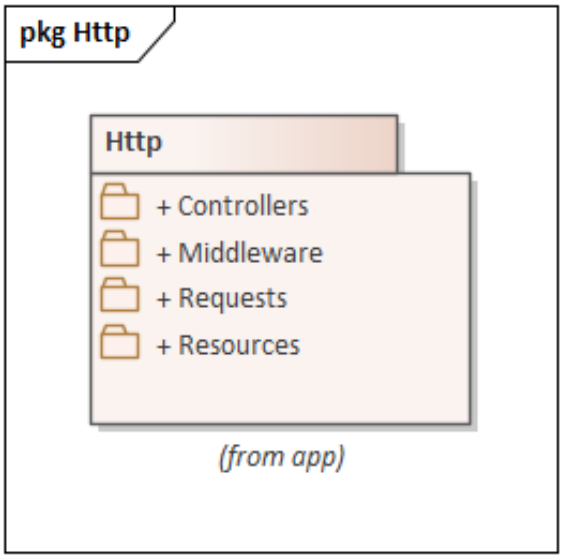
Dále obsahuje kernel konzole, kde jsou zapisovány tyto příkazy a definovány plánované úlohy.

2.5.2 Exceptions

Adresář `Exceptions` obsahuje *exception handler* aplikace, který určuje, jak jsou výjimky zobrazovány a renderovány. Dále také může obsahovat vlastní výjimky pro aplikaci.

2.5.3 Http

Adresář `http` je hlavní mozek aplikace, který zpracovává všechny požadavky a odpovědi. Jeho struktura je popsána na obrázku 2.12.



Obrázek 2.12 Struktura adresáře http
(vlastní zpracování)

2.5.4 Controllers

Třída kontroleru obsahuje akční metody pro ovládání routes. Vytvoří se pomocí příkazu

```
projectDirectory> php artisan make:controller UserController
```

V případě toho, že chceme vygenerovat všechny akční metody pro práci s modelem, použijeme příkaz

```
projectDirectory> php artisan make:controller UserController --resource
```

ten vygeneruje všechny potřebné cesty ke kontroleru dle následující tabulky (Tab. 2.1).

Tabulka 2.1 Akce a cesty ke kontroleru

Metoda	URI	Akce	Název cesty
GET	`/users`	`index`	`users.index`
GET	`/users/create`	`create`	`users.create`
POST	`/users`	`store`	`users.store`
GET	`/users/{id}`	`show`	`users.show`
GET	`/users/{id}/edit`	`edit`	`users.edit`
PUT/PATCH	`/users/{id}`	`update`	`users.update`
DELETE	`/users/{id}`	`destroy`	`users.destroy`

2.5.5 Middleware

Controller slouží k vyřízení požadavků klienta a předání odpovědi od modelu. Pomáhá s prověřením requestu formou autentizace, autorizace, atp.

Middleware je vytvořen pomocí příkazu

```
projectDirectory> php artisan make:middleware EnsureTokenIsValid
```

který vytvoří danou třídu.

Příklad middlewaru je znázorněn v kódu 2.13.

```
<?php

namespace App\Http\Middleware;

use Closure;

class EnsureTokenIsValid
{
  public function handle($request, Closure $next)
  {
    if ($request->input('token') !== 'my-secret-token') {
      return redirect('home');
    }

    return $next($request);
  }
}

```

Kód 2.13 Middleware [30]

Třídu je potom potřeba registrovat v `app/Http/Kernel.php`. V případě přidělení klíče do pole `$middleware`, je tato třída brána jako globální a tedy je spuštěna při každém HTTP požadavku. Pokud chceme třídu přiřadit pouze určitým cestám, vytvoříme nový index v poli `$routeMiddleware` a následně definovaný klíč použijeme při definování cesty v adresáři `routes` pomocí funkce `middleware('nazevKlice')` [30].

2.5.6 Requests

Request je příchozí požadavek od klienta, který je zpracováván kontrolerem. V Laravelu je pro požadavky přímo vytvořená třída `Request`, která je injektována do akční metody přes takzvaný service container. Ve třídě je kromě původní adresy a hlavičky požadavku uložen také input, soubory a cookies [31].

Požadavek přichází z webového serveru do souboru `public/index.php`, pak je poslán do `app/Http/Kernel.php`. První je spuštěno pole s `bootstrappers`, které konfigurují errory, logy a detekují prostředí aplikace. Ty navíc spustí service providers, které jsou konfigurovány v poli `providers` souboru `config/app.php`. Každý z nich je zavolán metodou `register` a po jejich registraci jsou volány jejich metody `boot`. Service providers mají na starosti načítání komponent frameworku tj. databáze, fronta,

validace a routes. Potom je tedy požadavek předán routeru, který jej odešle do kontroleru (nebo případně route) přes určený middleware. Jakmile je požadavek zpracován, jeho odpověď je poslána do `index.php`, který zavolá metodu `send` a pošle obsah odpovědi do webového prohlížeče klienta [32].

2.5.7 Models

Adresář `Models` obsahuje třídy Eloquent modelů. Eloquent slouží k objektovému mapování tabulek na modely, čímž usnadňují manipulaci s databází. Každá tabulka má tedy svoji třídu `Model`, díky které získáme záznamy databáze.

Modelová třída se vytvoří příkazem:

```
projectDirectory> php artisan make:model User
```

Ta vytvoří soubor s třídou v adresáři `app/Models`, jak je ukázáno v kódu 2.14.

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Flight extends Model
{
  protected $table = 'my_flights';
}

```

Kód 2.14 Vytvoření modelové třídy [33]

Po definici modelu, lze získat data z databáze voláním třídy metodami query builder, který poskytuje rozhraní pro vytvoření databázových příkazů. V kódu 2.15 je zobrazen příklad získání dat z tabulky `users`.

```
<?php

namespace App\Http\Controllers;

use App\Http\Controllers\Controller;
use Illuminate\Support\Facades\DB;

class UserController extends Controller
{

  public function index()
  {
    $users = DB::table('users')->get();

    return view('user.index', ['users' => $users]);
  }
}

```

Kód 2.15 Získání dat z databáze [34]

2.5.8 Providers

Adresář `Providers` obsahuje service providers, které aplikaci připravují na příchozí požadavky, které byly popsány v kapitole 2.5.6.

Provider je vytvořen příkazem

```
projectDirectory> php artisan make:provider RiakServiceProvider
```

Jak již bylo zmíněno, tak provider musí definovat metodu `register` (Kód 2.16).

```
<?php

namespace App\Providers;

use App\Services\Riak\Connection;
use Illuminate\Support\ServiceProvider;

class RiakServiceProvider extends ServiceProvider
{
  public function register()
  {
    $this->app->singleton(Connection::class, function ($app) {
      return new Connection(config('riak'));
    });
  }
}

```

Kód 2.16 Metoda register v provideru [35]

Provider musí být registrován v `config/app.php` (Kód 2.17).

```
'providers' => [
// Other Service Providers

App\Providers\ComposerServiceProvider::class,
],

```

Kód 2.17 Registrace provideru [35]

2.6 Zabezpečení

Laravel poskytuje jednoduché řešení pro zamezení hrozeb v aplikaci.

2.6.1 Autentizace

Autentizace je prováděna pomocí třídy `Auth`, která definuje přihlašování a odhlašování uživatele a kontroluje data uložená defaultně v tabulce `users`.

Je tvořená takzvanými guards a providers. Guard určuje, jak je uživatelský request autentizován, zatímco provider definuje, jakým způsobem jsou data uživatele získána.

Konfigurace autentizace se nachází v `config/auth.php` [36].

2.6.2 Autorizace

Autorizace kontroluje při requestu, zda byl požadavek oprávněný. Pro definování autorizace slouží gates a policies. Gate vytváří cestu k autorizaci a policy určuje její logiku [37].

2.6.3 Ochrana CSRF

Cross-site request forgery je útok neautorizovaným požadavkem skrz autentizovaným uživatelem. Tomuto zabraňuje takzvaný CSRF token, který se generuje pro každého aktivního uživatele. Ten se vkládá přímo do View pomocí direktivy `@csrf` vložené dovnitř formuláře [38].

2.6.4 Hashování

Pro bezpečné hashování Laravel používá hashe Bcrypt a Argon2. Jejich konfigurace je nastavena v souboru `config/hashing.php`. Pro vytvoření hashe slouží funkce `make()` a pro jeho kontrolu `check($string1, $string2)` [39].

2.6.5 Validace

Request má metodu validace, která zkontroluje příchozí data a také autorizuje požadavek a rozhodne, zda je zpracován nebo vrácen klientovi. Třída `Request` obsahuje metodu `validate`, která jako parametr přijímá pole validačních pravidel pro příchozí atributy formuláře, které jsou určeny klíči hodnot. Příklad takovéto metody je znázorněn v kódu 2.18.

```
1     public function store(Request $request)
2     {
3       $validated = $request->validate([
4       'title' => 'required|unique:posts|max:255',
5       'body' => 'required',
6       ]);
7
8       // The blog post is valid...
9     }
10
```

Kód 2.18 Validace požadavku [40]

Další možností je vytvořit potomka třídy `Request`

```
projectDirectory> php artisan make:request StorePostRequest
```

a definovat jeho validaci pomocí metody `rules()` jako je uvedeno v kódu 2.19.

```
1     public function rules()
2     {
3       return [
4       'title' => 'required|unique:posts|max:255',
5       'body' => 'required',
6       ];
7     }
8
```

Kód 2.19 Použití metody rules() [40]

V tomto případě získáme validovaná data v poli pomocí `$request->validated()`.

U obou způsobů se v případě nevalidního vstupu akční metoda nedokončí a je navrácena response s chybovými hláškami.

2.6.6 SQL injections

SQL injections je způsob napadení, kde je do inputu vložena sql query, která může získat případně i upravit data v databázi.

Eloquent i Query builder Laravelu jsou zabezpečeny proti SQL injections s výjimkou názvů sloupců tabulky (například při použití `orderBy($column)` s uživatelským vstupem) a také metody `raw()`. V takovém případě je nutné použít validaci [33].

3 VUE.JS FRAMEWORK

Vue.js je framework založen na vytváření komponent `.vue` pro uživatelské rozhraní které jsou vkládány do HTML kódu a mohou být rozšířené o funkce. Kód frameworku je složen z HTML kódu, který je definován v elementu `<template>` a potom definicí komponenty, která je v `<script>` a je psána pomocí JavaScriptu.

Tělo aplikace je definováno pomocí následujícího elementu (Kód 3.1).

```
<div id="counter">
Counter: {{ counter }}
</div>

```

Kód 3.1 Vytovření Vue HTML [41]

Inicializace Vue probíhá pomocí kódu 3.2.

```
const Counter = {
  data() {
    return {
      counter: 0
    }
  }
}

Vue.createApp(Counter).mount('#counter')

```

Kód 3.2 Inicializace Vue [41]

Pro definici chování elementů jsou v kódu používány direktivy, jejichž příklady jsou uvedeny v kódu 3.3.

```
<template>
<ol>
<li v-for="numer in list">{{ number }}</li>
<li v-if="false">I am not created</li>
<li v-show="false">I am visible</li>
</ol>
<new-component></new-component>

</template>

<script>
import NewComponent from './NewComponent'

export default {
  components: {
    NewComponent
  }
  data() {
    return {
      list: ["one", "two", "three"]

    }
  }
}
</script>

```

Kód 3.3 Použití komponenty [41]

4 TAILWIND CSS FRAMEWORK

Tailwind je framework, který pomocí tříd definuje vzhled elementů. Na rozdíl od frameworku Bootstrap neurčuje celý vzhled komponent, ale využívá více menších tříd dohromady a vytváří tak originálnější design.

Framework se instaluje pomocí nástroje `npm`.

```
projectDirectory> npm install -D tailwindcss@latest postcss@latest
autoprefixer@latest
```

Poté se nastaví jeho konfigurační soubor `tailwind.config.js`

```
projectDirectory> npx tailwindcss init
```

Který vypadá následovně (Kód 4.1).

```
1     module.exports = {
2       purge: [],
3       darkMode: false, // or 'media' or 'class'
4       theme: {
5         extend: {},
6       },
7       variants: {},
8       plugins: [],
9     }
10
```

Kód 4.1 Konfigurační soubor Tailwind [42]

V hlavním `.css` souboru aplikace je potom definován kód 4.2.

```
1     @tailwind base;
2     @tailwind components;
3     @tailwind utilities;
4
```

Kód 4.2 Inicializace Tailwind v CSS [42]

A v hlavním `.js` souboru se přidá napojení tříd v kódu 4.3.

