Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Systém pre správu litostratigrafických dát

Bakalárska práca

2020 Richard Dominik

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Systém pre správu litostratigrafických dát

Bakalárska práca

Študijný program:	Aplikovaná informatika
Študijný odbor:	2511 Aplikovaná informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	Ing. Alexander Šimko, PhD.
Konzultant:	doc. Mgr. Natália Hlavatá Hudáčková, PhD.

Bratislava 2020 Richard Dominik

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Richard Dominik
Študijný program: aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Systém pre správu litostratigrafických dát
System for managing lithostratigraphic data

Anotácia: Litostratigrafické dáta popisujú zloženie geografických formácií. V súčasnosti neexistuje centrálna databáza litostratigrafických dát pre Slovensko. Dáta sú distribuované v množstve publikácii.

Cieľ: Cieľom práce je vytvoriť webovú viacpoužívateľskú aplikáciu na správu litostratigrafických dát podľa požiadaviek Katedry geológie a paleontológie PRIF UK. Hlavnými funkciami aplikácie budú: správa dát o geografických formáciách a podformáciách, ich profiloch, materiáloch, relevantných publikáciách a pod., potvrdzovanie zmien schvaľovateľom, vyhľadávanie dát a export dát do pdf formátu. Funkcie aplikácie budú používateľom sprístupnené na základe používateľských rolí.

Vedúci: Ing. Alexander Šimko, PhD.
Konzultant: doc. Mgr. Natália Hlavatá Hudáčková, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Dátum zadania: 17.09.2019

Dátum schválenia: 07.10.2019 doc. RNDr. Damas Gruska, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie

Na tomto mieste by som sa chcel poďakovať môjmu školiteľovi Ing. Alexandrovi Šimkovi, PhD. za odborné rady a konzultácie počas tvorby a písania tejto bakalárskej práce. Moje poďakovanie patrí aj mojej konzultantke doc. Mgr. Natálii Hlavatej Hudáčkovej, PhD. za jej spoluprácu.

Abstrakt

Práca s litostratigrafickými dátami je nevyhnutná pri realizácii každého litostratigrafického výskumu. V súčasnosti neexistuje centrálna databáza litostratigrafických dát Slovenska. Dáta sú v súčasnosti dostupné iba vo forme publikácií. V bakalárskej práci sa zaoberáme vývojom viacpoužívateľskej webovej aplikácie pre správu litostratigrafických dát od zadefinovania požiadaviek, výberu vhodných technológií až po samotnú realizáciu a opis riešenia. Aplikácia je rozdelená na administračnú časť a časť pre neprihlásených používateľov. Funkcionality aplikácie sú jej používateľom sprístupnené na základe používateľských rolí. Prihláseným používateľom je umožnené vytvárať, editovať, importovať a exportovať litostratigrafické dáta, ktoré podliehajú procesu schvaľovania. Schválené dáta sú následne prístupné aj pre širokú verejnosť, ktorá si tieto dáta môže vyhľadať na základe názvu alebo geologických jednotiek.

Kľúčové slová: Webová aplikácia, Litostratigrafia, Správa dát, Schvaľovanie dát

Abstract

Working with lithostratigraphic data is mandatory when realizing any lithostratigraphic research. As of now there is no central database with lithostratigraphic data in Slovakia. Data are available only in the form of publications. In this Bachelor thesis we aim for developing a multi-user web application for managing lithostratigraphic data from defining requirements, choosing the right technologies to the finished and well-described solution. The application is divided into two parts. The first part of the application is for non-authenticated users and the second one is for administrators. Different features of the application can be accessed by having different user roles. Authenticated users can create, edit, import, and export lithostratigraphic data, which are the subject of the approval process. Approved data are then accessible for the public, who can search through the data by name or geological units.

Keywords: Web Application, Lithostratigraphy, Data management, Data approval

Obsah

Úvod

Skúmanie a práca s litostratigrafickými dátami je súčasťou každodennej náplne práce geovedcov, ktorí sa venujú výskumu vo vedeckej disciplíne s názvom Litostratigrafia. Litostratigrafické dáta popisujú zloženie geografických formácií a vzťahov medzi nimi. V súčasnosti neexistuje centrálna databáza obsahujúca litostratigrafické dáta o geografických formáciách, ktoré sa nachádzajú na území Slovenskej republiky. Tieto dáta sa nachádzajú v množstve publikácií, čo výrazne predlžuje proces hľadania potrebných informácií.

V tejto práci sa venujeme vytvoreniu viacpoužívateľskej webovej aplikácie, ktorej účelom je uchovávanie a zjednodušenie vyhľadávania litostratigrafických dát o geografických formáciách na území Slovenskej republiky. Funkcionality aplikácie sú jej používateľom sprístupnené na základe používateľských rolí a právomoci. Prihláseným používateľom je umožnené vytvárať, editovať, importovať a exportovať litostratigrafické dáta. Dáta aplikácie podliehajú procesu schvaľovania a všetky schválené dáta sú prístupné aj širokej verejnosti, čo výrazne uľahčuje proces hľadania relevantných informácií či už pre laikov, zberateľov, učiteľov geografie alebo ľudí z vedeckej komunity.

V prvej kapitole sa venujeme základným pojmom, požiadavkám na aplikáciu, predbežnému dátovému modelu, ako aj analýze existujúcich riešení, ktoré riešia podobnú problematiku ako táto práca. Pri analýze existujúcich riešení sme sa zamerali na používateľské rozhranie a jeho funkcionality z pohľadu bežného používateľa aplikácie.

V druhej kapitole sa venujeme výberu technológií na základe zadefinovaných požiadaviek na aplikáciu. Pri technológiách si uvedieme aj dôvod ich výberu a funkcionality, ktoré ponúkajú. Pri vybraných technológiách sa pozrieme aj na porovnanie s konkurenčnými technológiami.

V tretej kapitole popisujeme aplikáciu z pohľadu používateľa. Pri tomto popise sa na aplikáciu pozrieme z pohľadu prihlásených, ale aj neprihlásených používateľov. V prípade prihlásených používateľov sa na aplikáciu pozrieme z pohľadu rôznych rolí, ktoré môžu byť používateľom priradené.

Posledná kapitola sa venuje technickému pohľadu na aplikáciu. Predstavíme dátový model a vybrané funkcionality, ktoré aplikácia ponúka. V tejto časti práce popíšeme aj štruktúru kódu a implementačné detaily.

Kapitola 1

Základné pojmy a požiadavky aplikácie

V tejto kapitole sa venujeme základným pojmom akými sú stratigrafia, litostratigrafia, webová aplikácia, ako aj požiadavkám na aplikáciu pre správu litostratigrafických dát, ktorú sme vytvorili. Neskôr sa v tejto kapitole zaoberáme analýzou podobných dostupných riešení a predbežnému dátovému modelu.

1.1 Stratigrafia

Podľa [16] je stratigrafia opisná veda, ktorá sa zaoberá vertikálnymi vzťahmi v horninách. Stratigrafická metóda je jednou z najstarších metód pri práci s geologickými záznamami a dnes sa s ňou môžeme stretnúť napríklad pri vrtnom prieskume, banskej práci, či iných odvetviach geovedy. Stratigrafia sa zaoberá všetkými horninovými telesami, ktoré tvoria zemskú kôru a rozčleňuje ich do mapovateľných jednotiek podľa ich vlastností a atribútov. Podľa zamerania na špecifické vlastnosti horninových telies sa stratigrafia delí na nasledujúce kategórie:

- Litostratigrafia,
- Nekonformitami vymedzené jednotky,
- Biostratigrafia,
- Magnetostratigrafia,
- Iné (neformálne) stratigrafické kategórie (izotopové, mineralogické atď),
- Chronostratigrafia.

V tejto práci sa budeme zaoberať iba litostratigrafiou a jednotkami, s ktorými pracuje.

1.2 Litostratigrafia

Podľa [16] je litostratigrafia disciplína stratigrafie, ktorá sa zameriava na systematické triedenie horninových komplexov na základe ich litologického charakteru (v akých podmienkach a akými fyzikálno-chemickými mechanizmami sa horniny formovali) a stratigrafických vzťahov. Litostratigrafické jednotky sa používajú na vyjadrenie geologického vývoja určitého regiónu a v praxi pri geologickom mapovaní. Základnou litostratigrafickou jednotkou je súvrstvie (formácia), ktoré má atribúty, ako napríklad: meno, opis, hrúbka, geologický vek (v štandardnej globálnej škále), genéza (podmienky vzniku), odkazy na literatúru atď. Litostratigrafia nám teda pomáha pri pochopení geologickej stavby území, prípadne s porovnávaním geologického vývoja s inými časťami sveta.

1.3 Webová aplikácia

Webová aplikácia je softvér, ktorý je distribuovaný pomocou web servera a používatelia k nej pristupujú pomocou webového prehliadača (Google Chrome, Mozilla Firefox, Safari a mnoho ďalších). V porovnaní s desktopovými, alebo mobilnými aplikáciami ponúka mnoho výhod, akými sú napríklad:

- nezávislosť na platforme alebo operačnom systéme zariadenia,
- používateľ pre aktualizovanie aplikácie nemusí manuálne sťahovať aktualizácie alebo bezpečnostné záplaty,
- úspora miesta na úložisku zariadenia používateľa.

Webové aplikácie sa stávajú čoraz populárnejšími a mnoho známych a užitočných nástrojov sa vyvíja práve pre túto platformu. Existuje viacero prístupov vývoja webových aplikácií, ktorým sa v tejto kapitole nebudeme podrobnejšie zaoberať, nakoľko je naším cieľom v tejto časti práce uviesť iba základnú definíciu tohto pojmu.

1.4 Požiadavky na aplikáciu

Po uvedení základných pojmov, ktoré sú dôležité pre túto prácu, bližšie popíšeme požiadavky na aplikáciu. Aplikácia je vyhotovená formou viacpoužívateľskej webovej aplikácie na základe požiadaviek Katedry geológie a paleontológie Prírodovedeckej fakulty Univerzity Komenského v Bratislave, ktorá slúži ako databáza litostratigrafických dát pre Slovensko. Základnými požiadavkami aplikácie sú:

- Spravovanie litostratigrafických dát (statigrafické intervaly, fosílie, litotypy, metódy merania, oblasti, referenčné a typové profily, geologické jednotky, metódy stanovenia veku a formácie).

- Zamykanie dát pri ich editácii.
- Prehľadávanie histórie používateľov, ktorí upravovali dáta formácií.
- Exportovanie dát do formátu pdf.
- Zobrazovanie geografických bodov na interaktívnej mape.
- Importovanie stratigrafických intervalov vo formáte xlsx.
- Potvrdzovanie zmien schvaľovateľom.
- Zobrazovanie predošlých schválených verzií dát počas procesu schvaľovania.
- Vyhľadávanie litostratigrafických dát.
- Vytváranie používateľských kont (bez samoregistrácie).
- Sprístupnenie funkcionalít aplikácie na základe používateľských rolí.

1.4.1 Typy používateľov

Aplikácia je prístupná pre neprihlásených a prihlásených používateľov na základe používateľských rolí. Pre lepší prehľad právomocí prihlásených používateľov slúži tabuľka 1.1.

Tvorca

Táto používateľská rola má najnižšie právomoci. Tvorca môže prezerať a editovať schválené dáta v aplikácii, prípadne vytvárať nové litostratigfrafické dáta, ktoré po vyplnení všetkých potrebných atribútov odošle na schválenie. Tvorcovi je umožnený export formácií vo formáte pdf.

Recenzent

Používateľom s touto rolou je umožnené schvaľovať/zamietať dáta formácií. Recenzent disponuje všetkými právomocami tvorcu.

Administrátor

Administrátor je najvyššia možná právomoc v rámci aplikácie. Administrátor má právomoc pridávať, prezerať, editovať, schvaľovať a vymazávať všetky dáta v aplikácii. Tejto používateľskej role je umožnené vytváranie nových používateľských kont, upravovanie stratigrafických intervalov ako aj hromadné vymazanie dát.

Neprihlásený používateľ

Neprihláseným používateľom je umožnené vyhľadávať a prezerať dáta o schválených formáciách na základe ich názvov alebo geologických jednotiek.

Právomoc	Tvorca	Recenzent	Administrátor
Prezeranie schválených dát	X	X	X
Prezeranie neschválených dát	-	-	X
Vytváranie litostratigrafických dát⋆	X	X	X
Editácia schválených dát⋆	X	X	X
Odosielanie dát na schválenie	X	X	X
Schvaľovanie formácií	-	X	X
Upravovanie stratigrafických intervalov	-	-	X
Vymazanie dát	-	-	X
Hromadné vymazanie dát	-	-	X
PDF export	X	X	X
Pridávanie nových používateľov	-	-	X

⋆ **okrem stratigrafických intervalov ; X povolené ; - zamietnuté**

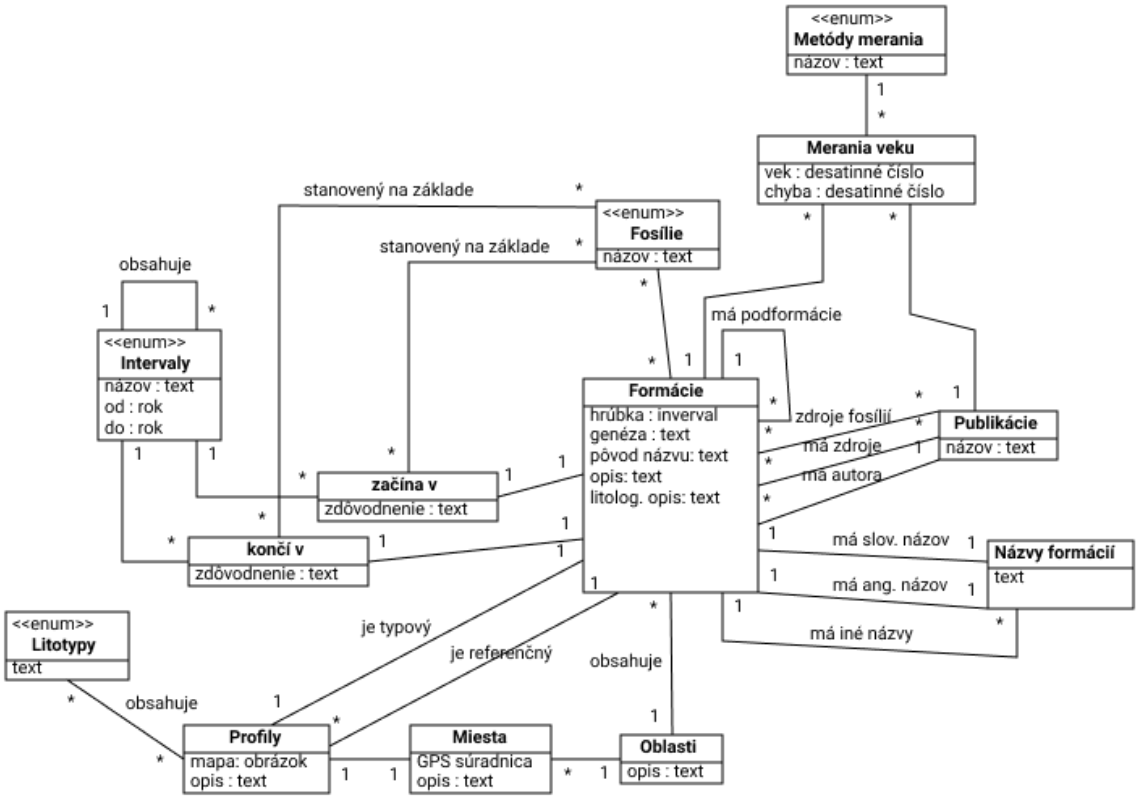
Tabuľka 1.1: Prehľad právomocí prihlásených používateľov podľa rolí

1.4.2 Proces schvaľovania formácií

Každé vytváranie, alebo editácia už existujúcej formácie, musí prejsť procesom schvaľovania recenzentom. Počas doby schvaľovania sú dáta o formácii, rodičovskej formácii a podformáciách zamknuté na úpravu iným používateľom. Recenzent má pri procese schvaľovania vidieť porovnanie s predošlou verziou dát, aby mal prehľad o všetkých zmenách dát, ktoré sa vykonali. Recenzent môže danú formáciu schváliť alebo zamietnuť. Pri zamietnutí recenzent môže uviesť aj dôvod zamietnutia pomocou komentára. Zamietnutú formáciu môže upraviť a poslať na opätovné schválenie iba používateľ, ktorý ju ako posledný odoslal na schválenie.

1.4.3 Predbežný dátový model

Za účelom lepšej špecifikácie všetkých litostratigrafických dát a ich vzťahov, ktoré si systém uchováva, vznikol predbežný dátový model na obrázku 1.1. Predbežný dátový model vytvoril školiteľ v spolupráci s konzultantkou počas stretnutí ohľadom špecifikácie zadania a požiadaviek tejto práce.



«enum» označuje dáta, ktoré sa často nemenia a pri editácii formácie sa vyberajú zo zoznamu

Obr. 1.1: Predbežný dátový model aplikácie

1.5 Existujúce riešenia

V časti 1.4 sme bližšie popísali požiadavky na aplikáciu, na základe ktorých sme sa zamerali na podrobnú analýzu podobných voľne dostupných aplikácií. Našim kritériám najviac vyhovovali aplikácie s názvami Australian Stratigraphic Units Database a Významné paleontologické lokality Slovenska. Analýza týchto aplikácií nám umožnila nazrieť bližšie do sveta geovedy a spôsobu interpretácie informácií, ktoré sú pre členov tejto vedeckej komunity najcennejšie. Táto analýza nám ponúkla aj cenné poznatky o výhodách a nedostatkoch aplikácií, ktoré nám pomôžu nie len pri návrhu používateľského rozhrania, ale aj pri výbere správnych technológií.

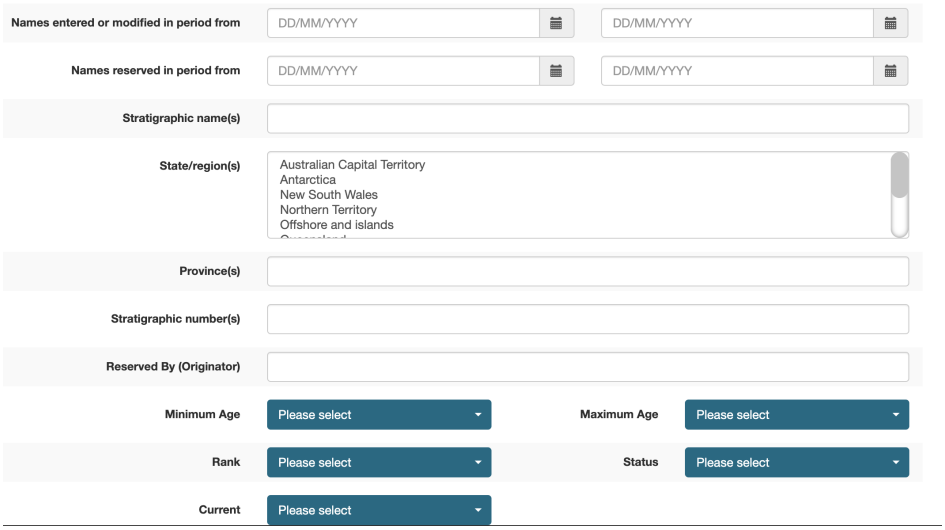
1.5.1 Australian Stratigraphic Units Database

Australian Stratigraphic Units Database [5] je webová aplikácia pod správou spoločenstva s názvom Geosience Australia v spolupráci s Austrálskou geologickou spoločnosťou. Cieľom tejto aplikácie je poskytnúť informácie o stratigrafických jednotkách na území Austrálie a ich použití v odbornej literatúre alebo publikáciách. Táto aplikácia umožňuje nasledujúce funkcionality:

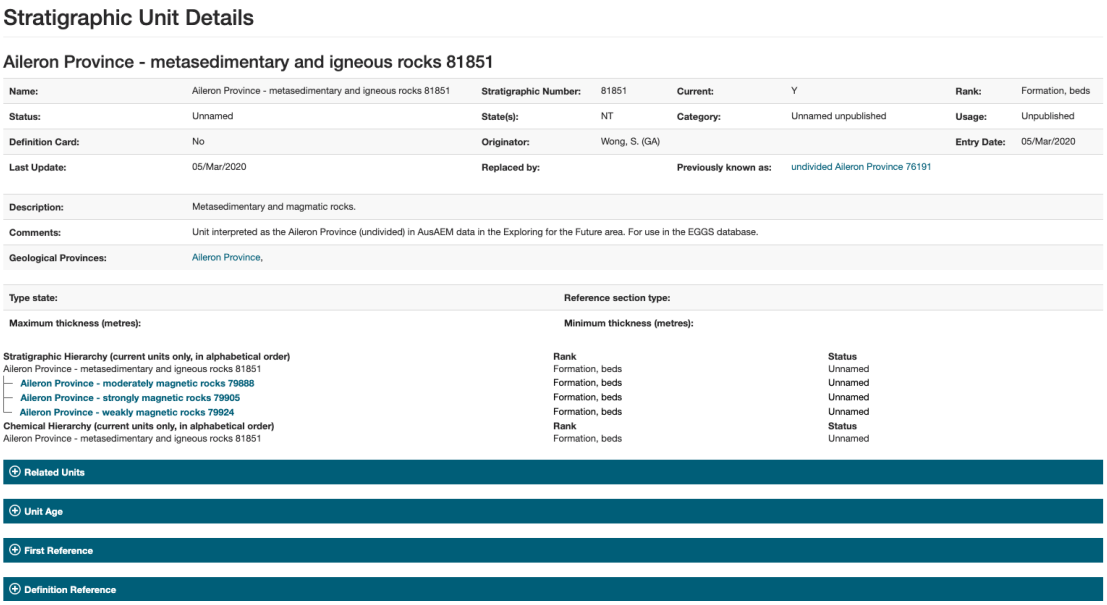
Vyhľadávanie stratigrafických jednotiek

Ukážka používateľského rozhrania pre tento typ vyhľadávania sa nachádza na obrázku 1.2. Jeho cieľom je umožniť vyhľadať stratigrafické jednotky na základe nasledujúcich atribútov:

(a) mená vytvorené alebo upravené v období od/do, (b) mená rezervované v oblasti od/do, (c) stratigrafické meno, (d) štát/región, (e) oblasť, (f) stratigrafické číslo, (g) rezervované osobou, (h) minimálny/maximálny vek, (i) hodnosť, (j) status, (k) aktuálna (áno/nie).



Obr. 1.2: Používateľské rozhranie pre vyhľadávanie stratigrafických jednotiek Australian Stratigraphic Units Database [5]



Obr. 1.3: Používateľské rozhranie pre nájdenú stratigrafickú jednotku Australian Stratigraphic Units Database [5]

Pre vyhľadanie nie je nutné zadať všetky spomenuté atribúty. Po akcii zakliknutia na výsledok tohto vyhľadávania sa používateľom zobrazí detail stratigrafickej jednotky, ktorý môžeme vidieť na obrázku 1.3.

Vyhľadávanie stratigrafických jednotiek na základe štátu alebo územia

Toto vyhľadávanie umožňuje vyhľadať stratigrafické jednotky v zvolenom štáte alebo oblasti a prináša možnosť výberu medzi aktuálnymi a historickými menami jednotiek. Pri tomto type vyhľadávania je možné stiahnutie exportu dát vo formáte txt.

Vyhľadávanie v databáze referencií

Toto vyhľadávanie umožňuje vyhľadať informácie o publikáciách, ktoré citujú litostratigrafické dáta. Stačí ak používateľ vyplní jeden alebo dva atribúty o publikáciách, ktoré nerozlišujú malé a veľké písmená.

Atribúty:

(a) časť názvu, (b) priezvisko jedného z autorov, (c) rok publikovania, (d) celý názov alebo časť publikácie, (e) kľúčové slová, (f) názov/ID mapy 250K, (g) názov/číslo mapy 100K, (h) názov miesta.