```
1     import "tailwindcss/tailwind.css"
2
```

Kód 4.3 Inicializace Tailwind v JS [42]

Použití frameworku demonstruje jednoduchý příklad (Kód 4.4), na kterém je nastavený text na tmavě červenou barvu s větším písmem odsazeným vlevo a při ukázáním myší ztmavne.

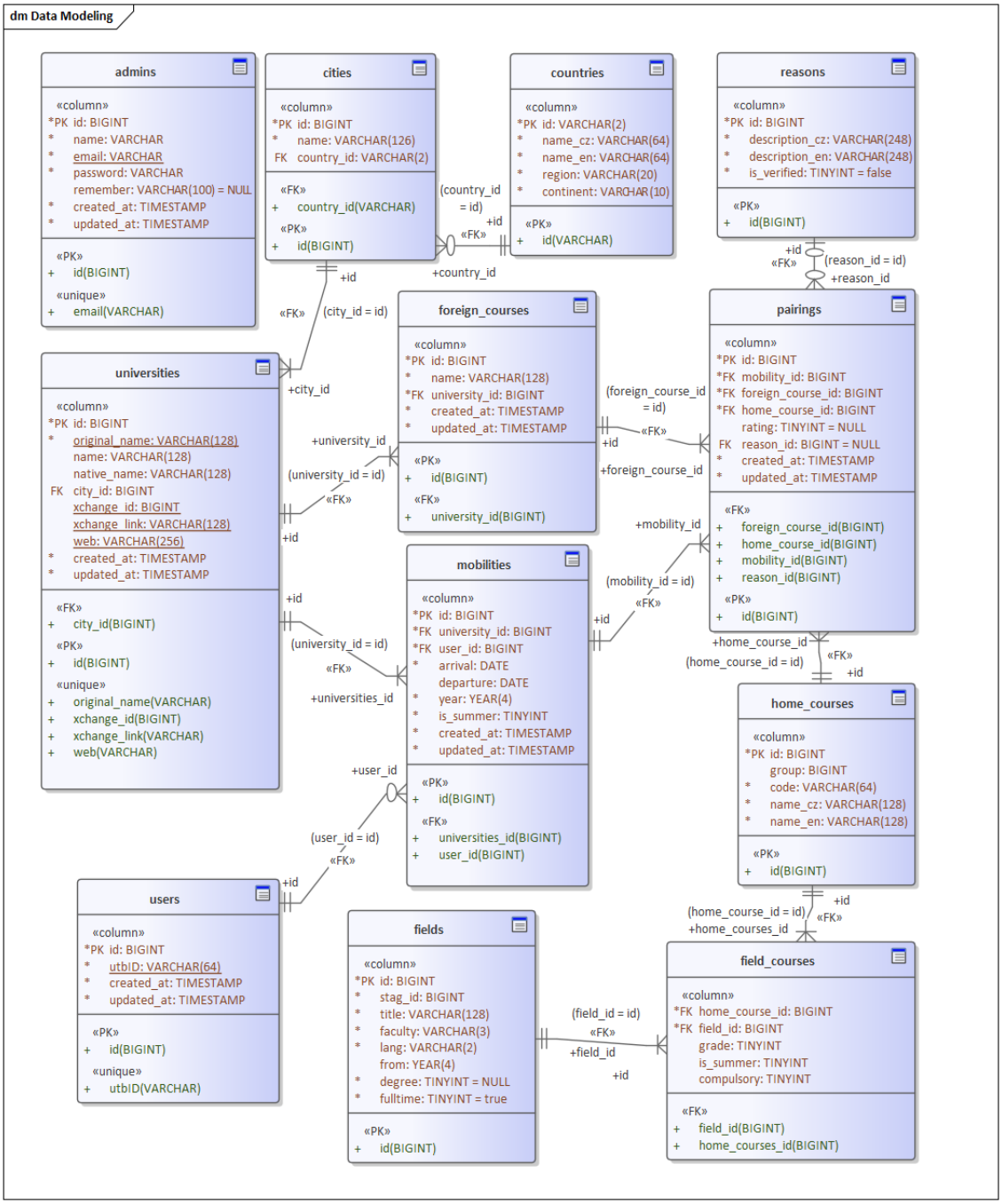
```
<p class="text-left text-red-700 hover:text-red-900 text-xl">
Lorem ipsum dolor sit amet ...
</p>
```

Kód 4.4 Příklad použití frameworku Tailwind [42]

II. PRAKTICKÁ ČÁST

5 DATABÁZE

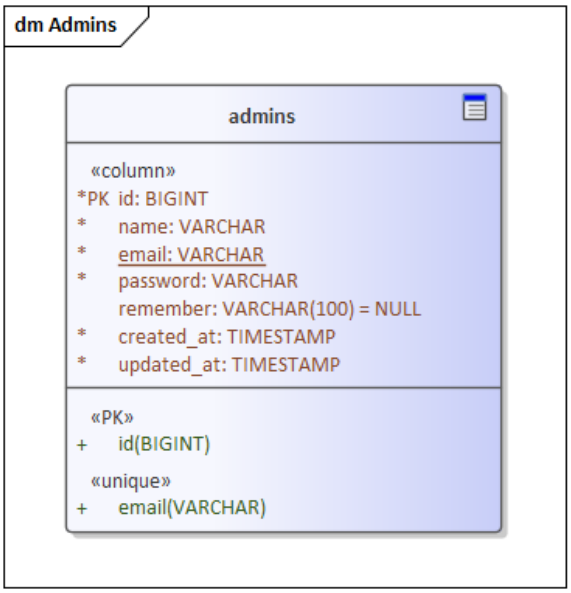
Schéma databáze bylo vytvořeno pomocí migrací ve frameworku, jeho výsledná podoba je znázorněna na obrázku 5.1.



Obrázek 5.1 Model databáze (vlastní zpracování)

5.1 Tabulka admins

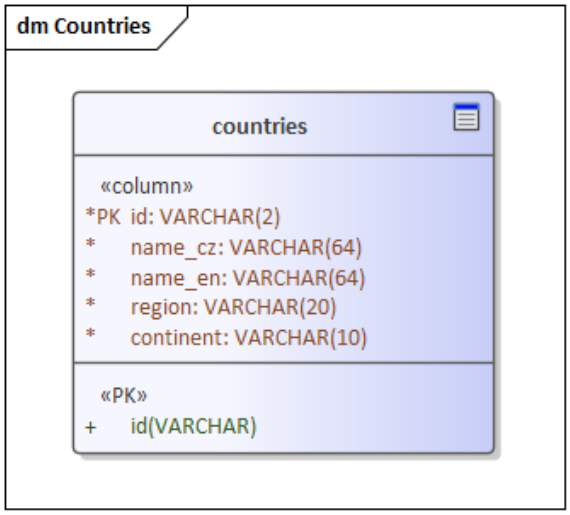
Tabulka `admins` je blíže ukázána na obrázku 5.2. Slouží k uložení přihlašovacích údajů administrátora. V současné době tabulka obsahuje pouze jeden údaj.



Obrázek 5.2 Tabulka admins (vlastní zpracování)

5.2 Tabulka countries

Tabulka `countries` slouží k uložení záznamů o zemích světa. Struktura tabulky je znázorněna na obrázku 5.3.

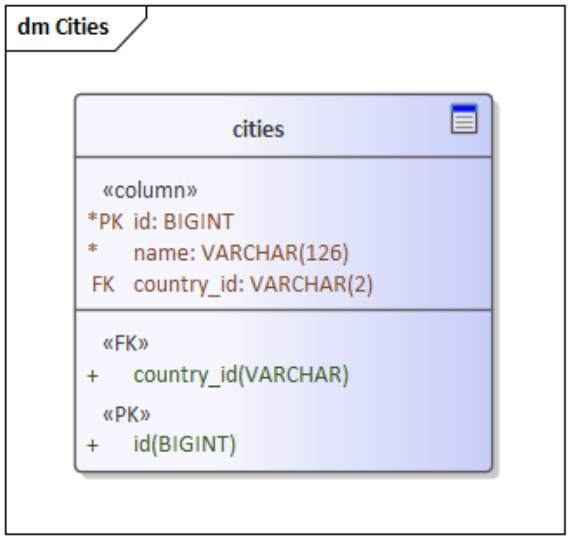


Obrázek 5.3 Tabulka countries (vlastní zpracování)

Tabulka český a anglický název země, kontinent a jeho oblast, ve které se země nachází. Je potřeba k filtraci vyhledávaných výsledků univerzit, podle kterých se student může lépe rozhodnout, které země by chtěl navštívit.

5.3 Tabulka cities

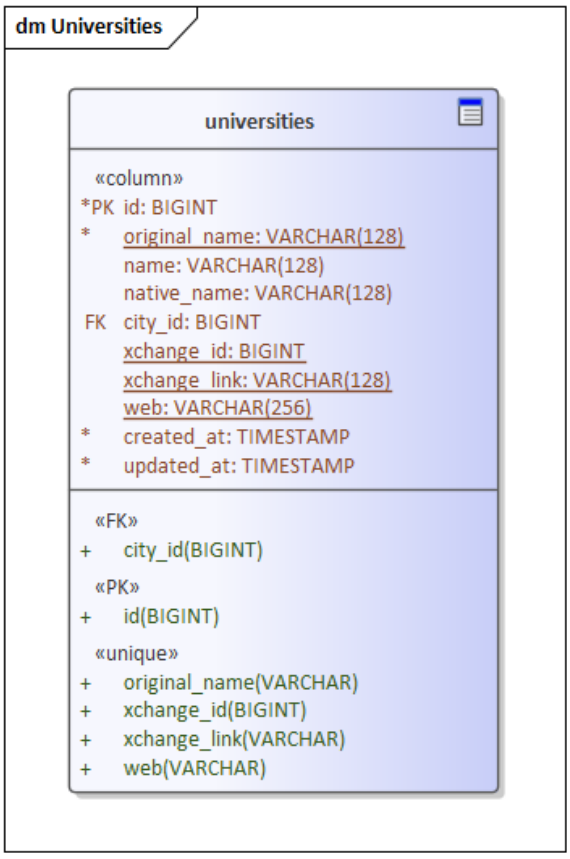
Tabulka `cities` obsahuje města, ve kterých se nachází dostupné univerzity. Tato tabulka je propojena s tabulkou `countries` ve vztahu m:1 - každému městu je přiřazena právě jedna země a každá země může mít několik měst, ve kterých jsou univerzity. Struktura tabulky je znázorněna na obrázku 5.5.



Obrázek 5.4 Tabulka cities (vlastní zpracování)

5.4 Tabulka universities

Tabulka `universities` je určena pro záznamy univerzit. Její struktura je znázorněna v následujícím modelu (Obr. 5.5).



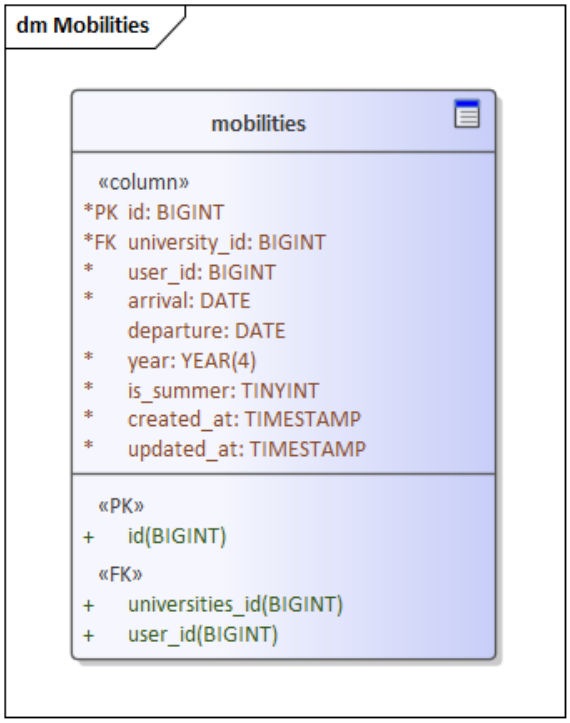
Obrázek 5.5 Tabulka universities (vlastní zpracování)

Každá univerzita obsahuje název `original_name`, pod kterým je uložena v databázi STAG. Pro možnost aktualizace jsou přidány sloupce `name`, do kterého se zapisuje anglický název univerzity, zatímco `native_name` je určen pro název v jazyce dané země. Každá univerzita je propojena s jedním městem v tabulce `cities`.

Tabulka obsahuje dvojí informace o portálu xchange - ID univerzity v databázi xchange a dále odkaz na její profil na tomto portálu. Aplikace měla být původně napojena na jeho databázi, aby četla data o hodnocení univerzit, které by zobrazovala uživateli. Propojení ovšem nebylo možné z důvodu nekompatibility databáze zmíněné v kapitole 2.4.4, protože externí databáze je starší verze, než Laravel podporuje. Z tohoto důvodu byl do naší tabulky přidán parametr `xchange_link`, který v tuto chvíli zajistí alespoň odkaz na profil univerzity a propojení s xchange může být realizováno později pomocí API.

5.5 Tabulka mobilities

Tabulka `mobilities` je určena pro samotná data výjezdů. Obrázek 5.6 znázorňuje její strukturu.

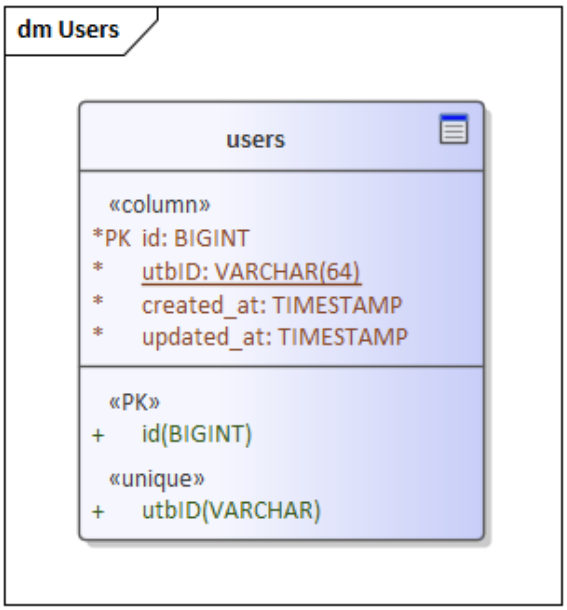


Obrázek 5.6 Tabulka mobilities (vlastní zpracování)

V této tabulce jsou uložena data o příjezdu a odjezdu, která identifikují mobilitu zároveň s propojením na jednoho uživatele, který se mobility účastnil. Mobility jsou propojeny s tabulkou `universities` ve vztahu m:1, kdy každá mobilita náleží právě jedné univerzitě, která obsahuje mnoho výjezdů.

5.6 Tabulka users

Tabulka `users` slouží k autentizaci uživatele, ale také k získání jeho mobilit. Strukturu tabulky popisuje obrázek 5.7.

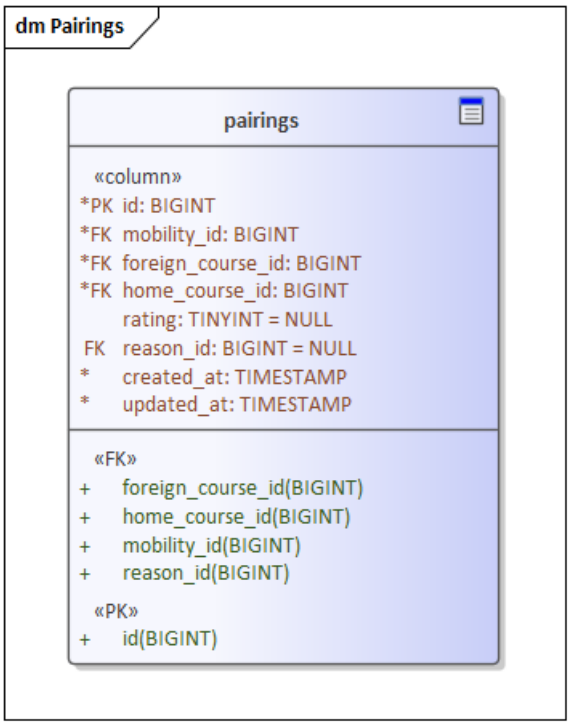


Obrázek 5.7 Tabulka users (vlastní zpracování)

Nejdůležitější položkou této tabulky je `utbID`, které je uloženo v zahashované podobě pomocí SHA-256. Každý uživatel může mít přiděleno žádnou až několik mobilit.

5.7 Tabulka pairings

Tabulka `pairings` slouží k ukládání spárovaných předmětů. Její struktura je popsána na obrázku 5.8.

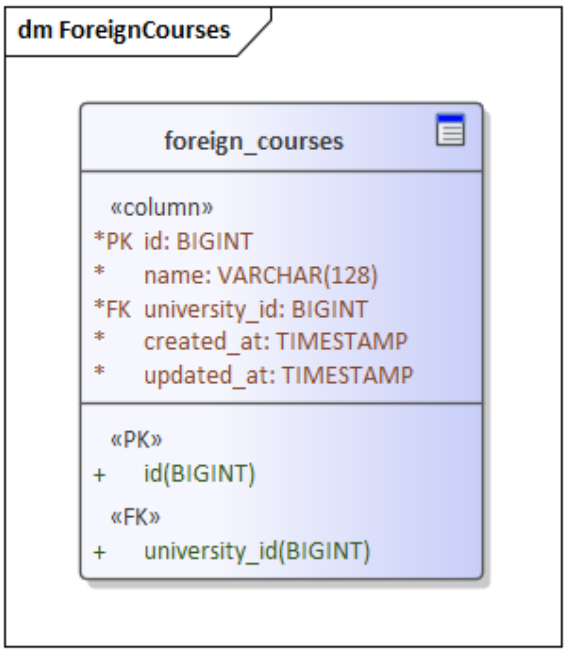


Obrázek 5.8 Tabulka pairings (vlastní zpracování)

Záznam této tabulky patří k právě jedné mobilitě. Zároveň obsahuje informace o zahraničním a domácím předmětu, které byly propojeny. V případě, že bylo propojení zrušeno, je záznam propojen i s důvodem zrušení. Hodnocení spárování je zadáno pomocí čísla od 1 do 5, kdy 1 je nejnižší a 5 nejvyšší hodnocení.

5.8 Tabulka foreign_courses

Tabulka `foreign_courses` (Obr. 5.9) obsahuje informace o předmětu zahraniční univerzity.

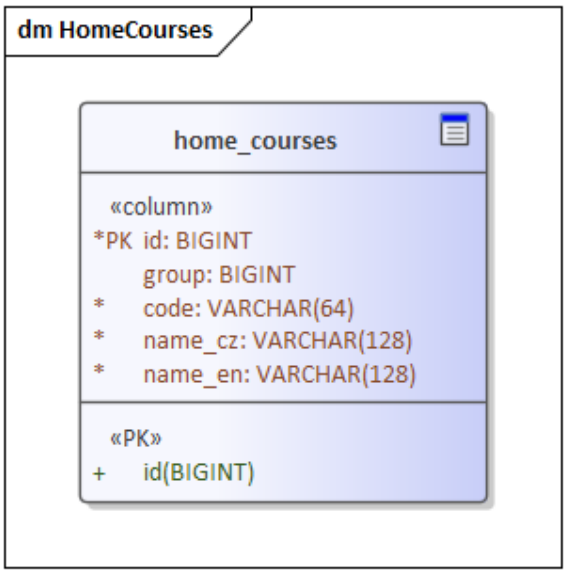


Obrázek 5.9 Tabulka foreign_courses
(vlastní zpracování)

Zahraniční kurz je definován svým názvem a náleží právě jedné univerzitě.

5.9 Tabulka home_courses

Domácí předměty jsou ukládány do tabulky `home_courses`, která je znázorněna na obrázku 5.10.



Obrázek 5.10 Tabulka home_courses
(vlastní zpracování)

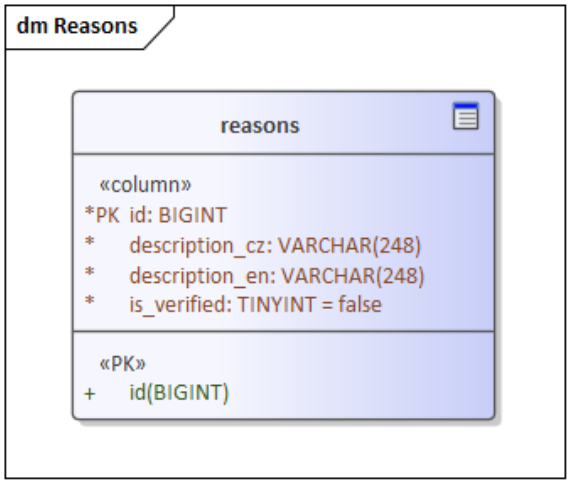
Domácí předměty obsahují kód předmětu ve formě `katedra/zkratka` a dále český a anglický název.

Kvůli možným rozdílným zkratkám mezi různými obory či akreditacemi je zavedeno neunikátní ID ve formě `group`, které udává stejný identifikátor společným předmětům, které mají ale různé definice.

Předmět je vázán na žádné až několik párování a na alespoň jeden obor.

5.10 Tabulka reasons

Tabulka `reasons` slouží k definování možných důvodů zrušení spárování předmětů. Obsahuje informace znázorněné na následujícím diagramu 5.11.

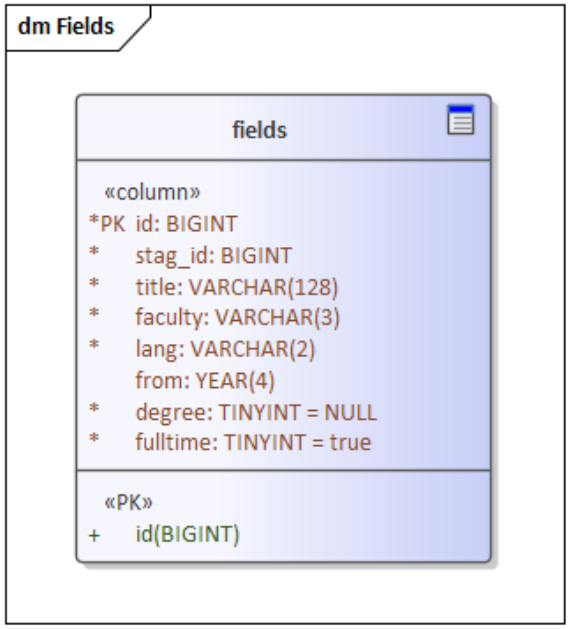


Obrázek 5.11 Tabulka reasons (vlastní zpracování)

Tabulka definuje český a anglický popis a dále určuje pomocí sloupce `is_verified`, jestli byl důvod ověřen administrátorem.

5.11 Tabulka fields

Obory univerzity jsou zapisovány do tabulky `fields` (Obr. 5.12).



Obrázek 5.12 Tabulka fields (vlastní zpracování)

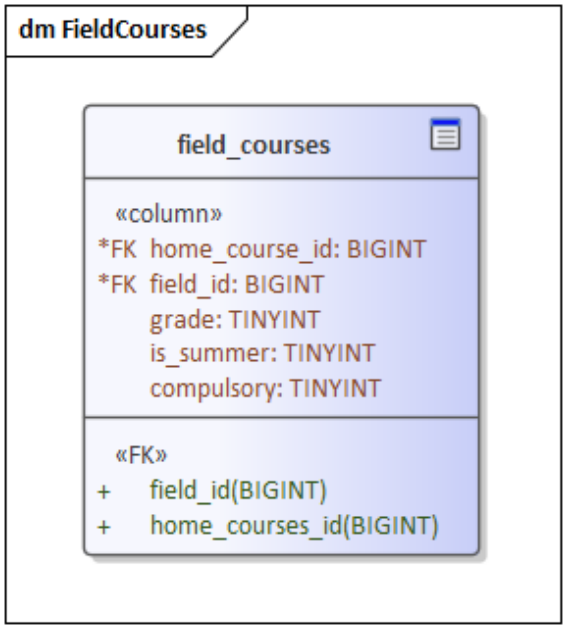
Tabulka obsahuje informace o názvu oboru, identifikátoru STAG, pomocí kterého získáme seznam jeho předmětů, informace o fakultě, jazyku a typu studia a oboru. Dále

je potřeba definovat i rok, od kterého je obor platný, aby student mohl rozlišit, který ze stejných oborů zahrnuje jeho předměty.

Tabulka `fields` je propojena s tabulkou `home_courses` vztahem m:n, která umožní, aby jeden předmět mohl být sdílen mezi více obory, ale zároveň jeden obor obsahuje více předmětů. Z tohoto důvodu musí být definována propojující tabulka popsána v následující kapitole.

5.12 Tabulka field_courses

Tabulka `field_courses` slouží k propojení záznamů tabulky `home_courses` a tabulky `fields`. Struktura tabulky je znázorněna na obrázku 5.13.



Obrázek 5.13 Tabulka field_courses
(vlastní zpracování)

Kromě cizích klíčů tabulka obsahuje informace o to, ve kterém ročníku je doporučeno tento předmět studovat v tomto oboru a také v jakém semestru. Mezi poslední položku patří údaj `compulsory`, který definuje, jestli je předmět pro tento obor volitelný či nikoli.

5.13 Naplnění databáze

Databáze je prvotně naplněna daty pomocí Seederů. Díky nim je inicializována tabulka zemí, do které jsou pomocí předem vytvořeného souboru JSON vložena data o světových státech a oblastech.

Dalším krokem je získání dat o domácích předmětech pomocí API informačního portálu STAG. Protože API nepovoluje fetch všech předmětů bez autentizace, nejprve

jsou získány informace o univerzitních programech, pomocí kterých se dotazuje na data o oborech. Následně můžeme získat všechny předměty oborů (Kód 5.1), které potom ještě propojíme s ostatními obory, které kurzy sdílí.