Na základe týchto atribútov sa používateľovi zobrazia všetky publikácie, ktoré vyhovujú zadaným kritériám. Pri každej nájdenej publikácii sa nachádza časť stratigrafické mená, v ktorej sa nachádza zoznam stratigrafických jednotiek, ktoré sú v danej publikácii citované.

Rezervácia názvu novej stratigrafickej jednotky

Táto funkcionalita umožňuje používateľovi pred každým pridaním novej stratigrafickej jednotky do aplikácie rezerváciu mena. Takáto rezervácia používateľovi zaručí, že sa v databáze nenachádza stratigrafická jednotka s rovnakým menom. Ďalším používateľom zakáže rezerváciu stratigrafickej jednotky s rovnakým menom.

Návrh referencie

Táto funkcionalita umožňuje používateľovi poukázať na publikácie o stratigrafických jednotkách, ktoré sa v aplikácii ešte nenachádzajú, vyplnením krátkeho formulára so všetkými potrebnými informáciami o publikácii, kontaktom na používateľa a jeho následným odoslaním.

Spätná väzba

Táto funkcionalita umožňuje používateľovi zaslať akékoľvek pripomienky alebo otázky k aplikácii vyplnením formulára.

Zhrnutie

Na základe podrobnej analýzy tejto aplikácie a jej funkcionalít sme prišli k záveru, ktorý sme rozdelili na dve časti. Prvou časťou sú nedostatky a druhou časťou sú výhody.

Nedostatky: Medzi hlavné nedostatky tejto aplikácie patrí viacero možností vyhľadávania v dôsledku čoho je pre každého nového používateľa menej intuitívne, ktoré vyhľadávanie má použiť práve pre jeho potreby. Aby používateľ lepšie pochopil možnostiam vyhľadávania a funkcionalitám aplikácie, má možnosť preštudovať si používateľskú príručku, ktorá bližšie popisuje každú možnosť vyhľadávania a jej účel, alebo možnosť kliknúť na ikonu „i“, ktorá sa nachádza pri názve vyhľadávania a popisuje účel daného vyhľadávania a jeho atribútov. Medzi ďalšie nedostatky aplikácie patrí počet atribútov pri vyhľadávaní a využitie formulárov vo forme tabuľky. Počet atribútov núti používateľa prejsť si celý formulár a nájsť to správne miesto, kde zadať informáciu ktorou disponuje. Pri atribútoch s možnosťou výberu hodnoty považujeme za nedostatok neprístupnosť vyhľadávania v týchto hodnotách a preto používateľ pre výber správnej hodnoty musí prejsť celý zoznam možností. Z pohľadu dát o stratigrafických jednotkách je nevýhodou aplikácie, že sú popísané iba neformálne a nedodržiavajú všetky normy určené na definovanie stratigrafických jednotiek. Zo subjektívneho hľadiska je dizajn používateľského rozhrania menej prívetivý.

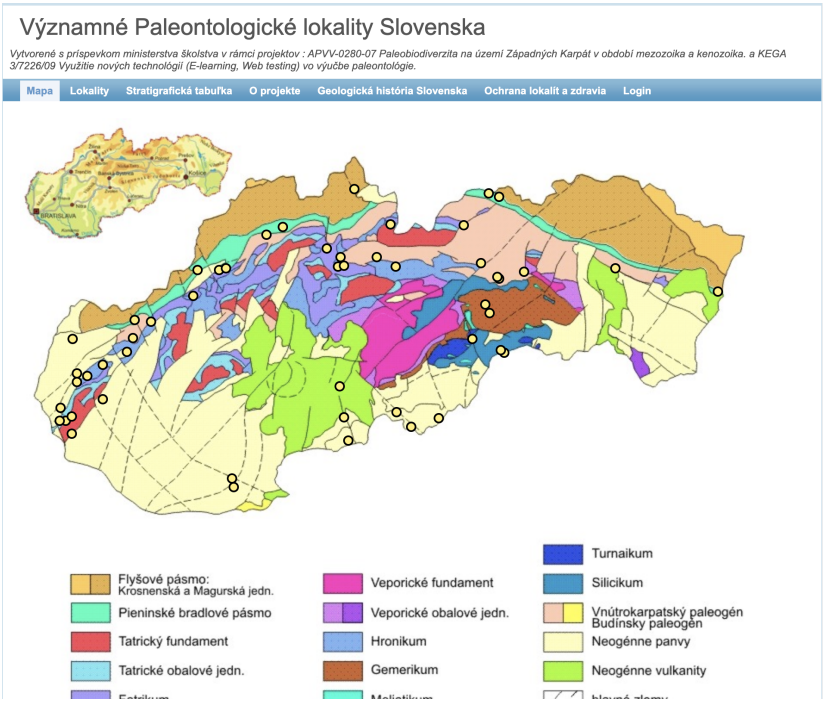
Výhody: Medzi hlavné výhody aplikácie považujeme využitie technológie Angular pre vývoj používateľského rozhrania aplikácie. Výber tejto technológie hovorí o tom, že vývojári sledovali novodobé trendy a štandardy vo vývoji webových aplikácií. Medzi ďalšie výhody patrí prispôsobenie používateľského rozhrania pre mobilné zariadenia a možnosť zaslania spätnej väzby.

1.5.2 Významné paleontologické lokality Slovenska

Významné paleontologické lokality Slovenska [12] je webová e-learningová platforma, ktorá pomáha pri výučbe paleontológie. Táto platforma je pod správou Univerzity Komenského v Bratislave a jej používateľom ponúka informácie o paleontologických lokalitách Západných Karpát z obdobia mezozoika a kenozoika, ktoré je možné aj osobne navštíviť. Táto aplikácia ponúka nasledujúce funkcionality:

Mapa lokalít

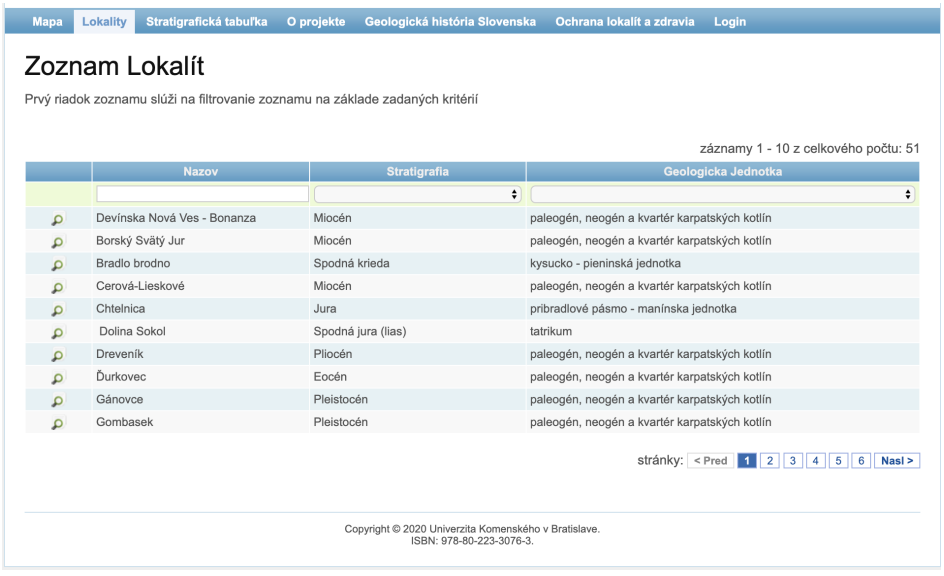
V záložke Mapa, ktorú môžeme vidieť na obrázku 1.4, aplikácia poskytuje interaktívnu mapu s legendami na ktorej sú vyznačené významné paleontologické lokality na Slovensku. Po nájdení kurzorom na vyznačenú lokalitu sa zobrazí jej názov a možnosť kliknutia pre viac informácií. Po kliknutí na lokalitu sa zobrazí jej detail.



Obr. 1.4: Používateľské rozhranie pre mapu lokalít aplikácie Významné paleontologické lokality Slovenska [12]

Lokality

V záložke Lokality, ktorú môžeme vidieť na obrázku 1.5, aplikácia poskytuje tabuľkový prehľad lokalít na území Západných Karpát. V tejto tabuľke sa nachádzajú 3 stĺpce a to pre názov, stratigrafiu a geologické jednotky. V tomto zozname je umožnené taktiež filtrovanie podľa spomenutých stĺpcov. Spolu sa v tomto zozname nachádza 51 lokalít.



Obr. 1.5: Používateľské rozhranie pre detail lokality aplikácie Významné paleontologické lokality Slovenska [12]

Stratigrafická tabuľka

V záložke Stratigrafická tabuľka, aplikácia poskytuje prehľad stratigrafických intervalov, ich veku, trvanie a popis hlavných udalostí a špecifických fosílií pre tieto intervaly. Cieľom stratigrafickej tabuľky je chronologický prehľad geologických období.

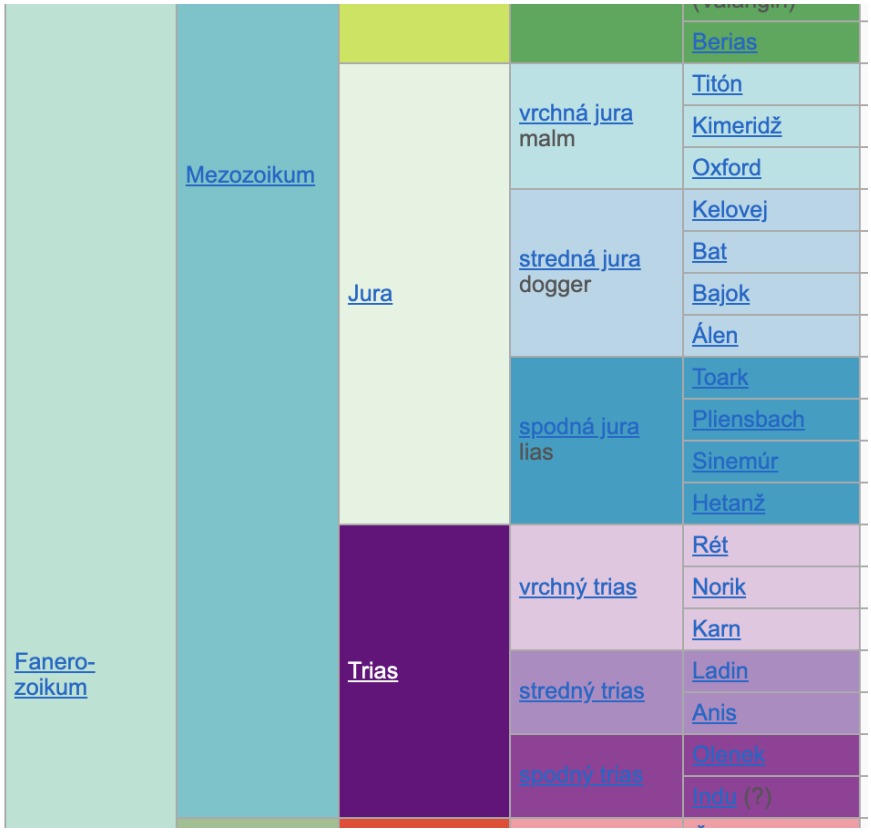
Zhrnutie

Na základe podrobnej analýzy tejto aplikácie sme zistili, že je primárne určená pre žiakov základných a stredných škol, ako aj pre širšiu verejnosť. Záver, ku ktorému sme dospeli, sme rozdelili na dve časti. Prvou časťou sú nedostatky a druhou časťou sú výhody.

Nedostatky: Medzi hlavné nedostatky tejto aplikácie patrí zistenie, že dáta o paleontologických lokalitách sú popísané iba neformálne, dokonca sú niektoré informácie prevzaté z wikipédie a čitateľa odkazujú na relevantnejšie publikácie. Ako ďalší nedostatok aplikácie považujeme to, že ponúka informácie iba o paleontologických lokalitách na území Západných Karpát v období mezozoika a kenozoika. Z pohľadu používateľského rozhrania ponúka nepresné umiestnenie súradníc paleontologických lokalít. V súvislosti s pozíciami lokalít sme zistili, že mapa s legendou sa negeneruje programovo, ale je vložená ako obrázok a body na nej sú vyznačené pomocou css vlastností. Takýto prístup prináša nevýhodu, že pri prípadnej požiadavke na zmenu, akou môže byť napríklad pridanie novej legendy, by bolo potrebné mapu editovať v grafickom editore ako obrázok a vložiť ju naspäť. Aplikácia ďalej nespĺňa požiadavky moderného webdizajnu. Základnou požiadavkou moderného webdizajnu, ktorú táto aplikácia nedodržiava, je neprispôsobiteľnosť pre mobilné zariadenia a zariadenia s menšou obrazovkou. Taktiež použitie veľkého počtu farieb a tmavého textu na tmavom podklade, ako môžeme vidieť na obrázku 1.6. Z pohľadu administračnej časti, táto aplikácia neponúka možnosť procesu schvaľovania dát a priradzovania funkcionalít na základe používateľských rolí a právomoci.

Výhody: Medzi hlavné výhody aplikácie považujeme jednoduchú orientáciu v navigácii aplikácie, ktorá je intuitívna a používateľovi je hneď po príchode na webovú stránku aplikácie jasné, čo má robiť, ak sa chce dopátrať k nejakej informácii. Ďalšou výhodou je možnosť prezretia si fotografií lokalít, ktoré sú dostupné v detaile danej lokality. Rovnako oceňujeme informácie o stave a prístupnosti danej lokality, čo v spojení s informáciami z publikácií na ktoré sa aplikácia odkazuje používateľovi poskytujú dostatočné vedomosti. Podľa týchto vedomostí môže používateľ danú lokalitu aj osobne navštíviť a prebádať, čo môže viesť k zvýšeniu návštevnosti ľudí takýchto lokalít, ktorí síce nie sú z odbornej komunity, ale majú záujem dozvedieť sa viac o paleontologických

lokalitách na našom území.



Obr. 1.6: Ukážka nedodržania princípov moderného webdizajnu aplikácie Významné paleontologické lokality Slovenska [12]

Kapitola 2

Výber technológií

V tejto kapitole sa zaoberáme výberom technológií vhodných na implementáciu webovej aplikácie podľa požiadaviek zadefinovaných v kapitole 1. Čitateľa oboznámime s použitými technológiami v tejto práci, funkcionalitami, ktoré ponúkajú, ako aj s argumentmi, prečo sú vybrané technológie vhodné pre túto aplikáciu a problematiku, ktorú rieši. Pri vývoji webových aplikácií je v ponuke veľké množstvo dostupných technológií a riešení. Preto bolo hlavným cieľom nájsť technológie, ktoré za sebou majú viacero úspešných a komplexných projektov, disponujú dostatočne veľkou komunitou používateľov a ich tvorcovia pokračujú v udržiavaní a vydávaní nových verzií. Výber technológií ovplyvnili aj predchádzajúce skúsenosti a v prípade nástroja Craftable aj spoluautorstvo. Všetky použité technológie spomenuté v tejto kapitole majú otvorený zdrojový kód.

2.1 Laravel

Podľa požiadaviek vieme, že aplikácia má byť vyhotovená formou viacpoužívateľskej webovej aplikácie. Najprv sme sa zamerali na technológie v ktorých budeme implementovať serverovú časť aplikácie. Technológií pre implementáciu tejto časti aplikácie je viacero, ale vybrali sme si webový aplikačný rámec s názvom Laravel. Laravel je MVC webový aplikačný rámec napísaný v jazyku PHP, ktorý je dostupný vo viacerých verziách. Najaktuálnejšie verzie sú 6 a 7. Pre túto prácu sme sa rozhodli pre výber verzie 6. Verzia 6 narozdiel od verzie 7 ponúka dlhodobú podporu a podľa oficiálnej dokumentácie [6] bude podporovaná do 3. septembra 2022, narozdiel od verzie 7, ktorá bude podporovaná do 3. marca 2021. Dôvodom výberu tohto aplikačného rámca je jeho kvalitná dokumentácia, široká škála funkcionalít, ktoré ponúka, rastúca popularita a komunita. Hlavné funkcionality, ktoré Laravel [6] prináša sú:

- **Eloquent ORM**, ktorý ponúka objektovo relačné mapovanie dát, čo nám umožňuje pristupovať k dátam uložených v databáze objektovo.

- **Query builder**, vďaka ktorému nemusíme písať databázové dopyty pomocou jazyka SQL, ale pomocou metód, ktoré nám ponúka. Taktiež nás chráni pred typom útoku s názvom SQL injection a zabezpečuje, že aplikácia nie je závislá na konkrétnom databázovom systéme, ale ponúka nám možnosť výberu zo 4 podporovaných databázových systémov. Podporované databázové systémy sú: MySQL, PostgreSQL, SQLite a SQL Server.
- **Kolekcie**, ktoré ponúkajú širokú škálu metód pre prácu s poliami dát.
- **Systém riadenia databázových migrácií**, ktorý uľahčuje proces vytvárania a aktualizovania databázových schém.
- **Jednotkové testy** ako súčasť aplikačného rámca, čo umožňuje spolu vyvíjať kód a testy, ktoré overujú jeho funkcionalitu.
- **Plánovač úloh** v ktorom môžeme spúšťať rôzne plánované úlohy, ako napríklad odosielanie emailových notifikácií, automatická anonymizácia používateľov a podobne.
- **Blade komponenty**, ktoré rozširujú html o možnosti použitia PHP kódu a vlastných konštrukcií.

Podľa portálu packagist [8], ktorý slúži ako portál pre prehľad všetkých voľne dostupných PHP balíčkov, má Laravel doposiaľ 90 964 081 inštalácií, čo hovorí o obľúbenosti a rozšírenosti tohto PHP aplikačného rámca. V tejto práci využívame spomenuté funkcionality, ktoré nám pomohli vytvoriť bezpečnú, spoľahlivú, jednoducho udržiavateľnú a škálovateľnú aplikáciu, pripravenú na prípadne zmeny alebo rozšírenia funkcionality počas jej prevádzkovania.

2.2 Vue.js

Po výbere technológie pre implementáciu serverovej časti aplikácie sme sa zamerali na výber technológie pre implementáciu používateľského rozhrania aplikácie. Naším výberom bola knižnica s názvom Vue.js, ktorá slúži na vytváranie používateľského rozhrania a je napísaná v jazyku Javascript. Vue.js je podľa prieskumu s názvom State of Javascript [11], ktorého sa zúčastnilo 21 717 respondentov z odbornej komunity za rok 2018 knižnica, ktorá je v rebríčku ako prvá v oblasti záujmu programátorov a v roku 2019 sa umiestnila na druhom mieste. V rebríčku podľa rokov skúseností programátorov sa Vue.js umiestnilo na druhom mieste. Ďalšou výhodou tejto knižnice je jej rozšírenosť v Laravel komunite. Dôvodom výberu tejto knižnice je jej rastúca popularita, krivka učenia a jednoduchá integrácia s Laravel aplikáciou. Hlavné výhody, ktoré táto knižnica prináša podľa oficiálnej dokumentácie [13] sú:

- veľmi malá veľkosť (18-21kb),
- detailná dokumentácia,
- jednosmerný tok dát,
- hlavné balíčky, ktoré nie sú súčasťou knižnice, sú vyvíjané tvorcami,
- komponenty.

Komponenty nám pomáhajú zabaliť a znovu použiť kód s rovnakou funkcionalitou, čo pomáha pri kvalite a čitateľnosti kódu. Výhodou, ktorú nám prináša jednosmerný tok dát, je možnosť sledovania zmien dát a akcií, ktoré tieto zmeny vykonali, čo pomáha pri procese vylaďovania komplexnejších aplikácií.

Tabuľka 2.1 zobrazuje porovnanie s konkurenčnými aplikačnými rámcami a knižnicami, ktoré slúžia na vytváranie používateľského rozhrania. Tabuľku sme zostavili na základe informácií z oficiálnych dokumentácií Vue.js [13], Reactu [10] a Angularu [2], ktoré sú doplnené o informácie z prieskumu State of Javascript [11] a informácie z platformy GitHub. Platforma GitHub je známa a rozšírená vo svete projektov s otvoreným zdrojovým kódom a programátorom umožňuje pri každom projekte vytvoriť pripomienku, ktorá môže byť návrh na vylepšenie, otázka alebo nahlásenie chyby. Z tohto dôvodu považujeme pri výbere technológií GitHub za relevantný zdroj informácií.

	Vue.js	**React**	**Angular 2**
Spoločnosť	žiadna spoločnosť	Facebook	Google
Komponenty	áno	áno	áno
Virtual DOM	áno	áno	nie
Veľkosť	18-21kb	109 kb	neuvádza
Popularita (počet GitHub hviezdičiek)	162k [14]	147k [4]	59.9k [1]
Počet pripomienok na GitHube	305 [14]	465 [4]	2 808 [1]
Tok dát	jednosmerný	jednosmerný	dvojsmerný
Miesto v rebríčku záujmu*	2.	3.	5.
Miesto v rebríčku skúseností*	2.	1.	3.

* **informácie za rok 2019 ; k - tisíc**

Tabuľka 2.1: Porovnanie technológií

Z dát tabuľky 2.1 môžeme vidieť, že Vue.js má v porovnaní s Angularom väčšiu popularitu. Pri analýze pripomienok na platforme GitHub sme zistili, že v prípade Vue.js je 45 pripomienok označených ako chyba, v prípade Reactu to je 103 pripomienok a v prípade Angularu 1198 z čoho môžeme usúdiť, že Vue.js je najspoľahlivejšou voľbou pre túto prácu.

2.3 Craftable

Podľa požiadaviek má naša aplikácia poskytovať prihlásenému používateľovi administračné rozhranie. Takéto rozhranie nemusíme implementovať od základov, ale môžeme ho čiastočne vygenerovať z dátového modelu pomocou generátora. Takýchto generátorov je v ponuke viac. Jeden z nich je Craftable. Craftable je nástroj na vytváranie administračných rozhraní napísaný v Laraveli a Vue.js. Craftable podľa [3] ponúka nasledujúce kľúčové vlastnosti:

- **CRUD generátor**, ktorý generuje funkcionalitu pre akcie vytvárania, zobrazovania, editácie a vymazávania dát danej entity.
- **Podpora používateľských rolí a právomocí**, ktorá nám pomôže pri definovaní vlastných rolí a právomoci pre prihlásených používateľov.
- **Autentifikačný modul**, ktorý sa stará o prihlasovanie a overovanie používateľov.
- **Podpora viacerých verzii Laravelu**, ktorý umožňuje vytváranie aplikácie v Laravel 6 a 7.
- **Manažment prekladov v aplikácii**, ktorý umožňuje pre každý jazyk v aplikácii samostatný preklad slov alebo fráz aplikácie.

Tento nástroj uľahčuje prácu pri tvorbe CMS systémov, CRM systémov ako aj Back Office systémov, čo je aj prípad aplikácie pre správu litostratigrafických dát. CRUD generátor poskytuje na základe databázových migrácií základnú štruktúru a funkcionalitu, čo programátorovi umožňuje zamerať sa viac na logiku aplikácie. Nakoľko je vygenerovaný kód v konečnom dôsledku len kód napísaný v Laraveli a Vue.js, programátorovi je umožnené si kód upraviť a rozšíriť podľa svojich predstáv. Craftable svoju konkurenciu nachádza napríklad v nástroji s názvom Laravel Nova [7], ktorý je vytvorený tvorcom Laravelu, ale narozdiel od Craftable je spoplatnený. V najlacnejšej variante je dostupný s cenovkou 99$ za projekt.

2.4 PostgreSQL

Na základe požiadaviek vieme, že aplikácia má umožňovať uchovávať informácie o litostratigrafických dátach, preto bol pred nami krok výberu databázového systému. Na základe predošlých skúseností a znalostí sme zvolili PostgreSQL. PostgreSQL je podľa [9] objektovo relačný databázový systém, ktorý používa a rozširuje jazyk SQL. PostgreSQL dodržiava vlastnosti ACID (Atomicita, Konzistentnosť, Izolovanosť, Trvanlivosť) databázových transakcií, čo zaručuje konzistentnosť dát uložených v databázovom systéme. Používateľom umožňuje vytváranie vlastných databázových funkcií v jazyku PL/SQL, vytváranie indexov nad JSON stĺpcami, materializované pohľady a

mnoho ďalších funkcionalít, čím sa líši od konkurenčných databázových systémov. Ponúka veľmi kvalitnú a podrobne spracovanú dokumentáciu, čo používateľom zaručuje istotu, že všetky potrebné informácie nájdu na jednom mieste.