```
1   public function getCourses(int $id)
2   {
3     $response = Http::withOptions(['verify' => false])->get(
4     'https://stag-ws.utb.cz/ws/services/rest2/predmety/
      getPredmetyByObor', [
5     'oborIdno' => $id,
6     'outputFormat' => 'JSON'
7     ]);
8     if ($response->ok()) {
9       return CourseCollection::toArray($response->json());
10    }
11    return null;
12  }
13
```

Kód 5.1 Seeder předmětů (vlastní zpracování)

Data kurzů musí být ukládána z důvodů nových akreditací, kdy dochází ke změně zkratky předmětu, nikoliv jejího obsahu. Proto je implementován seeder skupin (Kód 5.2), který předměty spojí pomocí neunikátního ID.

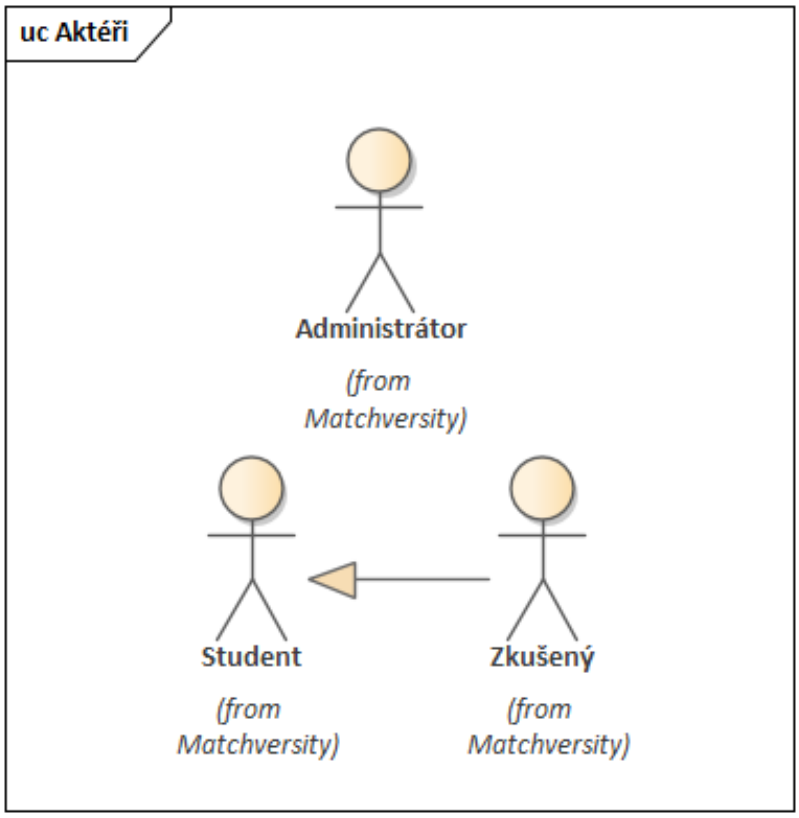
```
1   public function run()
2   {
3     $courses = HomeCourse::all();
4     $index = 1;
5     foreach ($courses as $course) {
6       if (empty(HomeCourse::find($course->id)->group)) {
7         $group = HomeCourse::where('name_cz', '=', $course->name_cz)
8         ->orWhere('name_en', '=', $course->name_en)
9         ->orWhere('name_en', '=', $course->name_cz)
10        ->orWhere('name_cz', '=', $course->name_en)->get();
11        if ($group->count() > 1) {
12          foreach ($group as $item) {
13            $item->group = $index;
14            $item->save();
15          }
16          ++$index;
17        }
18      }
19    }
20  }
21
```

Kód 5.2 Seskupení předmětů (vlastní zpracování)

Dále je vytvořen `UniversitySeeder`, kde jsou předpřipravena data o univerzitách, aby se nemusela manuálně vkládat administrátorem pomocí formuláře.

6 SPECIFICKÉ POŽADAVKY

Obrázek 6.2 zobrazuje uživatele systému.



Obrázek 6.1 Aktéři systému (vlastní zpracování)

Student Hlavním uživatelem aplikace je student UTB, obvykle ve věku od 19 do 26 let, který se zajímá o krátkodobé studium v zahraničí. Má základní znalosti používání webového prohlížeče. Student se dělí do dvou kategorií:

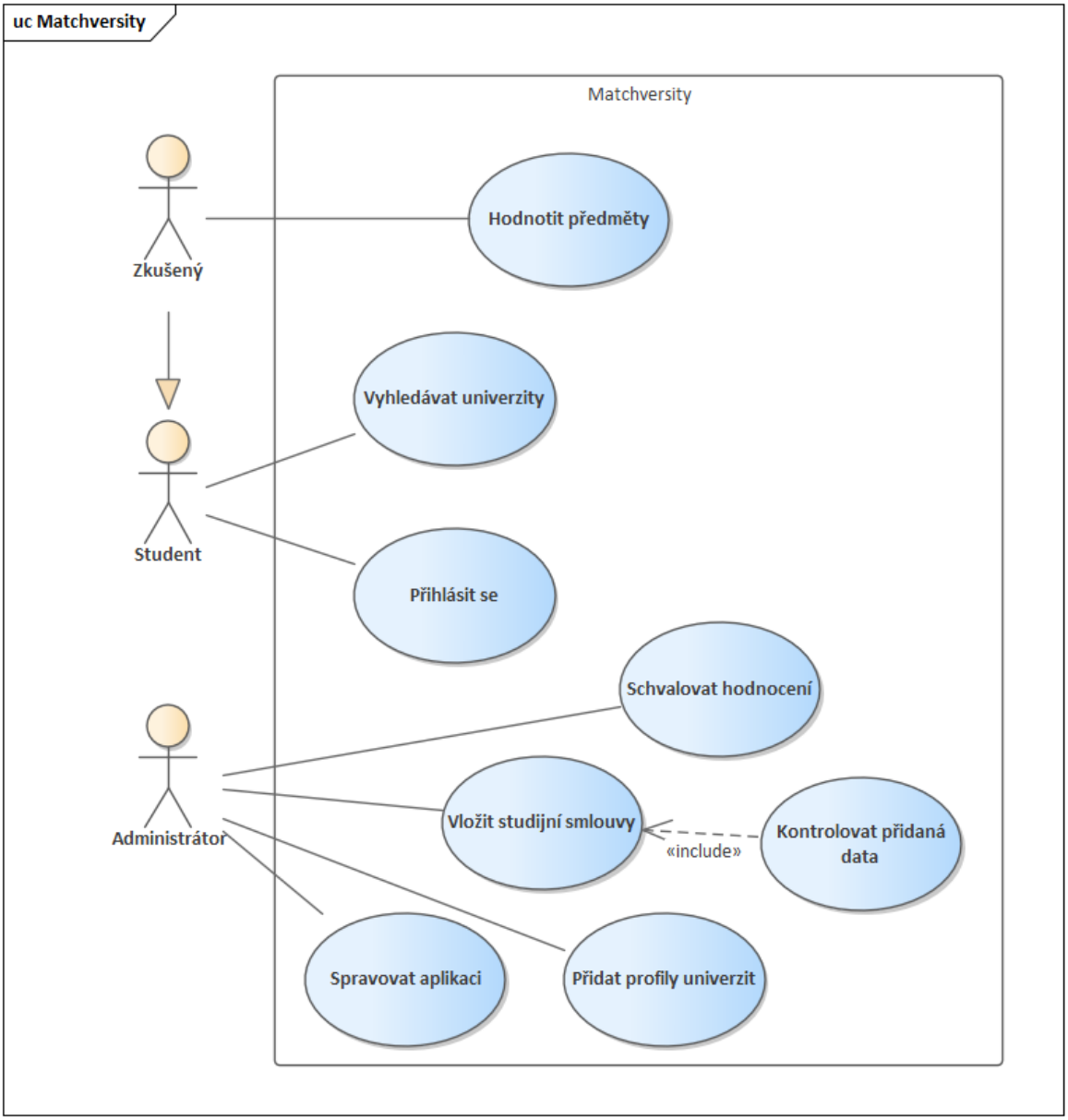
1. student, který uvažuje o absolvování studijního pobytu v zahraničí v rámci studia na UTB (dále jen zájemce),
2. student, který má již zkušenosti se zahraničním výjezdem v rámci studia na UTB (dále jen zkušený).

Zájemce pravděpodobně využije aplikaci minimálně jednou až dvakrát. V případě převyšujícího zájmu se používání může zvyšovat.

Zkušený Pokud zájemce, absolvuje studijní pobyt, pak se z něj stává zkušený, ale zároveň může aplikaci nadále využívat jako zájemce. Zkušený student má v databázi uložený záznam o mobilitě a tedy může tuto mobilitu hodnotit.

Administrátor Administrátor má na starosti správu chodu aplikace. Jeho úkolem je nahrávat data, která získá ze STAG, případně tato data validovat a kontrolovat.

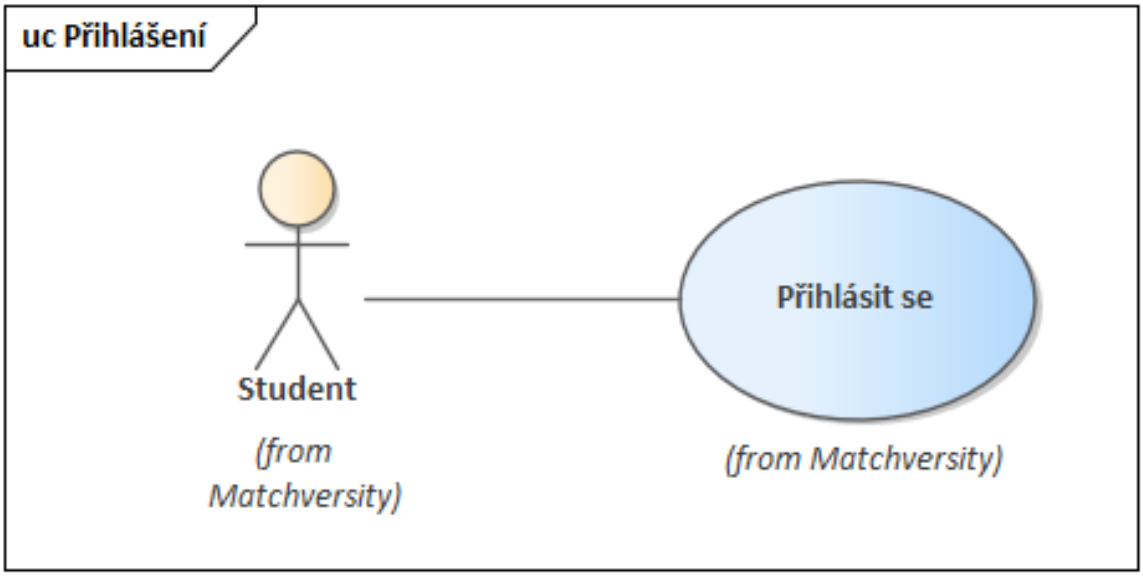
Na obrázku 6.2 je znázorněn diagram případů užití projektu.



Obrázek 6.2 Diagram případů užití (vlastní zpracování)

6.1 Přihlášení uživatele

Přihlášení uživatele potvrzuje identitu studenta UTB. Tato funkce je důležitá pro umožnění autorizovaného hodnocení mobilit zkušeným studentům. Na obrázku (Obr. 6.3) je znázorněn diagram případu užití přihlášení uživatele.



Obrázek 6.3 Diagram případů užití přihlášení (vlastní zpracování)

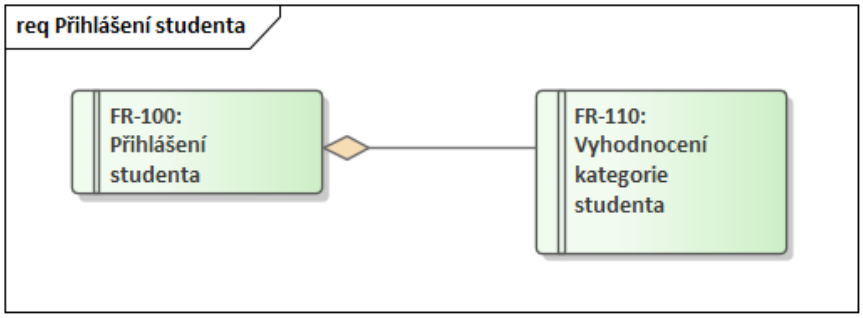
6.1.1 Sekvence podnětů a odpovědí

Podnět: Uživatel zvolil `Moje mobility`.

1. Systém přesměruje uživatele na přihlašovací stránku.
2. Uživatel zadá autentizační údaje.
3. Systém ověří zadané údaje.
4. Systém přesměruje studenta na stránku mobilit.
   - *Výjimka:* Uživatele se nepodařilo autentizovat.
     (a) Systém vrátí chybovou hlášku na přihlašovací stránku.
     (b) Pokračování v kroku 2.

6.1.2 Funkční požadavky

Funkční požadavky jsou znázorněny na obrázku 6.4.



Obrázek 6.4 Diagram požadavků na přihlášení (vlastní zpracování)

FR-100 Přihlášení studenta Aplikace umožní uživateli autentizaci jako student UTB.

FR-110 Vyhodnocení kategorie studenta Systém ověří existenci uživatele v databázi. V případě, že neexistuje jeho záznam, je do tabulky přidán.

6.1.3 Implementace

K přihlášení uživatele jsem využila třídu `Auth`, ve které jsem zablokovala možnosti registrace a změn hesel.

Pro účely bakalářské práce je k testování funkčnosti využito falešné přihlašování (Kód 6.1), díky kterému se lze podívat na profil uživatele a hodnocení jeho výjezdu.