2.5 Docker a Harbor

Posledným krokom, ktorý je počas životného cyklu aplikácie veľmi dôležitý a to nie len počas jej vývoja, ale aj udržiavania alebo pridávania nových funkcionalít, je zjednodušenie inicializácie aplikácie na lokálnom prostredí. Preto sme siahli po softvéri s názvom Docker a jeho rozšírení s názvom Harbor. Docker je softvér, ktorý pomocou kontajnerov poskytuje jednotné rozhranie pre aplikáciu za pomoci virtualizácie na úrovni operačného systému. Podľa [15] je kontajner jednotka softvéru, ktorá zabaľuje kód a všetky jeho závislosti, vďaka čomu je aplikáciu možné spúšťať na viacerých výpočtových prostrediach. Docker je možné spúšťať na operačných systémoch Linux, OSX a Windows. Harbor je nástroj, ktorý poskytuje Docker konfiguráciu pre Laravel a Craftable projekty. Harbor sa skladá z nasledujúcich kontajnerov:

- Nginx kontajner,
- PHP kontajner,
- PostgreSQL kontajner,
- PostgreSQL kontajner pre jednotkové testy,
- Node.js kontajner,
- Redis kontajner.

Kapitola 3

Aplikácia z pohľadu používateľa

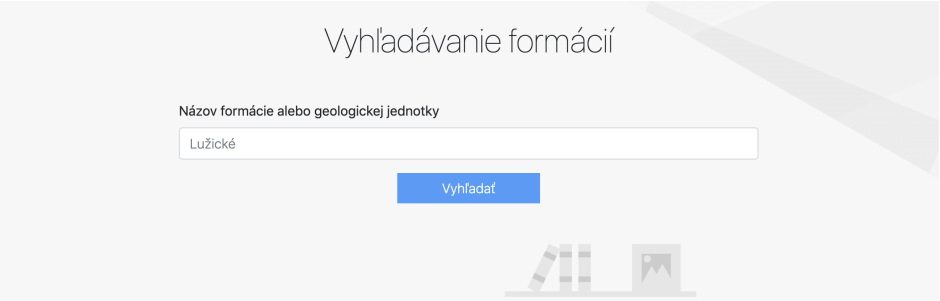
V tejto kapitole popisujeme aplikáciu z pohľadu viacerých typov používateľov. Aplikácia je rozdelená na dve hlavné časti a to časťou pre neprihlásených používateľov a administračnou časťou pre prihlásených používateľov. Ako prvú si predstavíme časť aplikácie z pohľadu neprihláseného používateľa a v druhej časti tejto kapitoly pohľad na administračnú časť prihláseného používateľa na základe jeho priradenej role. Cieľom bolo vytvoriť moderný, funkčný a intuitívny dizajn, ktorý sa riadi princípmi moderného webdizajnu.

3.1 Aplikácia z pohľadu neprihláseného používateľa

V tejto časti práce si bližšie popíšeme používateľské rozhranie pre neprihláseného používateľa, ktoré spĺňa požiadavky popísané v kapitole 1.

3.1.1 Vyhľadávanie formácií

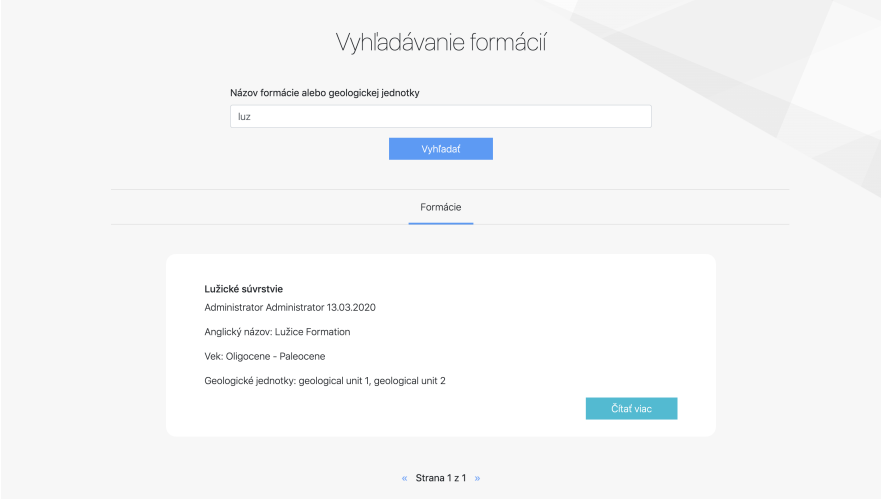
Neprihlásený používateľ je po príchode na adresu webovej aplikácie presmerovaný na obrazovku vyhľadávania litostratigrafických formácií, ktorej časť môžeme vidieť na obrázku 3.1.



Obr. 3.1: Hlavná obrazovka aplikácie pre neprihláseného používateľa

Používateľ môže do vyhľadávania zadať frázu, ku ktorej si praje nájsť výsledky.

Fráza zadaná do vyhľadávača musí obsahovať minimálne 3 znaky a po kliknutí na tlačidlo *Vyhľadať* sa zobrazia všetky relevantné výsledky pre zadanú frázu, ako môžeme vidieť na obrázku 3.2.



Obr. 3.2: Obrazovka aplikácie pre nájdené výsledky na základe zadanej frázy

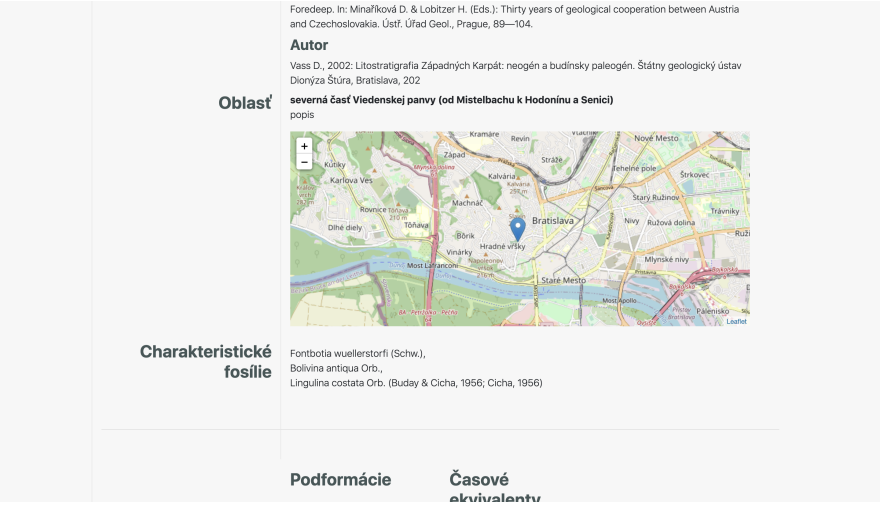
Prehľadávanie výsledkov je používateľovi poskytnuté formou stránkovania, kde sa na jednej stránke nachádza 5 výsledkov. Jeden výsledok v sebe obsahuje základné atribúty litostratigrafickej formácie pre lepšiu orientáciu vo výsledkoch. Atribútmi, ktoré sa zobrazujú sú: slovenský a anglický názov formácie, mená autorov a dátum poslednej úpravy formácie, intervaly a geologické jednotky. Pre zobrazenie detailu formácie je používateľovi umožnené kliknúť na tlačidlo *Čítať viac*.

3.1.2 Detail formácie

Na obrázkoch 3.3 a 3.4 môžeme vidieť obrazovky pre zobrazenie detailu litostratigrafickej formácie na ktorú sa používateľ dostane kliknutím na tlačidlo *Čítať viac*.

Obr. 3.3: Detail formácie pre neprihlásených používateľov (časť 1)

Tento detail poskytuje prehľadné zobrazenie všetkých atribútov a vzťahov litostratigrafickej formácie (autori formácie, slovenský názov, anglický názov, synonymá, statigrafické intervaly, hrúbka, geologické jednotky, zdôvodnenie názvu, história, genéza, litologický popis, charakteristické fosílie, výskyt, profily, publikácie, členy). Pre typové a referenčné profily je poskytnuté zobrazenie geografickej pozície pomocou ukazovateľov na interaktívnej mape, ako môžeme vidieť na obrázku 3.4.



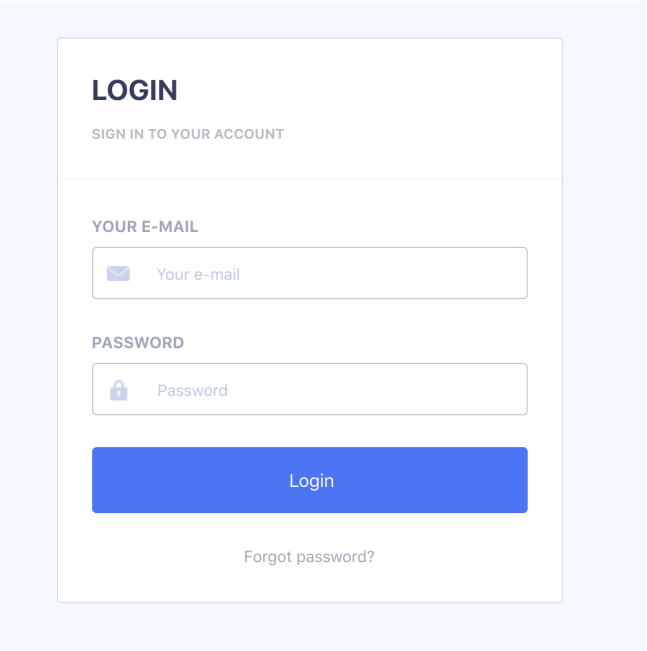
Obr. 3.4: Detail formácie pre neprihlásených používateľov (časť 2)

V detaile formácie pre neprihlásených používateľov sa zobrazujú dáta o poslednej schválenej verzii formácie. V detaile sa nezobrazujú systémové dáta, akými sú napríklad: stav formácie, komentár recenzenta, predošlé verzie, informácie o vykonaných úpravách atď.