```
1  public function login(Request $request)
2  {
3    $credentials = $request->only('email', 'password');
4
5    if ($credentials['email'] == 'test@utb.cz' &&
6      $credentials['password'] == '1234') {
7        Auth::login(User::find(353));
8        return redirect()->intended('/');
9    }
10
11   return back()->withErrors([
12     'email' =>
13       'The provided credentials do not match our records.',
14   ]);
15 }
16
```

Kód 6.1 Login uživatele(vlastní zpracování)

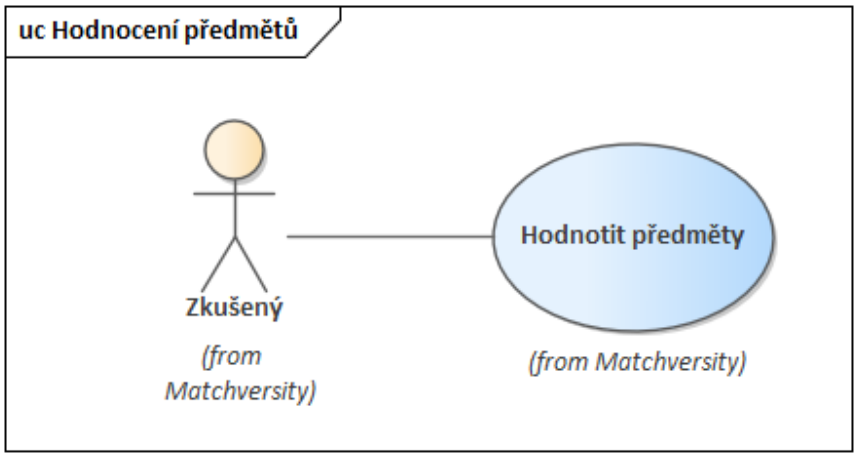
V budoucnosti bude tato funkce implementována pomocí univerzitního systému autentizace *shibboleth*, který zajistí přihlašování studentů.

6.2 Hodnocení spárování

Pro zkušeného studenta je tu možnost prohlédnout a ohodnotit spárované předměty, které absolvoval na výjezdech, aby pomohli s rozhodováním studentům se zájmem o výjezd.

Tato funkce není nezbytná pro funkci aplikace, nicméně jedná se o užitečnou funkci se středně vysokou prioritou.

Diagram případu užití hodnocení spárování je znázorněn na obrázku 6.5.



Obrázek 6.5 Diagram případů užití hodnocení spárování (vlastní zpracování)

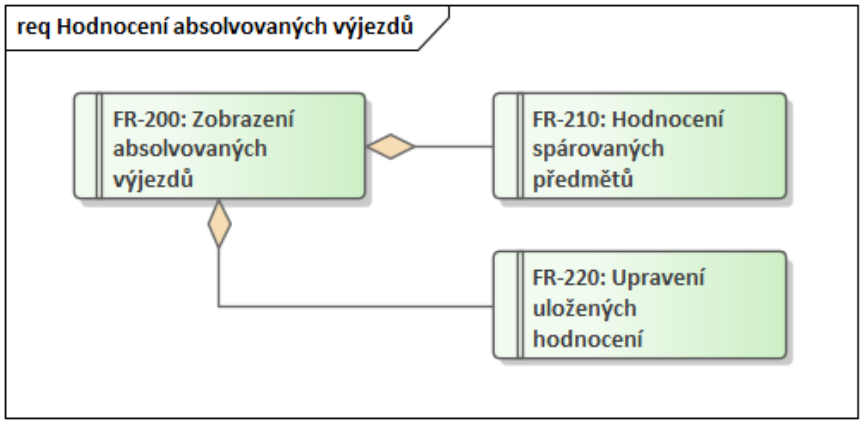
6.2.1 Sekvence podnětů a odpovědí

Podnět: Uživatel se přihlásil do systému a jeho záznam je uložený v databázi.

1. Systém zobrazí seznam studentových výjezdů.
2. Uživatel vybere výjezd k zobrazení
3. Systém ověří zadané údaje.
4. Systém zobrazí podrobnosti výjezdu s formulářem o předmětech.
5. Uživatel ohodnotí předměty a poskytne důvody rozpojení.
6. Systém uloží hodnocení do databáze a pokračuje s krokem 1.
   - *Výjimka:* Zadaná data nebyla validní.
     (a) Systém vrátí chybovou hlášku.
     (b) Pokračování v kroku 4.

6.2.2 Funkční požadavky

Funkční požadavky jsou zobrazeny na diagramu v obrázku 6.6



Obrázek 6.6 Diagram požadavků na hodnocení (vlastní zpracování)

FR-200 Zobrazení absolvovaných výjezdů Aplikace umožní studentovi zobrazit svoje absolvované výjezdy, které jsou v databázi.

FR-210 Hodnocení spárovaných předmětů Aplikace umožní ohodnotit předměty pomocí hvězdiček nebo zadání důvodu zrušení spárování.

FR-220 Upravení uložených hodnoceních Aplikace umožní studentovi vrátit se k již ohodnocenému výjezdu a upravit své původní hodnocení.

6.2.3 Implementace

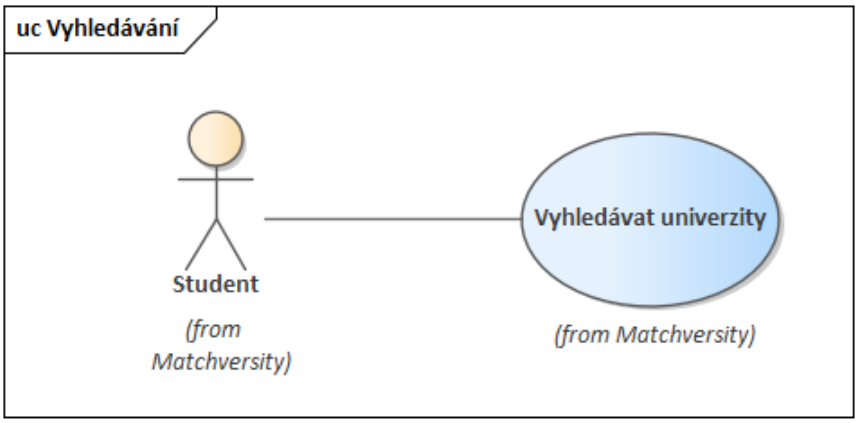
Po přihlášení uživatele jsou z databáze získána data o jeho mobilitách. Po zvolení mobility se objeví formulář na hodnocení výjezdu, který využívá k ohodnocení systém hvězdiček a v případě zrušeného spárování zadává pomocí select důvod jeho rozpojení, nebo může definovat nový. Validovaným POST requestem jsou pak data uložena do databáze.

```
1   public function updateMobility($data)
2   {
3     if (array_key_exists('rate',$data)) {
4       $this->saveRatings($data['rate']);
5     }
6     if (array_key_exists('reason', $data)) {
7       $this->unlinkCourses(
8         $data['reason'],
9         array_key_exists('new', $data) ? $data['new'] : null
10      );
11    }
12    $this->save();
13  }
14
```

Kód 6.2 Uložení hodnocení (vlastní zpracování)

6.3 Vyhledávání univerzit

Hlavní funkcí aplikace je vyhledávání zahraničních univerzit podle předmětů ke spárování, které byly uskutečněny na minulých výjezdech. Tato funkce má nejvyšší prioritu. Diagram případů užití vyhledávání univerzit je znázorněn na obrázku 6.7.



Obrázek 6.7 Diagram případů užití vyhledávání (vlastní zpracování)

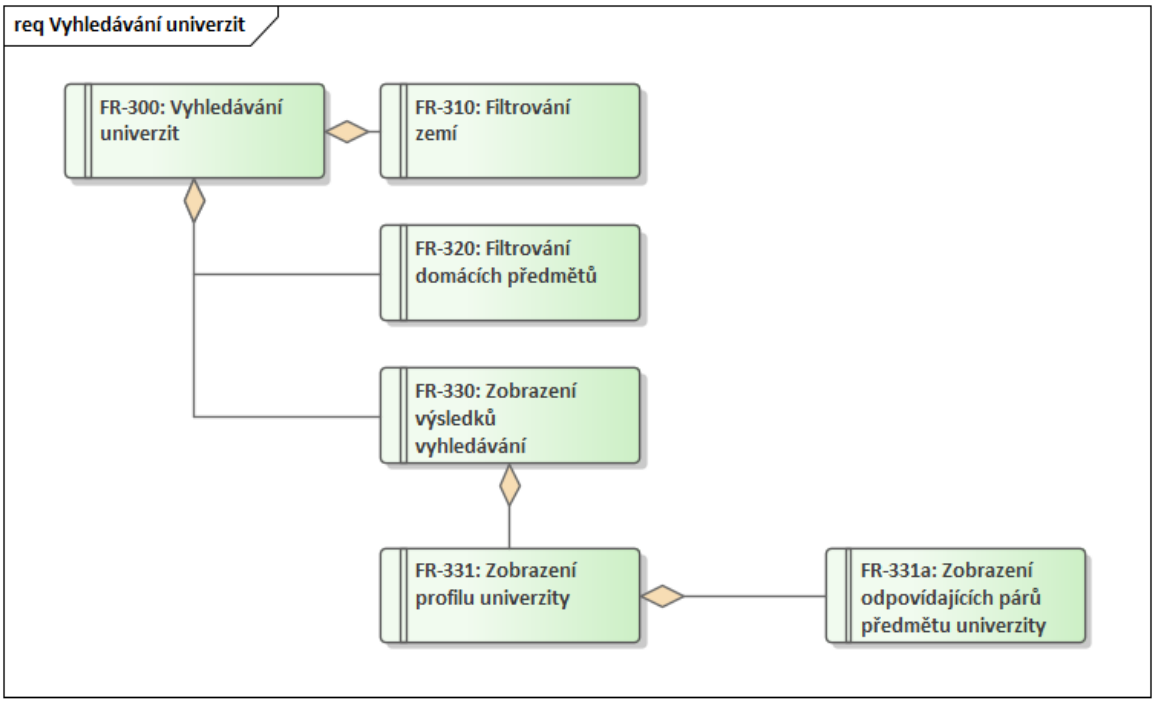
6.3.1 Sekvence podnětů a odpovědí

Podnět: Uživatel zvolí vyhledávání univerzit.

1. Uživatel zadá parametry pro výběr předmětů.
2. Systém vygeneruje seznam předmětů.
3. Student vyfiltruje požadované předměty a země.
4. Systém zobrazí dostupné univerzity.
5. Uživatel zvolí univerzitu
6. Systém otevře profil univerzity a zobrazí seznam předmětů ke spárování a jejich hodnocení.

6.3.2 Funkční požadavky

Diagram požadavků popisuje obrázek 6.8.



Obrázek 6.8 Diagram požadavků na vyhledávání (vlastní zpracování)

FR-300 Vyhledávní univerzit Aplikace umožní studentovi vyhledat zahraniční univerzity dle domácích předmětů.

FR-310 Filtrování zemí Aplikace umožní studentovi zvolit země, které chce zahrnout ve výsledcích.

FR-320 Filtrování domácích předmětů Student si může přidat/odebrat domácí předměty, podle kterých chce vyhledat univerzity.

FR-330 Zobrazení výsledků vyhledávání Aplikace zobrazí výsledky vyhledávání seřazené podle počtu uskutečněných mobilit.

FR-331 Zobrazení profilu univerzity Aplikace poskytne základní informace o univerzitě.

FR-332 Zobrazení odpovídajících spárovaných předmětů univerzity Aplikace zobrazí nalezené spárované předměty a jejich hodnocení.

6.3.3 Implementace

K vyhledávání předmětů pomáhá API, které zprostředkovává potřebné dotazy pro frontend. Pomocí vybrané fakulty a typu studia jsou získána data o dostupných oborech

v prezenční a kombinované formě. Následně může uživatel vyhledat všechny předměty studia případně je vyfiltrovat podle ročníku. Jakmile seznam získá, posílají se informace o vybraných předmětech, aby byly vyfiltrovány země, které nemají dostupné univerzity s daným párováním (Kód 6.3). Po zaslání requestu, která jsou z důvodu velkého počtu vyhledávaných informací zasílány přes POST, jsou z databáze získána data o dostupných univerzitách a parametry vyhledávání jsou uloženy do session (Kód 6.4). Při otevření profilu jsou zobrazeny předměty univerzity a pokud session obsahuje i vyhledávané kurzy, pak jsou zobrazena i vhodná spárování.

```
  methods: {
    async findFields() {
      if (this.type && this.faculty) {
        try {
          this.fields = [];
          const response = await fetch(this.fieldRequest);
          this.fields = await response.json();
        }
        catch (error) {
          console.log(error);
        }
      }
    },
    async findCourses() {
      if (this.field) {
        try {
          const response = await fetch(this.courseRequest, {
            headers: {
              "X-CSRF-TOKEN": this.token,
              "Access-Control-Allow-Credentials" : true
            },
            method: "GET",
            credentials: "same-origin"});
          const courseList = await response.json();
          this.summerList = Object.assign(
            {},
            courseList.summer,
            this.summerList
          );
          this.winterList = Object.assign(
            {},
            courseList.winter,
            this.summerList
          );
          const session = Object.keys(courseList.summer);
          await this.fetchCountries(
            session.concat(Object.keys(courseList.winter))
          );
        }
        catch (error) {
          console.log(error);
        }
      }
    }
  }

```

Kód 6.3 Dostupné země (vlastní zpracování)

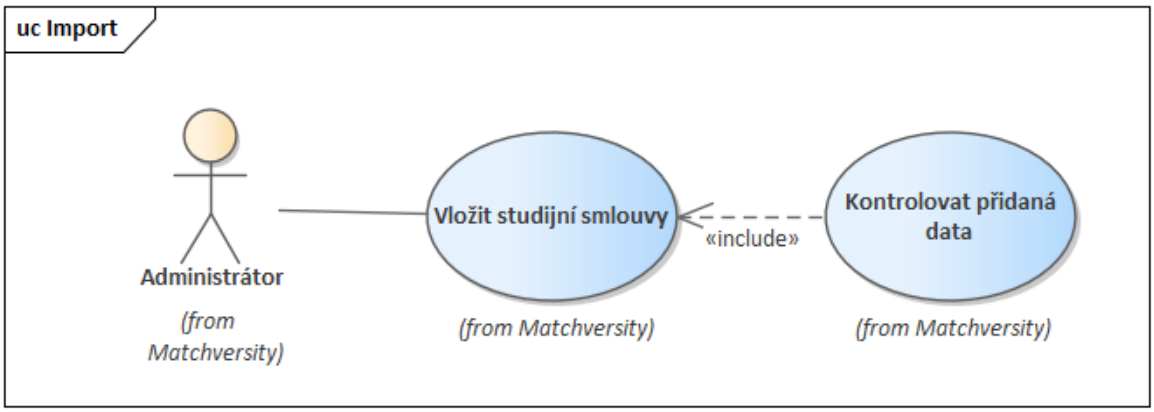
```
public function index(SearchRequest $request)
{
  if ($request) {
    $validated = $request->validated();
    HomeCourse::setSession($validated);
    Country::setSession($validated);
  }
  $universities = University::findResults();
  return view('universities.hunter', [
    'top3' => $universities->count() > 3 ?
      $universities->splice(0, 3) : null,
    'universities' => $universities,
  ]);
}
```

Kód 6.4 Vyhledávání (vlastní zpracování)

6.4 Import studijních smluv

Aby databáze odpovídala aktuálním informacím, je třeba 2x ročně vložit nové studijní smlouvy. Ty se uzavírají pro zimní a letní semestr a měsíc po zahájení semestru se aktualizují.

Priorita této funkce je nejvyšší - bez informací v databázi nemůže aplikace sloužit jejím účelům. Na obrázku 6.9 je znázorněn diagram případu užití importu studijních smluv.



Obrázek 6.9 Diagram případů užití import (vlastní zpracování)

6.4.1 Sekvence podnětů a odpovědí

Podnět: Administrátor se přihlásí do aplikace.

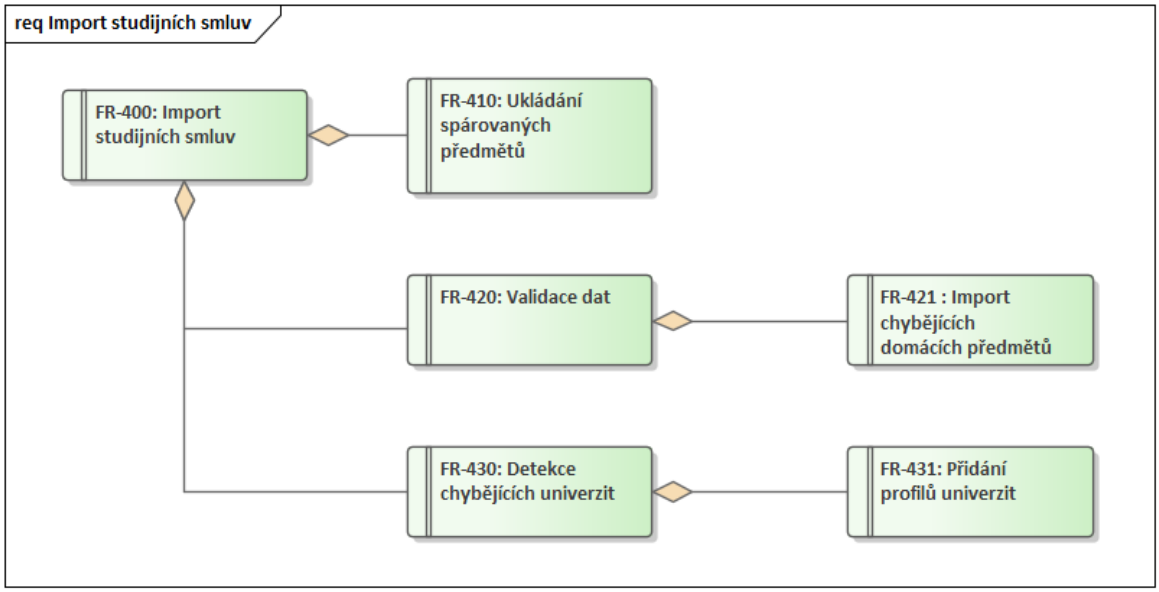
1. Systém zobrazí formulář na import smluv.
2. Administrátor vloží soubor se studijními smlouvami ze STAGu.
3. Systém validuje a uloží správně zadané údaje.

- *Výjimka:* Soubor nelze parsovat
  (a) Systém vrátí chybovou hlášku na stránku mobilit.

4. Systém přesměruje administrátora zpět na stránku s mobilitami.
   - *Výjimka:* Soubor obsahoval chybné údaje
     (a) Systém přesměruje administrátora na stránku k validaci chybných údajů.
     (b) Administrátor zkontroluje a opraví data a odešle formulář.
     (c) Návrat ke kroku 3.

6.4.2 Funkční požadavky

Funkční požadavky jsou znázorněny na obrázku 6.10.



Obrázek 6.10 Diagram požadavků na import (vlastní zpracování)

FR-400 Import studijních smluv Aplikace umožní administrátorovi vložit soubor se studijními smlouvami.

FR-410 Ukládání spárovaných předmětů Systém na základě studijních smluv ze STAGu uloží informace o spárovaných předmětech pro vyhledávání.

FR-420 Validace dat Systém parsuje data ze souboru administrátora a validuje jeho obsah podle typu sloupců. V případě chybného vstupu je administrátor přesměrován ke kontrole a opravě dat.

FR-421 Import chybějících domácích předmětů Systém vyhledá předměty ze souboru v databázi. V případě, že aplikace data neobsahuje, je odeslán dotaz na STAG API pro uložení jeho informací.

FR-430 Detekce chybějících univerzit Systém mobility přiděluje podle názvů univerzity jejímu záznamu v databázi.

FR-431 Přidání profilů univerzit V případě, že databáze neobsahuje profil, je vytvořen nový se zadaným STAG jménem.

6.4.3 Implementace

Pro import smluv slouží administrátorovi jednoduchý formulář, do kterého se vkládají data exportovaná ze STAG. Pokud je soubor správný, pak jsou data validována pomocí třídy `FileValidator` (Kód 6.5), který řádky o spárovaných předmětech z databáze spojí do jednotlivých mobilit, jichž data jsou pak jednotlivě podle typu validována (Kód 6.6), případně i upravována, aby zjednodušila práci administrátorovi.

```
    public function getData($file)
    {
      $this->file = $file;
      HomeCourseValidator::refreshField();
      $this->parseData();
      if ($this->data) {
        $mobilities = [];
        foreach ($this->data as $row) {
          if (($row[ImportColumns::DEGREE] !== 'doktorsky') &&
          (!empty($row[ImportColumns::HOME_COURSE])) &&
          !empty($row[ImportColumns::STUDENT_ID])) {
            $this->addMobility($mobilities, $row);
          }
        }
        $this->validateMobilities($mobilities, false);
        HomeCourseValidator::refreshField();
        if (count($this->toCheck) < 20) {
          return $this->validated;
        }
      }
      return null;
    }

```

Kód 6.5 FileValidator (vlastní zpracování)

To platí například pro názvy zahraničních předmětů, které jsou upravovány tak, aby měli stejný formát typu `title case`. Dále se mění automaticky roky výjezdů, které v letním semestru obsahují rok nižší, nebo například i aplikace detekuje případné chyby v semestru. Pokud je záznam označený jako semestr letní, ale podle dat příjezdu a odjezdu odpovídají semestru zimnímu či naopak, aplikace tato data změní a upozorní na to administrátora.

```
public function validate()
{
  $this->message = "";
  $this->isValid = !in_array(false, $this->getValidatedData());
  return $this->isValid;
}

private function getValidatedData()
{
  $validations = [
  $this->arrival->validate(),
  $this->departure->validate(),
  $this->student->validate(),
  $this->semester->validate(),
  $this->year->validate(),
  $this->university->validate()
  ];
  foreach ($this->pairings as $pairing) {
    array_push($validations, $pairing->validate());
  }
  return $validations;
}

```

Kód 6.6 MobilityValidator (vlastní zpracování)

Další důležitá kontrola je v případě, že daný student má více než jednu mobilitu ve stejný čas. Tehdy je uživatel upozorněn, že pozdější mobilita přepíše tu předchozí a má možnost tedy zvolit, kterou mobilitu ponechá.