3.2 Aplikácia z pohľadu prihláseného používateľa

V tejto časti práce si bližšie popíšeme administračné rozhranie pre prihlásených používateľov na základe ich rolí. Administračné rozhranie ponúka jednotný dizajn a responzívnosť pre mobilné zariadenia. Funkcionalita je pre prihlásených používateľov sprístupnená na základe požiadaviek popísaných v kapitole 1 v časti požiadavky na aplikáciu. Narozdiel od časti pre neprihlásených používateľov je administračná časť aplikácie v anglickom jazyku, za účelom zjednodušenia ovládania aplikácie aj pre používateľov, ktorý neovládajú slovenský jazyk (dobrým príkladom môže byť zahraničný študent doktorandského štúdia). V prípade potreby je možné administračnú časť preložiť a prepnúť do slovenského jazyka. Z požiadaviek na aplikáciu vieme, že prihláseným používateľom môže byť pridelená jedna z používateľských rolí *tvorca*, *recenzent* alebo *administrátor* a dáta musia prejsť procesom schvaľovania. Procesom schvaľovania musí prejsť každé vytvorenie alebo editácia už existujúcej formácie. Schvaľovanie dát je úlo-

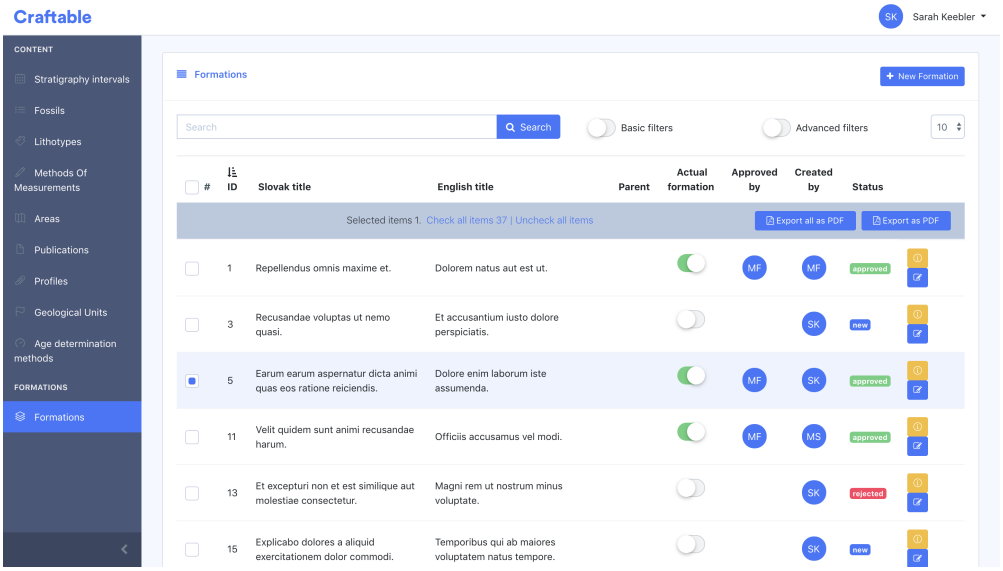
hou používateľskej role *recenzent*. Používateľ sa do aplikácie prihlasuje pomocou svojho konta, ktoré mu na základe emailovej adresy vytvoril administrátor. Prihlasovací formulár do administračnej časti aplikácie môžeme vidieť na obrázku 3.5



Obr. 3.5: Prihlasovací formulár do administračnej časti aplikácie

3.2.1 Tvorca

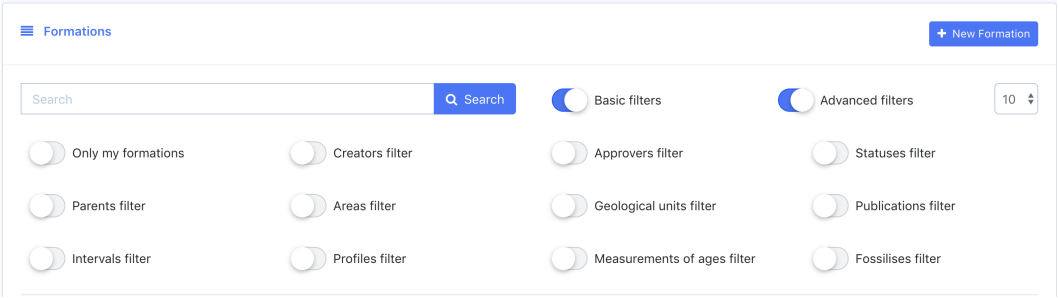
Tvorcom je po prihlásení do administračnej časti aplikácie v záložke formácie umožnené vidieť výpis svojich a schválených formácií, ako môžeme vidieť na obrázku 3.6.



Obr. 3.6: Výpis schválených litostratigrafických formácií pre tvorcu

Tieto dáta je umožnené exportovať po zakliknutí konkrétnych formácií. Nakoľko je úlohou tvorcu vytváranie nových formácií, alebo prípadná editácia a korektúra už schválených formácií, môže vo výpise formácií kliknúť pri konkrétnej formácii na akciu editácie, alebo tlačidlo vytvárania, ktoré môžeme vidieť v pravom hornom rohu.

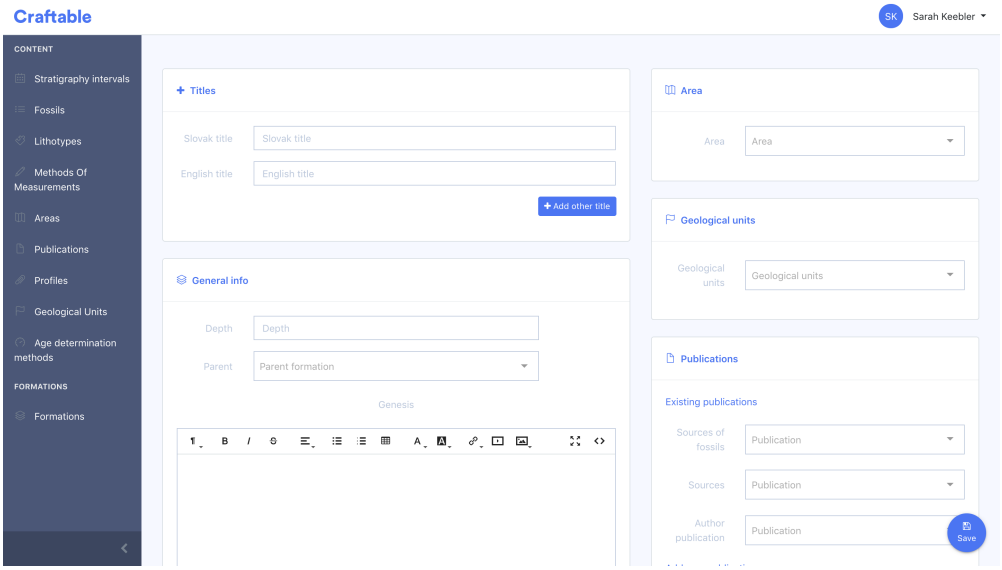
Pre zefektívnenie práce s litostratigrafickými formáciami umožňuje aplikácia základné a pokročilé filtrovanie dát, ktoré môžeme vidieť na obrázku 3.7.



Obr. 3.7: Filtrovanie dát litostratigrafických formácií

Vytváranie a editácia litostratigrafickej formácie

Tvorcovi sa po zakliknutí tlačidla akcie vytváranie novej/editácie existujúcej litostratigrafickej formácie zobrazí formulár, ktorý môžeme vidieť na obrázku 3.8.



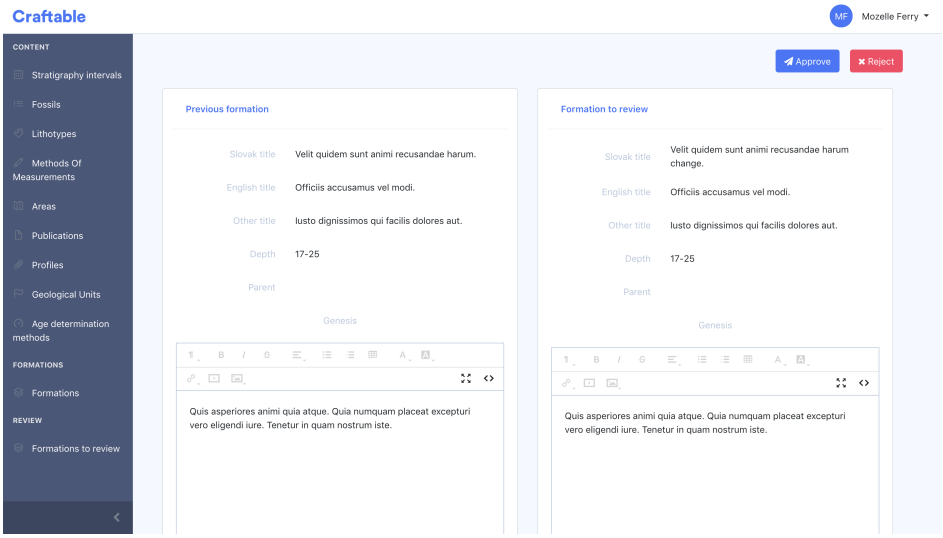
Obr. 3.8: Formulár pre vytvorenie novej litostratigrafickej formácie

V prípade editácie existujúcej formácie je tento formulár vyplnený dátami o zvolenej formácii. Všetky dáta formulára sú povinné a tvorca si môže formáciu uložiť v prípade, ak plánuje doplniť alebo zmeniť dáta formácie. Po uložení ju môže tvorca odoslať na schválenie. Dáta o oblasti, geologických jednotkách, publikáciách, rodičovskej formácii,

fosíliách, stratigrafických intervaloch, profiloch a metód určovania veku sa vyberajú zo zoznamu, v ktorom je umožnené vyhľadávanie.

3.2.2 Recenzent

Recenzentovi je po prihlásení do administračnej časti aplikácie v záložke formácie na recenzovanie zobrazený výpis formácií, ktoré boli odoslané na schválenie. Recenzent môže pri takejto formácii začať akciu recenzovania. Pri akcii recenzovania sa používateľovi zobrazí formulár, ktorý môžeme vidieť na obrázku 3.9.

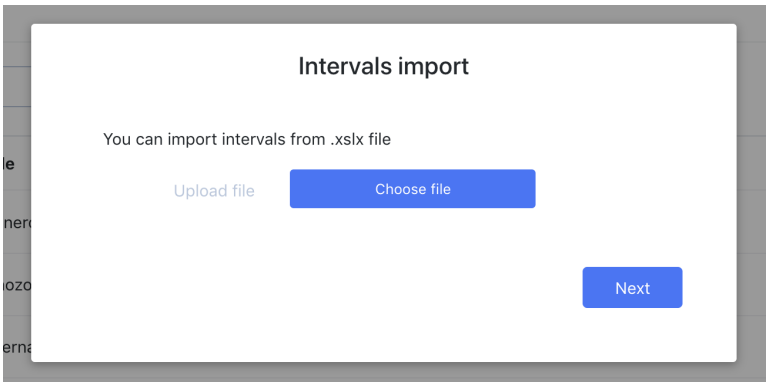


Obr. 3.9: Formulár pre recenzovanie stratigrafických formácií

Tento formulár sa skladá z dvoch stĺpcov. V ľavom stĺpci recenzent vidí dáta pôvodnej schválenej formácie a v pravom stĺpci vidí dáta odoslané na schválenie. V prípade formácií, ktoré nemajú predchádzajúcu schválenú verziu je ľavý stĺpec prázdny. Recenzent môže formáciu schváliť stlačením tlačidla *Approve*, čo spôsobí, že pri danej formácii sa bude zobrazovať jeho meno pri stĺpci *Approved by*. Recenzent môže taktiež formáciu zamietnuť tlačidlom *Reject*, kde môže uviesť aj komentár s dôvodom zamietnutia v políčku *Review*.

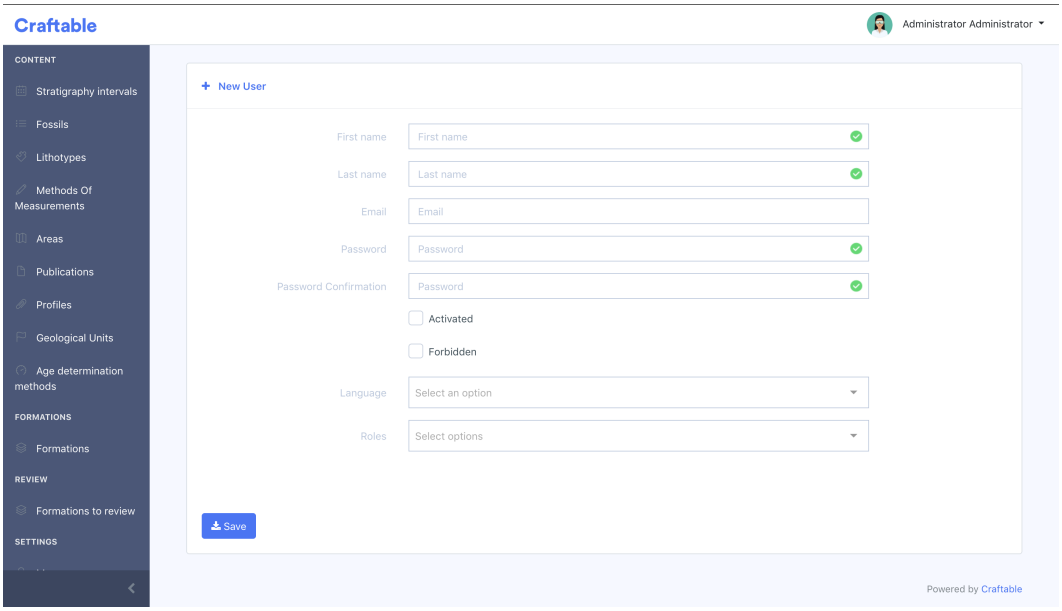
3.2.3 Administrátor

Administrátor má najvyššie právomoci v aplikácii, čo znamená, že môže robiť všetko, čo ostatné spomenuté roly. Navyše mu je po prihlásení do administračnej časti aplikácie umožnené v záložke stratigrafické intervaly zvoliť akciu importovania, zobrazenia, editácie alebo vytvorenia stratigrafického intervalu. V prípade zvolenia možnosti importovania, ktorého modálne okno môžeme vidieť na obrázku 3.10, musí byť importovaný súbor vo formáte xslx.