```
  public function validate()
{
  if (empty($this->data)) {
    return $this->result("Student ID is missing.");
  }
  if (strlen($this->data) != 64) {
    if (!preg_match("/[A-Z]{2}[0-9]{6}/", $this->data)) {
      return $this->result("Wrong student ID format.");
    }
    $this->data = hash('sha256', $this->data);
  }
  if ($this->arrival->isValid && $this->departure->isValid) {
    if (!User::hasUniqueMobility(
      $this->data,
      $this->arrival->data,
      $this->departure->data)) {
        return $this->result(self::REWRITE);
    }
  }
  return $this->result("");
}

```

Kód 6.7 StudentValidator (vlastní zpracování)

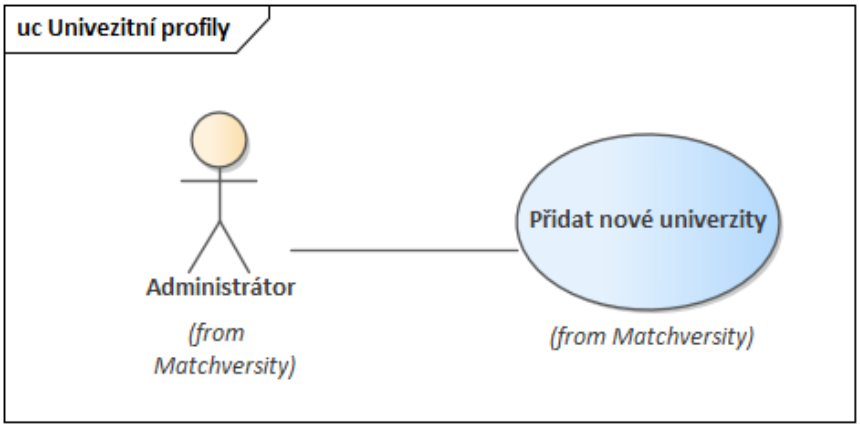
Jelikož jsou data velmi obsáhlá a překračují rámec povolených vstupů pro formulář, musí být omezena. V případě tedy, že soubor obsahuje více než dvacet chybných mobilit, není potom parsován a administrátor je upozorněn, aby zkontroloval soubor. Pokud se pokusí o vložení souboru se stejnými daty, data pak nejsou uložena.

6.5 Vytváření univerzitních profilů

Data ve STAG nepodléhají validaci a jsou často neupravená. Proto je nutné profily univerzit přidávat manuálně, aby aplikace byla aktuální a přehledná.

Priorita toho případu je vysoká a má velký vliv na UX aplikace.

Diagram případu užití je znázorněn na obrázku 6.11.



Obrázek 6.11 Diagram případů užití profily univerzit (vlastní zpracování)

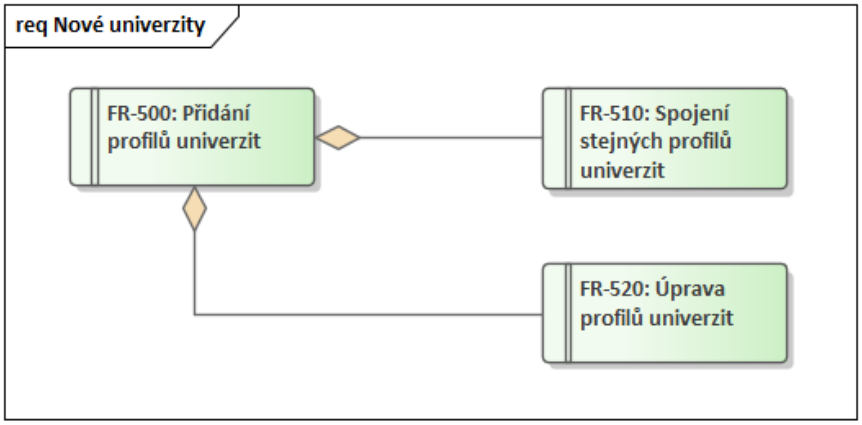
6.5.1 Sekvence podnětů a odpovědí

Podnět: Administrátor zvolí profil univerzity.

1. Systém zobrazí formulář pro úpravu profilu univerzity.
2. Administrátor zadá .
3. Systém validuje a uloží zadané údaje.
   - *Výjimka:* Udaje nejsou správné
     (a) Systém vrátí chybovou hlášku.

6.5.2 Funkční požadavky

Funkční požadavky jsou zobrazeny v obrázku 6.12.



Obrázek 6.12 Diagram požadavků na přihlášení (vlastní zpracování)

FR-500 Přidání profilů univerzit Aplikace umožňuje administrátorovi vytvořit profil pro univerzitu z databáze.

FR-510 Spojení stejných profilů univerzit Aplikace umožňuje administrátorovi sjednotit profil univerzity s jiným existujícím profilem v databázi.

FR-520 Úprava profilů univerzit Aplikace umožňuje administrátorovi upravit data profilů univerzit.

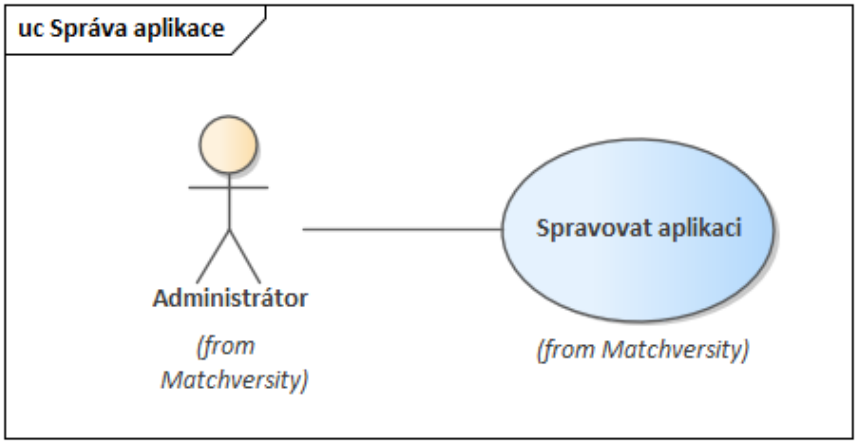
6.5.3 Implementace

Implementace vytváření profilů univerzit je přes jednoduchý formulář, který přes POST request ukládá data. V případě toho, že univerzita v databázi existuje pod více než jedním názvem, je možné tyto profily spojit dohromady, kdy profil, ze kterého je spojení vyžádáno, je smazán a jeho data jsou přepojena k druhé univerzitě.

6.6 Správa aplikace

Aby aplikace poskytovala aktuální informace, bude možné základní informace aplikace spravovat administrátorem. Priorita této funkce je střední.

Diagram případu užití je znázorněn na obrázku 6.13



Obrázek 6.13 Diagram případů užití správa aplikace (vlastní zpracování)

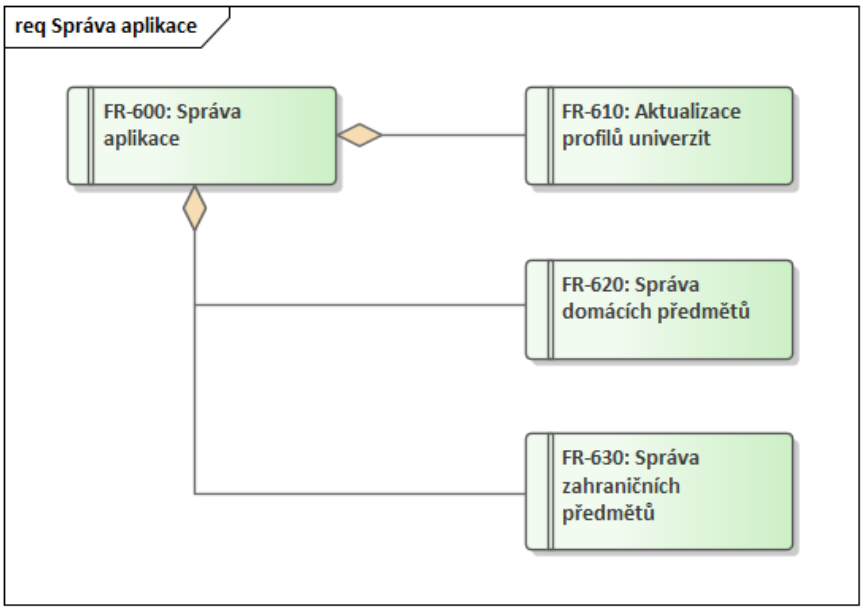
6.6.1 Sekvence podnětů a odpovědí

Podnět: Data aplikace potřebují aktualizovat.

1. Systém zobrazí formulář pro úpravu dat.
2. Administrátor upraví data.
3. Systém validuje a uloží zadané údaje.
   - *Výjimka:* Udaje nejsou správné
     (a) Systém vrátí chybovou hlášku.

6.6.2 Funkční požadavky

Funkční požadavky znázorňuje obrázek 6.14.



Obrázek 6.14 Diagram požadavků na přihlášení (vlastní zpracování)

FR-600 Správa aplikace Administrátor může aktualizovat informace o předmětech.

FR-610 Správa domácích předmětů Aplikace umožní administrátorovi přeskupovat domácí předměty.

FR-620 Správa zahraničních předmětů Aplikace umožní administrátorovi přejmenovávat předměty.

6.6.3 Implementace

Správa zahraničních předmětů je rozdělena podle univerzit. Administrátor může měnit názvy předmětů dané univerzity. V případě, že je předmět nahrazen názvem, který pod touto univerzitou již existuje, jsou jeho párování přepojena na původní název a změněný předmět je vymazán. Vyhneme se tak duplicitám ve vyhledávání.

Pro domácí předměty je implementována funkce přeskupování předmětů, které mají stejnou funkci. Jelikož databáze obsahuje přes sedm tisíc záznamů, které jsou seskupeny do dvou tisíc skupin, práce s těmito daty nejde zcela ulehčit. Předměty jsou tedy seřazeny abecedně a lze měnit jejich skupinové ID.

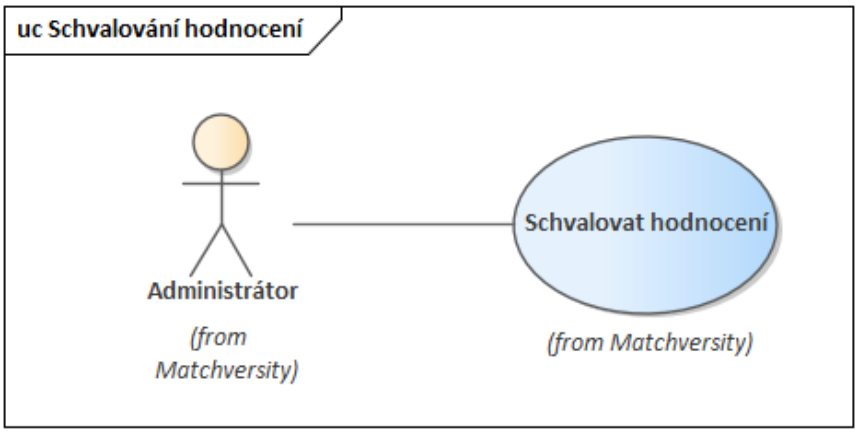
6.7 Schvalování hodnocení

Jelikož uživatelé mohou poskytnout důvod pro zrušené předměty, je třeba tento vstup validovat tak, aby byl srozumitelný pro uživatele. Aplikace defaultně obsahuje několik

důvodů, které může student zvolit, ovšem může se vyskytnout takový, se kterým není mezinárodní oddělení obeznámeno. Z tohoto důvodu může uživatel poskytnout nový vstup, který je pak validován administrátorem.

Priorita tohoto případu je nízká - případně se může nastavit volba pouze z defaultních sedmi.

Případ užití je znázorněn na obrázku 6.15.



Obrázek 6.15 Diagram případů užití schvalování hodnocení (vlastní zpracování)

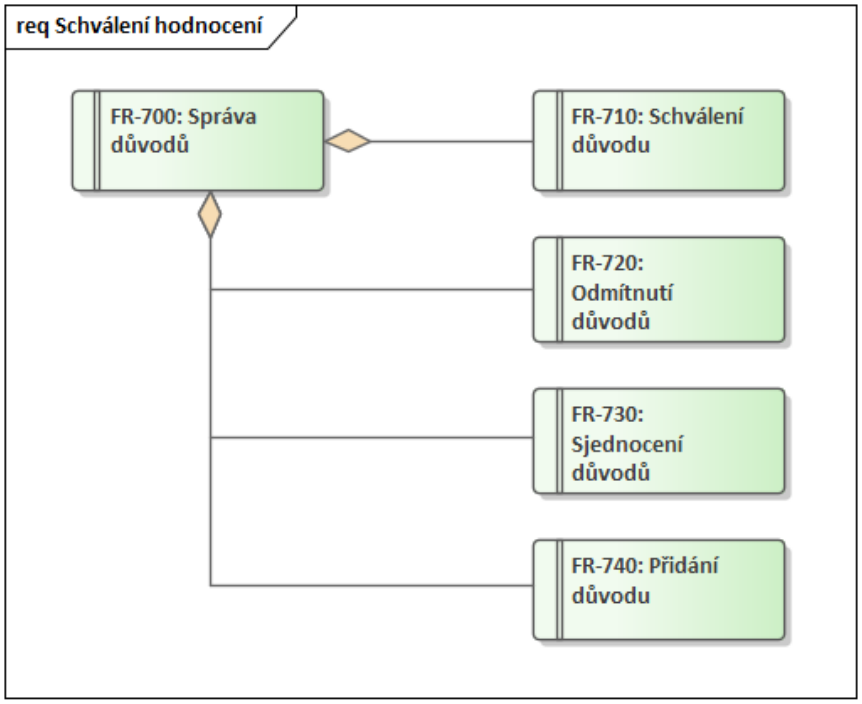
6.7.1 Sekvence podnětů a odpovědí

Podnět: Byl přidán nový důvod v hodnocení.

1. Systém zobrazí formulář pro úpravu dat.
2. Administrátor upraví data.
3. Systém validuje a uloží zadané údaje.
   - *Výjimka:* Udaje nejsou správné
     (a) Systém vrátí chybovou hlášku.

6.7.2 Funkční požadavky

Diagram požadavků je znázorněn v obrázku 6.16.



Obrázek 6.16 Diagram požadavků na přihlášení (vlastní zpracování)

FR-700 Správa důvodů Aplikace umožní administrátorovi spravovat nově přidané důvody do databáze.

FR-710 Schválení důvodu Aplikace umožní administrátorovi schválit důvod a tím potvrdit jeho zobrazení uživateli a zpřístupnit jej ostatním zkušeným studentům k použití.

FR-720 Odmítnutí důvodu Aplikace umožní administrátorovi odmítnout důvod a tím jej odstranit z databáze.

FR-730 Sjednocení důvodů Aplikace umožní administrátorovi sjednotit důvod s již existujícím důvodem.

FR-740 Přidání důvodu Aplikace umožní administrátorovi přidat nový důvod.

6.7.3 Implementace

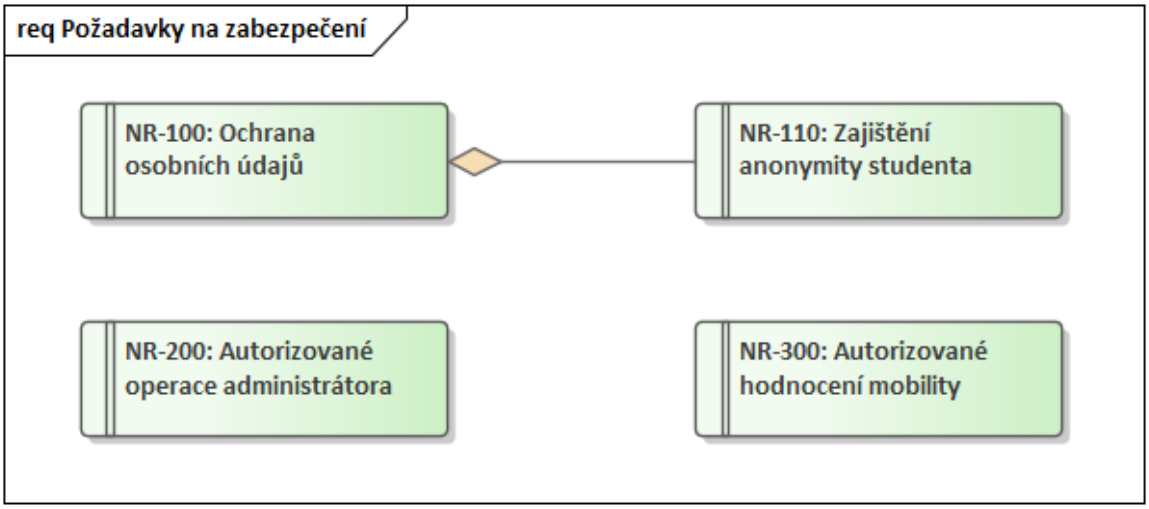
Administrátor má k dispozici okno s formuláři, kde může manipulovat s důvody rozpojení. Důvody může spojovat, přidávat nebo mazat a především potvrzovat jako platné. Jakmile je důvod potvrzený, nelze s ním již manipulovat, aby nebyla narušena návaznost dat s párováními.

6.8 Nefunkční požadavky

Nefunkční požadavky jsou rozděleny do dvou kategorií: požadavky na zabezpečení a na kvalitu software.

6.8.1 Požadavky na zabezpečení

Požadavky na zabezpečení jsou znázorněny na obrázku 6.17.



Obrázek 6.17 Diagram požadavků na zabezpečení (vlastní zpracování)

NR-100: Ochrana osobních údajů Funkce a implementace aplikace budou v souladu se zákonem o ochraně osobních údajů.

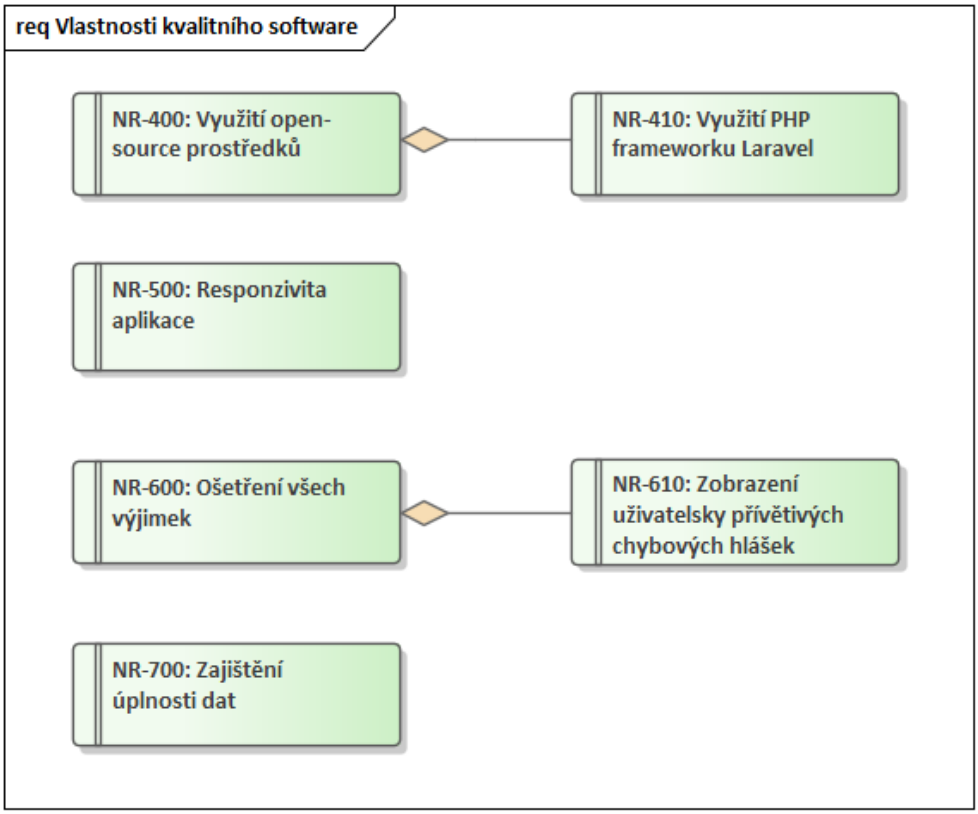
NR-110: Zajištění anonymity studenta Data o mobilitách uložená v databázi budou obsahovat pouze zašifrované UTBid.

NR-200: Autorizované operace administrátora Přístup administrátora bude ověřován heslem a veškeré funkce budou zabezpečeny.

NR-300: Autorizované hodnocení mobility Přístup k hodnocení výjezdu má pouze student, který se mobility zúčastnil.

6.8.2 Požadavky na kvalitu software

Požadavky na kvalitu software jsou znázorněny na obrázku 6.18.



Obrázek 6.18 Diagram požadavků na kvalitu (vlastní zpracování)

NR-400: Využití open-source prostředků Aplikace bude pro svůj vývoj používat pouze open-source prostředky.

NR-410: Využití PHP frameworku Laravel Aplikace bude vyvíjena ve frameworku Laravel.

NR-500: Responsivita Vzhled aplikace bude přizpůsobe zobrazování na mobilních zařízení

NR-600: Ošetření všech výjimek Systém ošetří výjimky tak, aby aplikace uživateli neselhala.

NR-610: Zobrazení uživatelsky přívětivých chybových hlášek Systém zpracuje výjimky takovým způsobem, aby uživatel chápal jejich význam.

NR-700: Zajištění úplnosti dat Aplikace poskytne prostředky pro zajištění úplnosti dat v databázi.

7 EXTERNÍ POŽADAVKY NA UŽIVATELSKÉ ROZHRANÍ

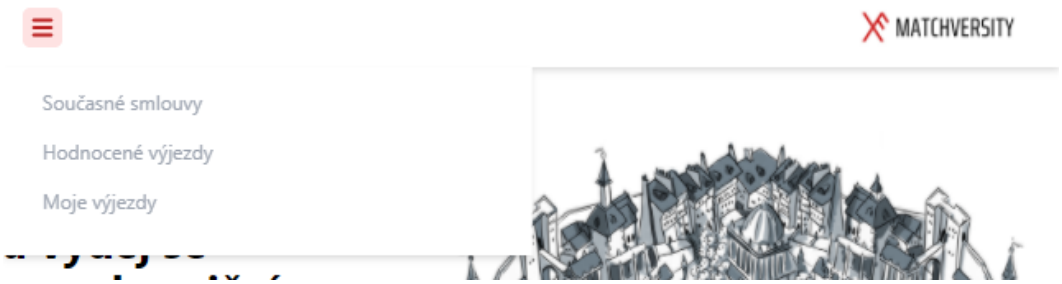
Uživatelské rozhraní aplikace bude jednoduché a srozumitelné, aby poskytovalo uživateli co nejlepší zážitek. Wireframes použité v následujících ukázkách jsou prvotní návrhy a design využitý pro obhajobu práce se bude ještě vylepšovat.

7.1 Header

Záhlaví aplikace bude obsahovat logo, odkazy na externí stránky a na mobility uživatele.

Obrázek 7.1 znázorňuje záhlaví aplikace pro zařízení s větším formátem obrazovky a na obrázku 7.2 je záhlaví přizpůsobeno menším obrazovkám.

Obrázek 7.1 Design pro header (vlastní zpracování)

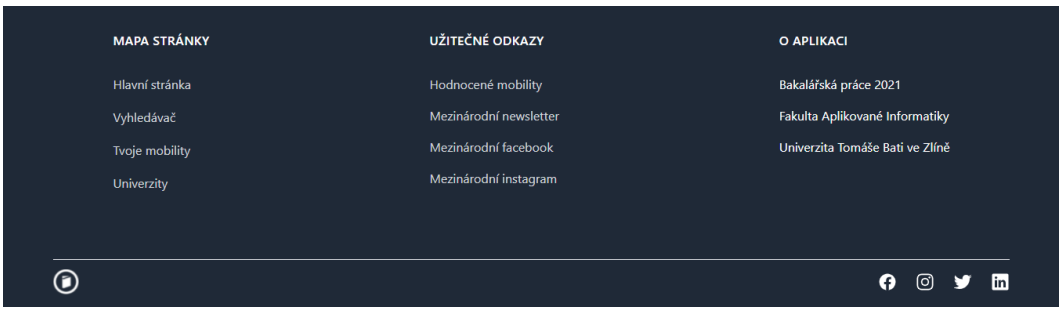


Obrázek 7.2 Design pro header pro menší zařízení (vlastní zpracování)

7.2 Footer

Zápatí bude obsahovat: logo UTB, odkazy na sociální média.

Na obrázku 7.3 je ukázána implementace designu pro zápatí aplikace.



Obrázek 7.3 Design pro footer (vlastní zpracování)

7.3 Hlavní stránka

Hlavní stránka bude obsahovat popis aplikace a navigaci na vyhledávání zahraničních univerzit. Wireframe (Obr. 7.4) zobrazuje prvotní návrh úvodní stránky a na následujícím obrázku (Obr. 7.5) je znázorněn implementovaný design.

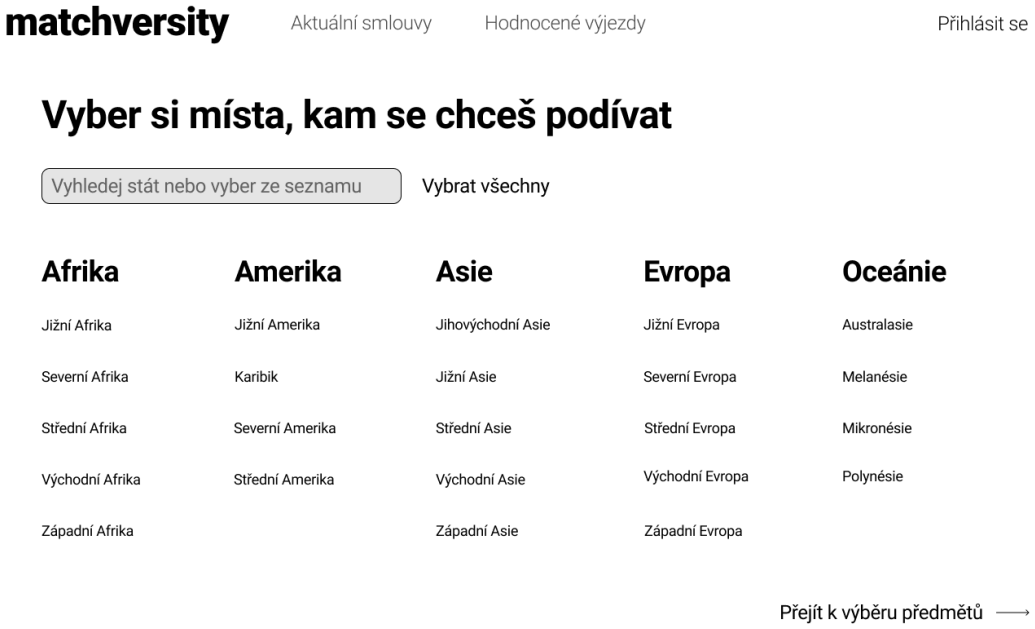
Obrázek 7.4 Wireframe hlavní stránky (vlastní zpracování)

Obrázek a logo použité v designu (Obr. 7.5) jsou návrhy MgA. Aliaxandry Laurove, která pomáhá s ilustracemi projektu.

Obrázek 7.5 Design hlavní stránky (vlastní zpracování)

7.4 Vyhledávač

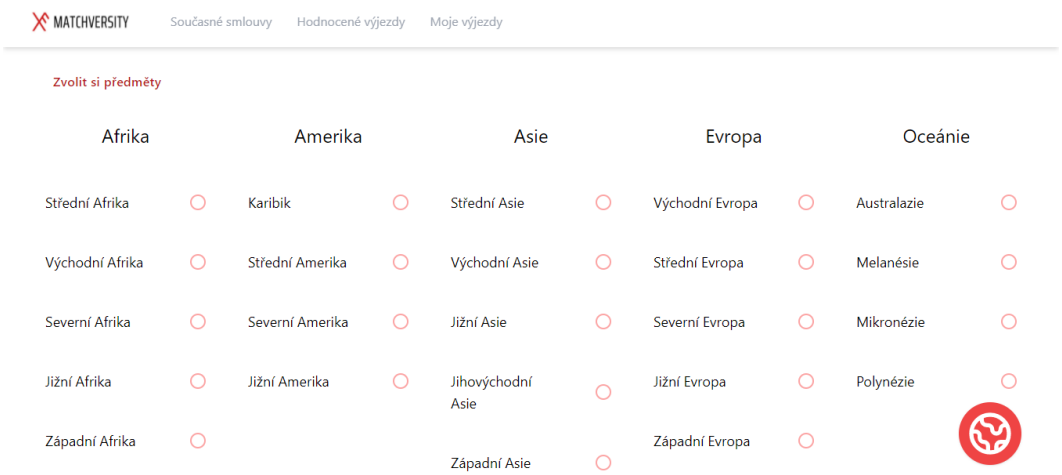
Stránka s vyhledávačem bude umožňovat zadat základní informace o studiu. V pokročilém vyhledávání bude umožněno volit země výjezdu a předměty ke spárování. Podle prvotního návrhu (Obr. 7.6) byl implementován výsledný design na obrázku 7.7.



Obrázek 7.6 Wireframe vyhledávače zemí (vlastní zpracování)

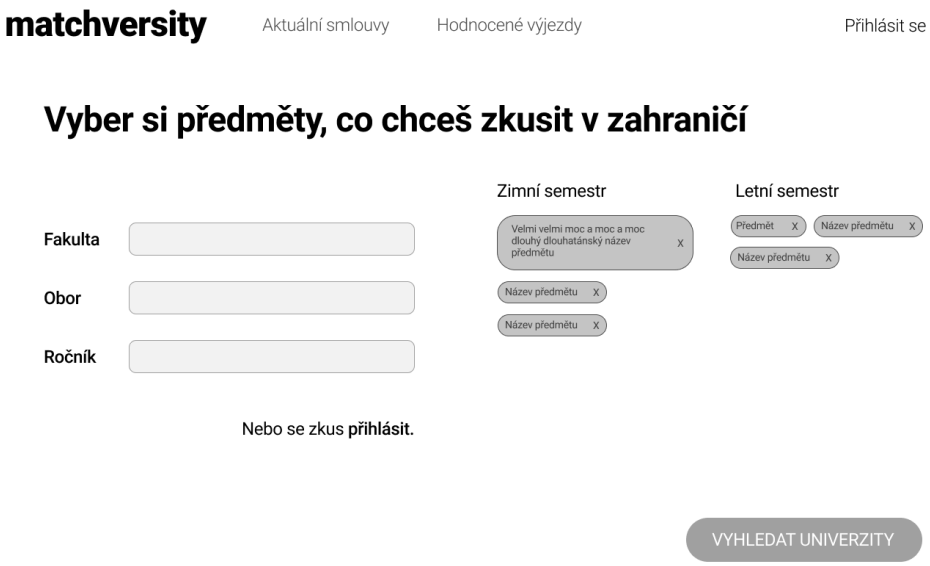
Design výběru zemí je rozdělen do jednotlivých kontejnerů podle kontinentů, které obsahují selecty s regiony, které volí všechny země v regionu. Po kliknutí na jeho název so zobrazí dropdown select s jeho zeměmi, které lze vybírat jednotlivě.

Návrh tohoto designu je nicméně závislý na zpětné vazbě uživatelů a bude se hledat nejvhodnější možnost, jak jednoduše vybírat velké množství zemí.



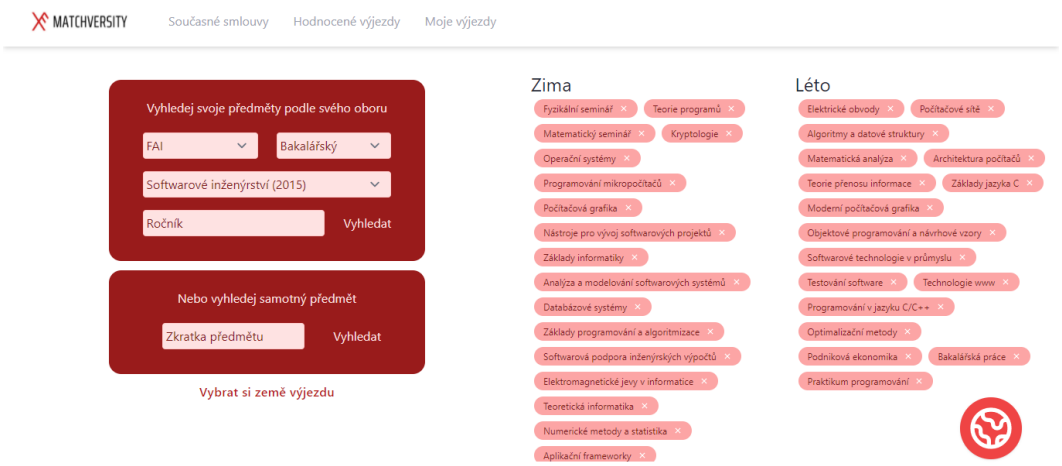
Obrázek 7.7 Design vyhledávače zemí (vlastní zpracování)

Vyhledávání kurzů muselo být omezeno o přihlašování. Prvotní návrh (Obr. 7.8) pracoval s možností přihlášení studenta, díky kterému by se z API STAG získala data o neabsolvovaných předmětech studenta a automatické načtení těchto předmětů. Tento dotaz na data ovšem vyžadoval autentizaci uživatele pro STAG, čímž by bylo nutné zavést dvojí přihlášení.



Obrázek 7.8 Wireframe vyhledávače kurzů (vlastní zpracování)

Z tohoto důvodu bylo místo přihlášení navrženo vyhledávání samostatných předmětů, které pomůže více přizpůsobit výsledky potřebám studenta. Následující obrázek 7.9 znázorňuje implementaci návrhu.



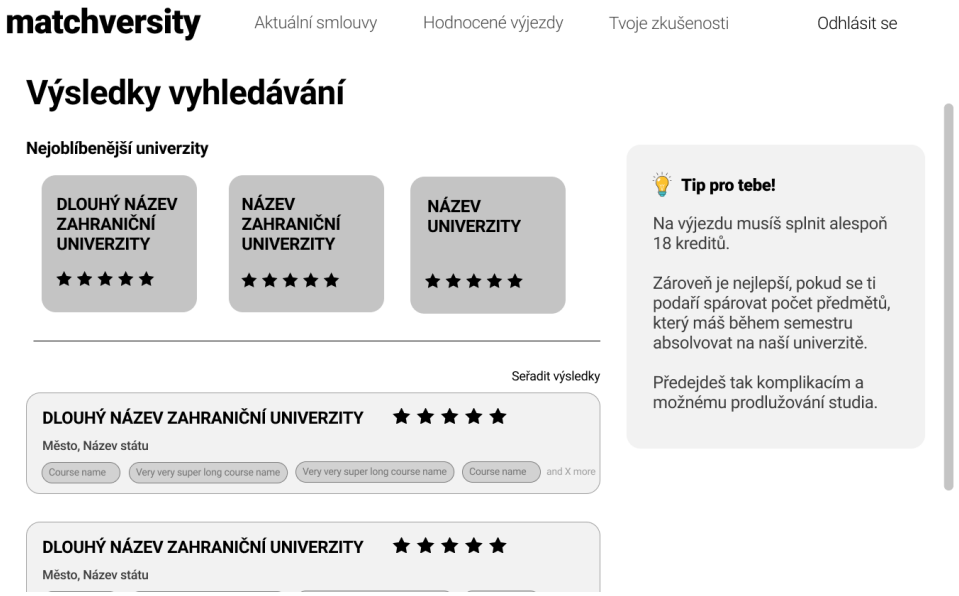
Obrázek 7.9 Design vyhledávače kurzů (vlastní zpracování)

Polovina stránky obsahuje rozdělení do dvou semestrů, které jsou skrolovatelné. Jednotlivé předměty lze kliknutím vymazat.

7.5 Výsledky vyhledávání

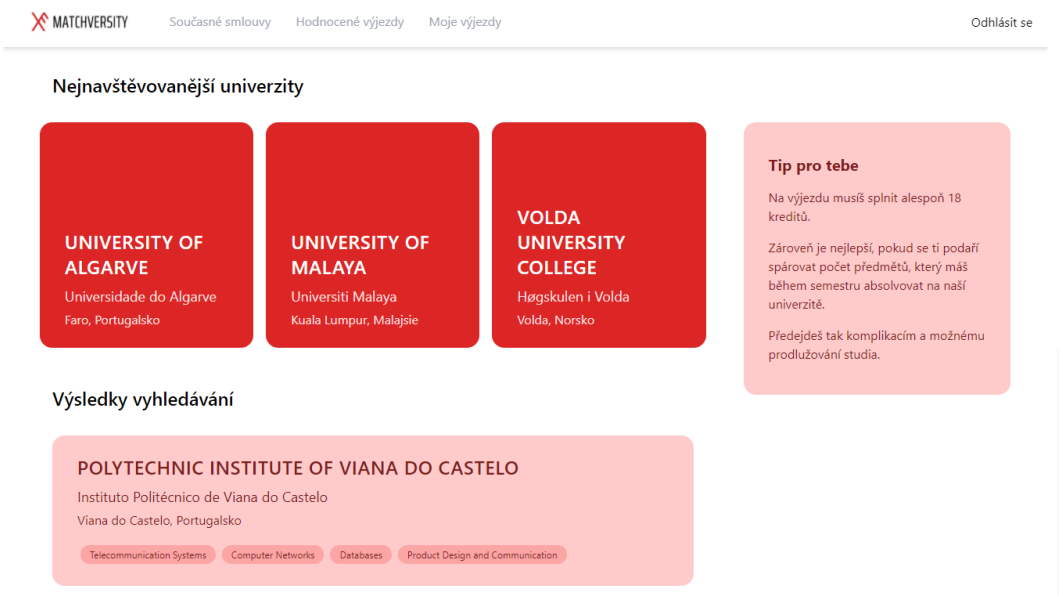
Stránka s výsledky bude obsahovat tři nejoblíbenější výjezdy a ostatní shody seřazené podle počtu mobilit.

Prvotní návrh stránky znázorňuje obrázek 7.10.



Obrázek 7.10 Wireframe výsledků (vlastní zpracování)

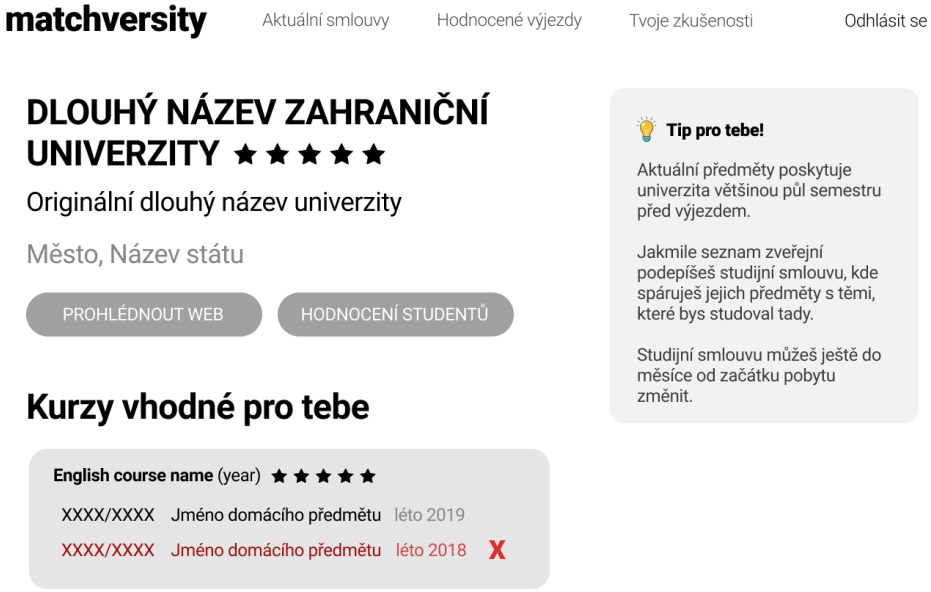
Obrázek 7.11 zobrazuje implementaci vyhledaných výsledků. Konečný design neobsahuje hodnocení univerzity, z důvodů zmíněných v kapitole 5.4.



Obrázek 7.11 Design výsledků (vlastní zpracování)

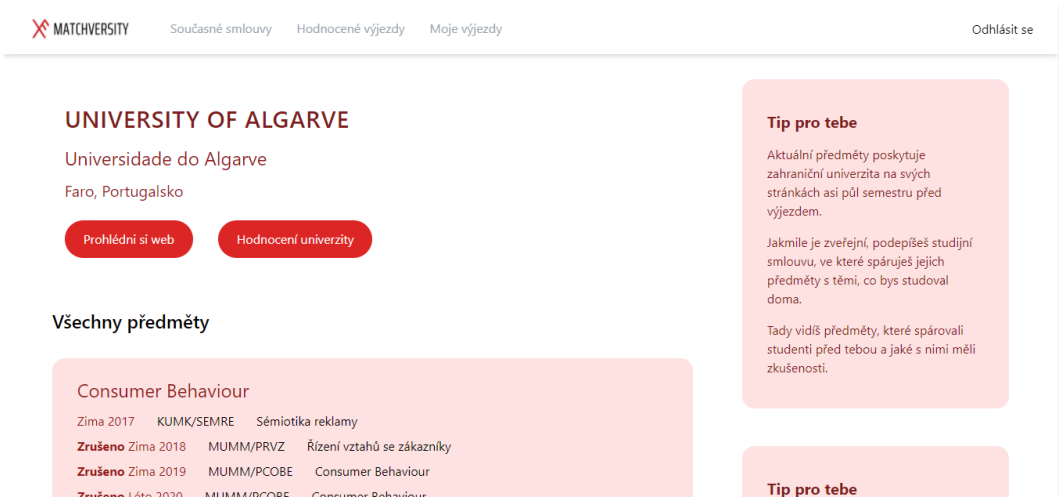
7.6 Profil univerzity

Při zobrazení univerzity bude k dispozici její název, hodnocení na xchange a list spárovaných předmětů. Tento wireframe je ukázán na obrázku 7.12.



Obrázek 7.12 Wireframe profilu univerzity (vlastní zpracování)

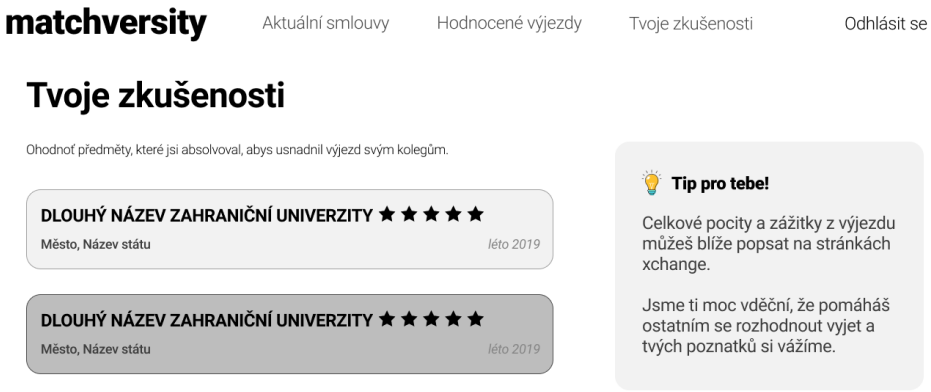
Design této stránky (Obr. 7.13) patří ke složitějším návrhům a je třeba na něm ještě zapracovat, aby poskytl co nejlepší zážitek uživateli. Jedná se především o znázornění zrušených předmětů a velkého množství umístěných dat, které se stávají nepřehledné.



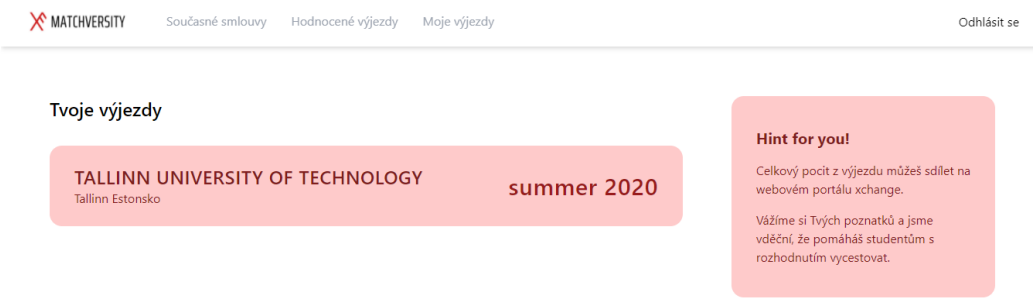
Obrázek 7.13 Design profilu univerzity (vlastní zpracování)

7.7 Výjezdy studenta

Student, který absolvoval výjezdy, bude mít k dispozici seznam svých výjezdů. Wireframe (Obr. 7.14) a design (Obr. 7.15) tohoto požadavku je znázorněn na obrázcích níže.



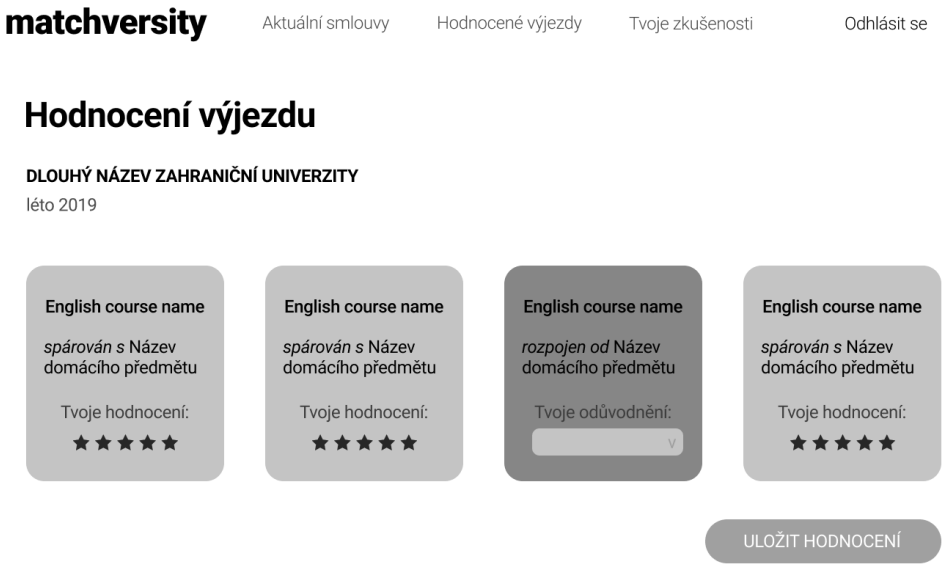
Obrázek 7.14 Wireframe výjezdů studenta (vlastní zpracování)



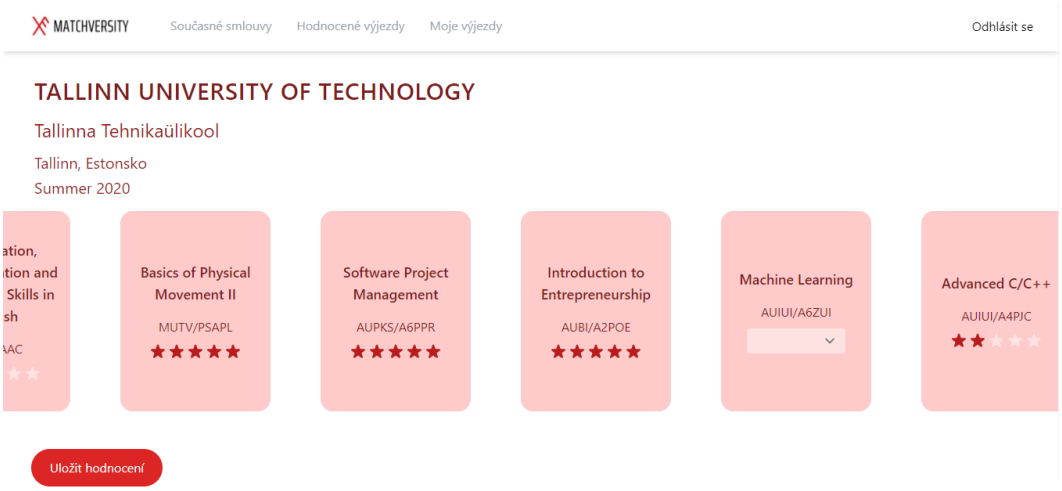
Obrázek 7.15 Design výjezdů studenta (vlastní zpracování)

7.8 Hodnocení výjezdu

Pro studenta, který má výjezd již za sebou, bude k dispozici formulář pro hodnocení kvality spárovaných předmětů. Tento wireframe (Obr. 7.16) byl implementován jako design na obrázku 7.17.



Obrázek 7.16 Wireframe hodnocení výjezdu (vlastní zpracování)



Obrázek 7.17 Design hodnocení výjezdu (vlastní zpracování)

8 IMPLEMENTACE ZABEZPEČENÍ APLIKACE

V první řadě má aplikace zabezpečení směrování požadavků (routes) pomocí middlewaru, který v případě nutné autentizace přesměruje uživatele. To je implementováno jak u administrátorské části (Kód 8.1), tak u uživatelské části, která přistupuje k mobilitám uživatele (Kód 8.2).