Obr. 3.10: Modálne okno pre importovanie stratigrafických intervalov

V záložke manažmentu prístupu môže administrátor zvoliť akciu vytvorenia, alebo editácie používateľského konta, ktorého formulár môžeme vidieť na obrázku 3.11.



Obr. 3.11: Formulár pre vytváranie používateľského konta

V prípade editácie sú položky formulára vyplnené. Používateľom je vo formulári okrem vyplnenia/zmeny atribútov, umožnený výber používateľskej role zo zoznamu, nastavenie jazyka aplikácie z dostupných jazykov, alebo možnosť zablokovania účtu pre vybraného používateľa.

Kapitola 4

Technický pohľad na aplikáciu

V tejto kapitole sa zaoberáme technickým pohľadom na aplikáciu. Čitateľ sa oboznámi so všetkými potrebnými detailmi, postupmi a riešeniami pre lepšie porozumenie tejto práci. Na začiatku tejto kapitoly sa dozvieme o dátovom modeli aplikácie, potrebných zmenách oproti predbežnému dátovému modelu a transformácii na relačný model. Po oboznámení s dátovým modelom sa pozrieme na štruktúru kódu a princípov pri jeho implementácii. Priblížime si zamykanie dát, export a import dát, ako aj vybrané funkcionality, ktoré sú nad rámec požiadaviek zadefinovaných v kapitole 1. V tejto kapitole sa často stretneme so slovným spojením status formácie, čim máme na mysli atribút *locked*.

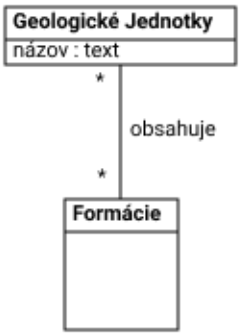
4.1 Dátový model

Dátový model vychádza z predbežnej verzie dátového modelu, ktorý sa nachádza na obrázku 1.1. Predbežný dátový model v sebe nezahŕňal riešenie pre duplikovanie dát za účelom uchovávania viacerých verzií a procesu schvaľovania. Ďalej v sebe nezahŕňal množiny entít pre geologické jednotky, používateľov, ich rolí a právomocí ako aj systémových informácií pre chod aplikácie. Všetky potrebné vykonané zmeny a vylepšenia si bližšie popíšeme vo viacerých bodoch. Z dôvodu počtu množín entít a lepšej čitateľnosti je dátový model rozdelený na 3 časti, ktoré môžeme vidieť na obrázkoch 4.6, 4.8 a 4.9.

Nové atribúty a množiny entít

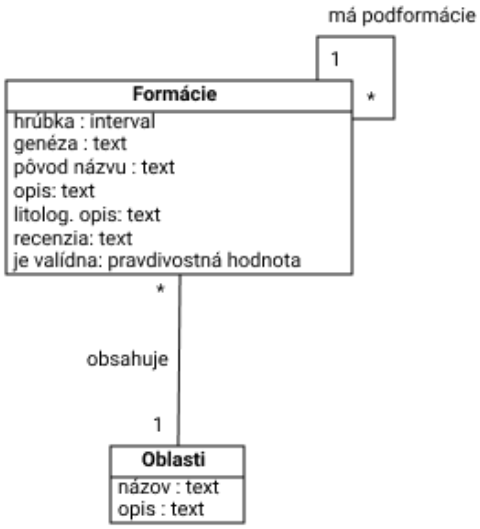
V priebehu vývoja aplikácie prichádzali nové požiadavky, medzi ktoré patrilo aj pridanie podpory aplikácie pre uchovávanie informácií o geologických jednotkách. Nakoľko sa môže geologická jednotka vyskytovať vo viacerých formáciách a rovnako môže mať formácia viacero geologických jednotiek, rozhodli sme sa, že do dátového modelu pridáme množinu entít pre geologické jednotky vo vzťahu $m : n$ k formáciám s atribútom

názov, ako môžeme vidieť na obrázku 4.1.



Obr. 4.1: Pridanie množiny entít pre geologické jednotky

Ďalšími požiadavkami bolo umožnenie nastavovať formácii informáciu o jej platnosti, možnosť pridávania komentárov pre recenzentov v procese schvaľovania dát a možnosť priradiť oblasti názov. V dôsledku týchto požiadaviek sme do množiny entít *Formácie* pridali atribúty *je valídna* a *recenzia* a pre množinu entít *Oblasti* atribút *názov*, ako môžeme vidieť na obrázku 4.2. Dôvodom pridania atribútu *recenzia* priamo do množiny entít *Formácie* je, že recenzent v procese schvaľovania dát pridáva komentár ku konkrétnej formácii, ktorá bola odoslaná na schválenie.



Obr. 4.2: Pridanie nových atribútov pre množiny entít Formácie a Oblasti

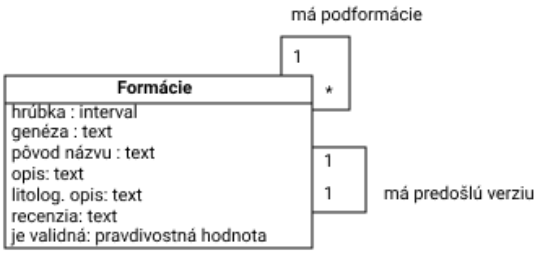
Ukladanie viacerých verzií dát

Za účelom zobrazovania poslednej schválenej verzie formácie pri jej editácii a následnom procese schvaľovania bolo potrebné rozšíriť dátový model o verziovanie. Pri takejto zmene dátového modelu bolo dôležité zvoliť vhodný postup, ktorým sa dopracujeme k

riešeniu, ktoré bude jednoducho implementovateľné a výkonnostne nezaťaží databázový systém. Jedno z možných riešení bolo pridať pre každú množinu entít, ktorá si vyžaduje podporu verziovania, ďalšiu množinu entít pre ukladanie jej novšej verzie dát. Pri takomto riešení by sa v pôvodných množinách entít ukladali aktuálne schválené verzie dát a v novo pridaných množinách entít ich novšie neschválené verzie. Pri takomto riešení by sa v prípade schválenia dát prepísala aktuálna schválená verzia novou, čím by sme informácie o predošlej verzii dát stratili. Takéto riešenie by pri jeho implementácii viedlo k duplicite atribútov a tried, ktoré by reprezentovali dané množiny entít rozšírené o verziovanie. Ďalším problémom tohto riešenia by bola nutnosť pridania zložených primárnych kľúčov pre novo pridané množiny entít za účelom rozoznávania poradia verzii. Primárny kľúč by musel byť zložený z atribútov *vytvorená* a cudzieho kľúča na tabuľku entity, ktorú chceme rozšíriť o podporu verziovania. To by viedlo k zložitému dopytovaniu pre zobrazovanie dát v rôznych verziách, ktoré potrebujeme napríklad pri procese schvaľovania. Nakoľko pre porovnanie rozdielov vo verziách by sme museli hľadať záznamy aj v tabuľkách a vzťahoch určených pre novšie verzie, narozdiel od dopytovania záznamov a ich vzťahov z jednej tabuľky, čo je časovo menej náročné.

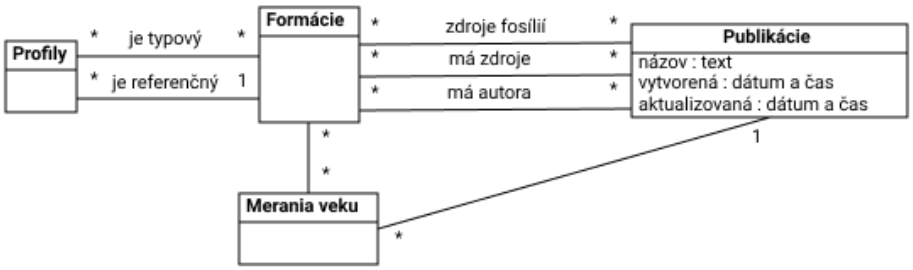
Naše riešenie

Pre ukladanie viacerých verzií dát sme zvolili nasledovné riešenie. Rôzne verzie jednej entity sú uložené ako samostatné entity v tej istej množine entít. Každá entita je potom cez vzťah *má predošlú verziu* naviazaná na svoju predošlú verziu, čo nám umožní jednoducho zobraziť dáta v rôznych verziách pri procese schvaľovania recenzentom. Pridanie vzťahu má predošlú verziu pre množinu entít *Formácie* môžeme vidieť na obrázku 4.3.



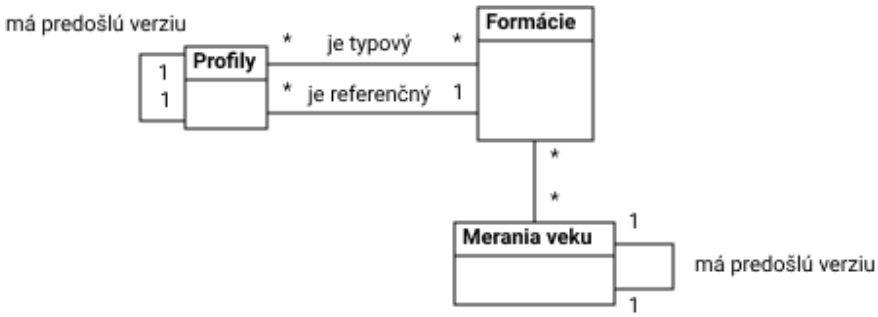
Obr. 4.3: Pridanie vzťahu má predošlú verziu pre množinu entít Formácie

Tým, že v množine entít si namiesto jednej entity ukladáme jej viaceré verzie, potrebujeme zmeniť kardinalitu vzťahov tejto množiny entít. Zmenili sme kardilalitu *1 : 1* pre vzťah *je typový*, kardinalitu *1 : n* pre vzťah *formácie a merania veku* a kardinalitu *n : 1* pre vzťah *má autora* na vzťahy s kardinalitou *n : m*, ako môžeme vidieť na obrázku 4.4.



Obr. 4.4: Zmena kardinalít v dátovom modeli

Táto zmena nám umožnila naviazať *typové profily* a *merania veku* k viacerým verziám formácie, čo by pri pôvodných kardinalitách nebolo možné. Dôsledkom tejto zmeny bola nutnosť pridať vzťah *má predošlú verziu* aj pre množiny entít *Profily* a *Merania veku* v prípade, že by používateľ editoval existujúce záznamy týchto entít, ktoré boli naviazané na už schválenú verziu formácie. V spomenutom prípade chceme z dôvodu, aby nedochádzalo k neželaným zmenám v už schválených dátach, vytvoriť novšiu verziu editovanej entity a formácie, ktorá sa odošle na schválenie. Pridanie vzťahu *má predošlú verziu* môžeme vidieť na obrázku 4.5.



Obr. 4.5: Pridanie vzťahu má predošlú verziu pre množiny entít Profily a Merania veku

Zamykanie formácií pri ich editácii a procese schvaľovania dát

Podľa požiadavky na zamykania dát počas editácie a procesu schvaľovania formácií sme vykonali v dátovom modeli nasledujúce zmeny:

Množina entít *Formácie*

Pre množinu entít *Formácie* sme do dátového modelu pridali nasledujúce atribúty:

- *zamknutá* - účelom tohto atribútu je rozoznať v akom stave procesu schvaľovania sa formácia nachádza. Môže nadobúdať hodnoty: *new*, *in_review*, *rejected*, *approved*.

- *editovaná* - dátum a čas začiatku akcie editácie formácie.

Ďalej sme do dátového modelu pridali pre množinu entít *Formácie* vzťahy *vytvoril* a *editoval* v kardinalite *n : 1* k množine entít *Administračný používatelia*, na základe ktorých vieme zistiť, ktorí používatelia majú dáta zamknuté.