```
Route::middleware('auth:admin')->prefix('admin')
     ->name('admin.')->group(function() {
  Route::get('/', [AdminMobilityController::class, 'index']);

  Route::resource('foreign-courses', ForeignCourseController::class)
  ->only(['index', 'update']);

  Route::resource('home-courses', HomeCourseController::class)
  ->only(['index', 'update']);

  Route::post('import', [AdminMobilityController::class, 'import'])
  ->name('mobilities.import');
  Route::resource('mobilities', AdminMobilityController::class)
  ->except(['create', 'edit', 'delete']);

  Route::resource('reasons', ReasonController::class)
  ->except(['show', 'edit', 'create']);

  Route::resource('universities', AdminUniversityController::class)
  ->only(['index', 'update', 'edit']);
});

```

Kód 8.1 Administrátorská část(vlastní zpracování)

```
Route::middleware('auth:web')
   ->resource('mobilities', MobilityController::class)
   ->only(['index', 'show', 'edit', 'update']);

```

Kód 8.2 Studentská část(vlastní zpracování)

Všechny HTTP požadavky vyžadující změnu v modelové části aplikace, mají vytvořenou vlastní třídu, která je potomkem `FormRequest` a dotazy jsou tu autorizovány a následně validovány. Příklad autorizace administrátora ukazuje následující kód 8.3.

```
public function authorize()
{
  return Auth::guard('admin')->check();
}

```

Kód 8.3 Autorizace administrátora (vlastní zpracování)

Autorizace studenta (Kód 8.4) je nutná v případě, kdy je zaslán požadavek na změnu hodnocení mobility `UpdateMobilityRequest`.

```
1   public function authorize()
2   {
3     $mobility = Mobility::find($this->route('mobility'));
4     return $mobility->user_id === $this->user()->id;
5
6   }
7
```

Kód 8.4 Autorizace studenta (vlastní zpracování)

Příklad kódu 8.5 obsahuje validaci formuláře z requestu vyhledávání `SearchRequest`, která kontroluje, zda obsahuje správné hodnoty pro země a kurzy.

```
1     public function rules()
2   {
3     return [
4       'countries' => ['nullable', 'array'],
5       'countires.*' => ['alpha', 'size:2', 'string'],
6       'courses' => ['nullable', 'array'],
7       'courses.*' => ['string']
8     ];
9   }
10
```

Kód 8.5 Validace požadavku (vlastní zpracování)

K ochraně proti CSRF se využívá tokenů ve formulářích a proti SQL injections jsou využity chráněné metody Eloquent Modelů.

Nejdůležitější ochranou je však samotná interpretace dat. Data o mobilitách jsou přiřazeny k uživatelům pomocí utbID, což je identifikátor využívaný systémem STAG. Tento údaj není veřejný a je využíván pouze jako soukromý klíč v systému, takže ani samotný student k tomuto ID nemá přístup. I kdyby tedy data o mobilitách s utbID unikla, jejich obsah by dokázal přidělit studentům pouze ten, který by rozluštil tento identifikátor, a jelikož je informace ještě zahashovaná, znamená to, že je ochrana dvojnásobná. O uživatelích nejsou ukládané žádné jiné citlivé údaje.

9 NÁVRH DALŠÍHO VÝVOJE

Mezi funkce k implementaci patří již zmíněné propojení s portálem xchange, které posílí informace poskytnuté aplikací, a také propojení s shibboleth, které usnadní autentizaci uživatele.

Dále je třeba usilovně pracovat na designu a přidat animace. Jakmile budou návrhy odsouhlaseny mezinárodním oddělením, budou se pracovat na vylepšení celkového uživatelského rozhraní.

Po provedení beta testování bude uskutečněn průzkum, který by mohl odhalit další možné požadavky ze strany studentů.

Jako další úkol bude přizpůsobit aplikaci pro studenty anglických oborů.

Aplikace je nicméně otevřená vylepšení. V současné chvíli není možné, aby obsahovala aktuální seznam kurzů všech univerzit, protože každá instituce používá odlišný způsob zveřejnění předmětů a v různé období. Toto by šlo nicméně v budoucnosti změnit a software by se tak mohl stát autonomním.

Funkce aplikace by také mohla sloužit všem univerzitám, kde by šlo pouze o menší kosmetické úpravy databáze a rozšíření přihlašování a administrátorské části uživatelského rozhraní, které by umožnilo jednotlivým univerzitám spravovat vlastní profily i seznamy předmětů a poskytly tak studentům nejlepší prostředí pro výběr výjezdu.

ZÁVĚR

Bakalářská práce se zabývala vývojem webové aplikace pro mezinárodní oddělení UTB. Projekt byl implementován za pomocí frameworku Laravel, Vue.js a Tailwind CSS.

Hlavním cílem bylo vytvořit funkční prototyp, který bude obsahovat hlavní funkce aplikace a bude postupně vylepšován pro nasazení na server UTB.

Teoretická část se v úvodu věnuje rešerši technologií webových aplikací. Poté je rozebrán framework Laravel a jeho hlavní funkce, kde jsou rozebrány i metody dostupné pro zabezpečení aplikace. Dále je teorie doplněna stručnou rešerší frameworku Vue.js a Tailwind.

Praktická část se zabývá sběrem požadavků k implementaci aplikace. Požadavky byly diskutovány s mezinárodním oddělením. V úvodní části je popsána struktura databáze a jejích tabulek. Následuje analýza případů užití, které jsou doplněny o funkční požadavky na aplikaci, které jsou rozšířeny o popis a/nebo ukázku implementace. Dále je popsán návrh a provedení designu a nakonec popis realizovaného zabezpečení aplikace.

Po dokončení analýzy požadavků byla pozornost věnována vývoji backendu. Následně byl k frameworku připojen Vue.js, který rozšířil funkce frontendu. Po domluvě spolupráce s MgA. Aliaxandrou Laurovou byla zahájena práce na designu, která byla později rozšířena o skicy jejích ilustrací pro větší reprezentativnost, kde byla aplikace rozšířena o funkce frameworku Tailwind CSS pomocí inicializace popsané v rešerši. Během tohoto procesu probíhala rešerše frameworků z důvodu počátečních nulových zkušeností autorky s těmito technologiemi.

Aplikace byla poté prezentována na soutěži STOČ, kde obstála třetí místo v kategorii Informačních technologií.

V rámci předmětu Testování software je aplikace testována za pomocí frameworku Roboto.

Kvůli obsáhlosti projektu nebylo možné zcela práci dokončit k připravenosti na nasazení na server. V závěru praktické části jsou zmíněny body, kterým bude věnována pozornost po obhajobě projektu.

Pro snadnou udržitelnost aplikace byly zvoleny funkce frameworků, které zvětšují přehlednost kódu, který byl pravidelně refaktorován a komentován.

SEZNAM POUŽITÉ LITERATURY

[1] TONCROVÁ, Hana. *Statistiky* [e-mailová komunikace]. 28. dubna 2021 12:47 [cit. 2021-5-6].

[2] Client-Server Model. *GeeksforGeeks: A computer science portal for geeks* [online]. Uttar Pradesh: GeeksforGeeks, b. r., 15 Nov, 2019n. l. [cit. 2021-4-25]. Dostupné z: https://www.geeksforgeeks.org/client-server-model

[3] CHRISTENSSON, P. Client-Server Model Diagram. *TechTerms: The Computer Dictionary* [online]. Minneapolis: Sharpened Productions, c2021, June 17, 2016 [cit. 2021-4-25]. Dostupné z: https://techterms.com/definition/client-server_model

[4] CHRISTENSSON, P. Backend definition. *TechTerms: The Computer Dictionary* [online]. Minneapolis: Sharpened Productions, c2021, April 11, 2020 [cit. 2021-4-25]. Dostupné z: https://techterms.com/definition/backend

[5] What can PHP do? *PHP: HyperText Preprocessor* [online]. The PHP Group, c2001-2021 [cit. 2021-4-25]. Dostupné z: https://www.php.net/manual/en/intro-whatcando.php

[6] Relational Database (RDB). Techopedia™ [online]. Edmonton: Techopedia, c2021 [cit. 2021-4-25]. Dostupné z: https://www.techopedia.com/definition/1234/relational-database-rdb

[7] HTML. *Computer Hope* [online]. Riverton: Computer Hope, c2021, 2. 1. 2021 [cit. 2021-4-26]. Dostupné z: https://www.computerhope.com/jargon/h/html.htm

[8] A brief SGML tutorial. *W3C* [online]. Cambridge (Massachuttes): W3C, c2021 [cit. 2021-4-26]. Dostupné z: https://www.w3.org/TR/WD-html40-970708/intro/sgmltut.html

[9] BEAL, Vangie. What Is HTML?: HTML Meaning & Definition. *Webopedia* [online]. Nashville, TN: TechnologyAdvice, c2021, April 7, 2021 [cit. 2021-4-26]. Dostupné z: https://www.webopedia.com/definitions/html

[10] SURGUY, Maksim. History of Laravel PHP framework, Eloquence emerging. *Maxoffsky* [online]. Seattle: Maxoffsky, b. r., JULY 27, 2013 [cit. 2021-4-26]. Dostupné z: https://maxoffsky.com/code-blog/history-of-laravel-php-framework-eloquence-emerging

[11] RATHORE, Deven. Brief Overview Of Design Patterns Used in Laravel: The Builder Pattern. *Dunebook.com* [online]. Dunebook.com, c2015-2021, August 21, 2016 [cit. 2021-4-26]. Dostupné z: https://www.dunebook.com/brief-overview-of-design-patterns-used-in-laravel

[12] RATHORE, Deven. Brief Overview Of Design Patterns Used in Laravel: The Factory pattern. *Dunebook.com* [online]. Dunebook.com, c2015-2021, August 21, 2016 [cit. 2021-4-26]. Dostupné z: https://www.dunebook.com/brief-overview-of-design-patterns-used-in-laravel/2

[13] RATHORE, Deven. Brief Overview Of Design Patterns Used in Laravel: The Repository pattern. *Dunebook.com* [online]. Dunebook.com, c2015-2021, August 21, 2016 [cit. 2021-4-26]. Dostupné z: https://www.dunebook.com/brief-overview-of-design-patterns-used-in-laravel/3

[14] RATHORE, Deven. Brief Overview Of Design Patterns Used in Laravel: The Strategy pattern. *Dunebook.com* [online]. Dunebook.com, c2015-2021, August 21, 2016 [cit. 2021-4-26]. Dostupné z: https://www.dunebook.com/brief-overview-of-design-patterns-used-in-laravel/4

[15] RATHORE, Deven. Brief Overview Of Design Patterns Used in Laravel: The Provider pattern. *Dunebook.com* [online]. Dunebook.com, c2015-2021, August 21, 2016 [cit. 2021-4-26]. Dostupné z: https://www.dunebook.com/brief-overview-of-design-patterns-used-in-laravel/5

[16] RATHORE, Deven. Brief Overview Of Design Patterns Used in Laravel: The Facade pattern. *Dunebook.com* [online]. Dunebook.com, c2015-2021, August 21, 2016 [cit. 2021-4-26]. Dostupné z: https://www.dunebook.com/brief-overview-of-design-patterns-used-in-laravel/6

[17] TORSTEN, Ruger. *Model View Controller: theory and practice.* Medium: Where good ideas find you [online]. A Medium Corporation, b. r., Aug 22, 2017 [cit. 2021-5-8]. Dostupné z: https://medium.com/rubydesign_web/model-view-controller-theory-and-practice-e5db514c18d9

[18] Artisan Console. *Laravel: The PHP Framework for Web Artisans* [online]. Laravel, c2011-2021 [cit. 2021-4-26]. Dostupné z: https://laravel.com/docs/8.x/artisan

[19] Concepts. *Webpack* [online]. MIT, b. r. [cit. 2021-4-26]. Dostupné z: https://webpack.js.org/concepts

[20] Compiling Assets (Mix). *Laravel: The PHP Framework for Web Artisans* [online]. Laravel, c2011-2021 [cit. 2021-4-26]. Dostupné z: https://laravel.com/docs/8.x/mix

[21] Directory Structure. *Laravel: The PHP Framework for Web Artisans* [online]. Laravel, c2011-2021 [cit. 2021-4-26]. Dostupné z: https://laravel.com/docs/8.x/structure

[22] Database: Getting Started. *Laravel: The PHP Framework for Web Artisans* [online]. Laravel, c2011-2021 [cit. 2021-4-26]. Dostupné z: https://laravel.com/docs/8.x/database

[23] Database Testing. *Laravel: The PHP Framework for Web Artisans* [online]. Laravel, c2011-2021 [cit. 2021-4-26]. Dostupné z: https://laravel.com/docs/8.x/database-testing

[24] Database: Migrations. *Laravel: The PHP Framework for Web Artisans* [online]. Laravel, c2011-2021 [cit. 2021-4-26]. Dostupné z: https://laravel.com/docs/8.x/migrations

[25] Database: Seeding. *Laravel: The PHP Framework for Web Artisans* [online]. Laravel, c2011-2021 [cit. 2021-4-26]. Dostupné z: https://laravel.com/docs/8.x/seeding

[26] Blade Templates. *Laravel: The PHP Framework for Web Artisans* [online]. Laravel, c2011-2021 [cit. 2021-4-26]. Dostupné z: https://laravel.com/docs/8.x/blade

[27] Localization. *Laravel: The PHP Framework for Web Artisans* [online]. Laravel, c2011-2021 [cit. 2021-4-26]. Dostupné z: https://laravel.com/docs/8.x/localization

[28] Routing. *Laravel: The PHP Framework for Web Artisans* [online]. Laravel, c2011-2021 [cit. 2021-4-26]. Dostupné z: https://laravel.com/docs/8.x/routing

[29] Controllers. *Laravel: The PHP Framework for Web Artisans* [online]. Laravel, c2011-2021 [cit. 2021-4-26]. Dostupné z: https://laravel.com/docs/8.x/controllers

[30] Middleware. *Laravel: The PHP Framework for Web Artisans* [online]. Laravel, c2011-2021 [cit. 2021-4-27]. Dostupné z: https://laravel.com/docs/8.x/middleware

[31] HTTP Requests. *Laravel: The PHP Framework for Web Artisans* [online]. Laravel, c2011-2021 [cit. 2021-4-27]. Dostupné z: https://laravel.com/docs/8.x/requests

[32] Request Lifecycle. *Laravel: The PHP Framework for Web Artisans* [online]. Laravel, c2011-2021 [cit. 2021-4-27]. Dostupné z: https://laravel.com/docs/8.x/lifecycle

[33] Eloquent: Getting Started. *Laravel: The PHP Framework for Web Artisans* [online]. Laravel, c2011-2021 [cit. 2021-4-27]. Dostupné z: https://laravel.com/docs/8.x/eloquent

[34] Database: Query Builder. *Laravel: The PHP Framework for Web Artisans* [online]. Laravel, c2011-2021 [cit. 2021-4-27]. Dostupné z: https://laravel.com/docs/8.x/queries

[35] Service Providers. *Laravel: The PHP Framework for Web Artisans* [online]. Laravel, c2011-2021 [cit. 2021-4-27]. Dostupné z: https://laravel.com/docs/8.x/providers

[36] Authentication. *Laravel: The PHP Framework for Web Artisans* [online]. Laravel, c2011-2021 [cit. 2021-4-27]. Dostupné z: https://laravel.com/docs/8.x/authentication

[37] Authorization. *Laravel: The PHP Framework for Web Artisans* [online]. Laravel, c2011-2021 [cit. 2021-4-27]. Dostupné z: https://laravel.com/docs/8.x/authorization

[38] CSRF Protection. *Laravel: The PHP Framework for Web Artisans* [online]. Laravel, c2011-2021 [cit. 2021-4-27]. Dostupné z: https://laravel.com/docs/8.x/csrf

[39] Hashing. *Laravel: The PHP Framework for Web Artisans* [online]. Laravel, c2011-2021 [cit. 2021-4-27]. Dostupné z: https://laravel.com/docs/8.x/hashing

[40] Validation. *Laravel: The PHP Framework for Web Artisans* [online]. Laravel, c2011-2021 [cit. 2021-4-27]. Dostupné z: https://laravel.com/docs/8.x/validation

[41] Introduction. *Vue.js* [online]. Cambridge (Massachuttes): Evan You, c2014-2021, 15. 1. 2021 [cit. 2021-4-27]. Dostupné z: https://v3.vuejs.org/guide/introduction.html

[42] Installation. Tailwind CSS [online]. Cambridge (Massachuttes): Tailwind CSS, b. r. [cit. 2021-4-27]. Dostupné z: https://tailwindcss.com/docs/installation

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

API	Application Programming Interface
ASP	Active Server Page
CD	Compact disc
CSS	Cascading Style Sheet
CSRF	Cross-site Request Forgery
DTD	Document Type Definition
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
ID	Identification Data
JSON	JavaScript Object Notation
PC	Personal Computer
PHP	Hypertext Preprocesor
MSSQL	Microsoft SQL Server
MVC	Model-View-Controller
NPM	Node Package Manager
SHA	Secure Hash Algorithm
SQL	Structured Query Language
SQML	Standard Generalized Markup Language
STAG	Information System Study Agenda
URI	Uniform Resource Identifier
UTB	Univerzita Tomáše Bati
UX	User experience
WWW	World Wide Web
xchange	https://xchange.utb.cz/
XSS	Cross-site scripting

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM KÓDŮ

SEZNAM PŘÍLOH

PŘÍLOHA P I. PŘILOŽENÉ CD

CD přiložené k bakalářské práci obsahuje:

- bakalářskou práci ve formátu PDF,
- zdrojové kódy aplikace,
- příklad souboru k importu.