Ukladanie systémových dát

V tejto časti práce si bližšie popíšeme časť dátového modelu, ktorú automaticky vytvoril používaný nástroj Craftable. Ďalej si popíšeme nové systémové atribúty, ktoré sme pridali pre vybrané množiny entít.

Craftable

Nástroj Craftable pridal do dátového modelu nasledujúce množiny entít:

- *Administračný používatelia* - táto množina entít reprezentuje používateľov administračnej časti aplikácie. Uchováva informácie o mene, emailovej adrese, hesle a podobne.

- *Aktivácie administrátorov* - táto množina entít v sebe uchováva informácie o overovaní emailov pre používateľov administračnej časti aplikácie.

- *Resetovania administračných hesiel* - táto množina entít v sebe uchováva informácie o tokenoch, ktoré sa používajú na obnovenie hesla pre používateľov administračnej časti aplikácie.

- *Média* - táto množina entít v sebe uchováva informácie o nahratých súboroch a médiach v aplikácii (obrázky, videá, zvukové nahrávky atď...).

- *Role* - táto množina entít v sebe uchováva informácie o používateľských rolách, ktoré môžu byť priradené používateľom aplikácie.

- *Právomoci* - táto množina entít v sebe uchováva informácie o používateľských právomociach, ktoré môžu byť pridelené roli alebo samotnému používateľovi.

- *Preklady* - táto množina entít v sebe uchováva informácie o jazykových prekladoch textov a hlášok aplikácie.

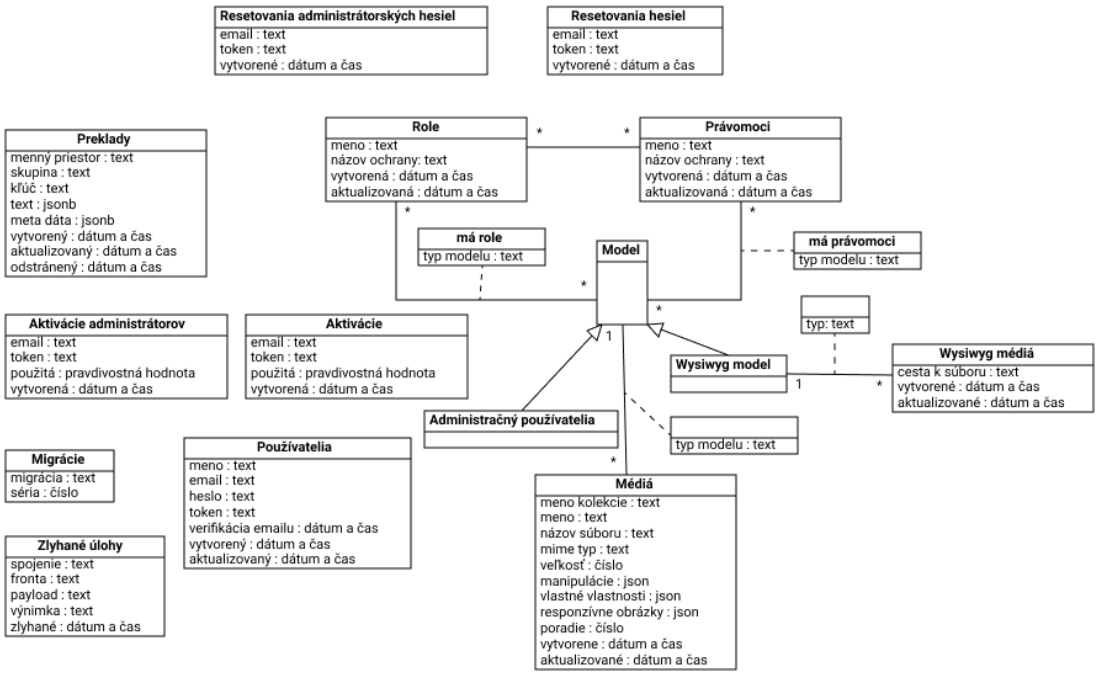
- *Wysiwyg média* - táto množina entít v sebe uchováva informácie o nahratých súboroch a médiach pomocou wysiwyg editora.

Laravel

Ako z kapitoly 2 vieme, Craftable je postavené na aplikačnom rámci Laravel, čoho dôsledkom je pridanie množín entít s názvami *Používatelia, Aktivácie, Resetovania hesiel, Migrácie* a *Zlyhané úlohy* do nášho dátového modelu. Tieto množiny entít sú súčasťou každého Laravel projektu a my si z týchto množín entít popíšeme iba množinu entít *Migrácie*, ktorú v našej aplikácii využívame.

- *Migrácie* - táto množina entít v sebe uchováva meta informácie o úspešne zmigrovaných databázových migráciách. Vďaka informáciám z tejto tabuľky sa vieme v prípade potreby vrátiť k stavu databázového systému pred zmigrovaním zadaného počtu migrácií.

Spomenuté množiny entít a ich atribúty môžeme vidieť na obrázku 4.6

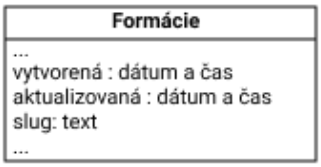


Obr. 4.6: Dátový model ukladania systémových dát

Nové systémové atribúty

Pre umožnenie sledovania informácií o tom, kedy boli dáta v našich množinách entít vytvorené alebo naposledy aktualizované, sme sa rozhodli pridať pre každú množinu entít nové atribúty s názvami *vytvorená* a *aktualizovaná*. Využitie týchto atribútov sme mohli vidieť napríklad v kapitole 3, kde sa neprihláseným používateľom pri výsledkoch vyhľadávania zobrazovala informácia o tom, kedy boli informácie o formácii naposledy aktualizované. Ďalej sme sa rozhodli pre množinu entít *Formácie* pridať atribút *slug*,

ktorého účelom je zjednodušenie zapamätania si url adresy detailu formácie pre neprihlásených používateľov. Ako príklad si môžeme uviesť formáciu, ktorá má názov Lužické súvrstvie. Pre takúto formáciu je hodnota atribútu *slug luzicke_ suvrstvie* a url adresa detailu tejto formácie pre neprihlásených používateľov obsahuje text */detail/luzicke_ suvrstvie*. Spomenuté zmeny si ilustrujeme na množine entít *Formácie*, ktorú môžeme vidieť na obrázku 4.7.

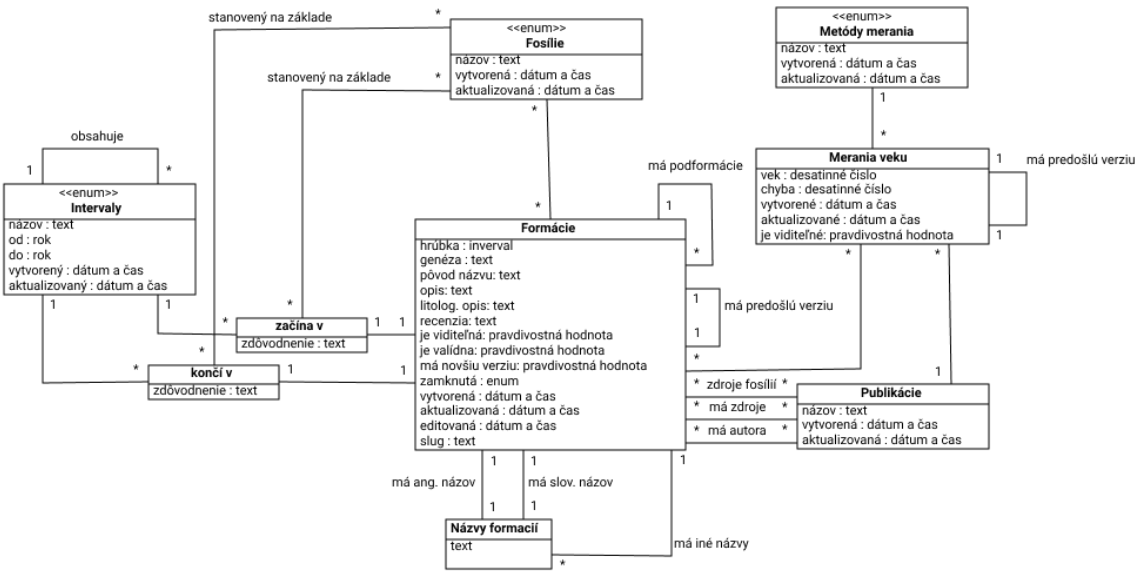


Obr. 4.7: Pridanie atribútov vytvorená, aktualizovaná a slug pre množinu entít Formácie

Množiny entít *Formácie*, *Profily* a *Merania veku*

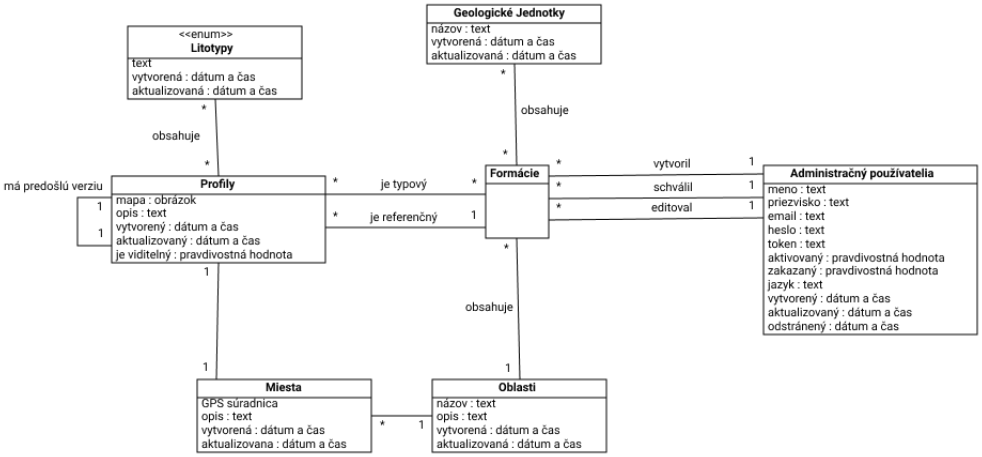
Pre množiny entít *Formácie*, *Profily* a *Merania veku* sme do dátového modelu pridali atribút *je viditeľná/ý*, ktorého účelom je zjednodušenie dopytovania dát, ktoré sa majú zobrazovať v administračnom rozhraní a neprihláseným používateľom. Tento atribút je nastavený na pravdu iba ak ide o dáta poslednej schválenej verzie formácie, inak je nastavený na nepravdu.

Všetky spomenuté zmeny v tejto časti práce môžeme vidieť na obrázkoch 4.8 a 4.9, na ktorých sa nachádza finálna verzia dátového modelu aplikácie.



«enum» označuje dáta, ktoré sa často nemenia a pri editácii formácie sa vyberajú zo zoznamu

Obr. 4.8: Dátový model (časť 1)

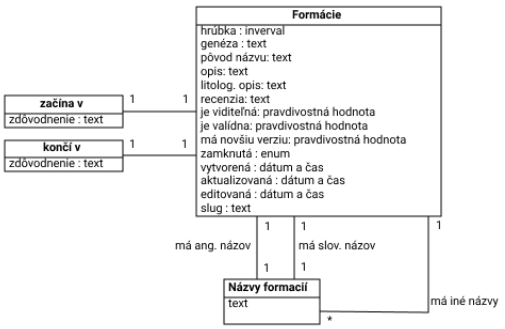


«enum» označuje dáta, ktoré sa často nemenia a pri editácii formácie sa vyberajú zo zoznamu

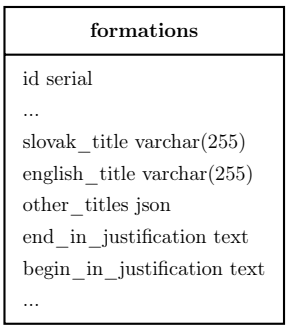
Obr. 4.9: Dátový model (časť 2)

Transformácia entitno-relačného modelu na relačný model

Pri transformácii entitno-relačného modelu na relačný model sme všetky množiny entít a ich atribúty preložili do anglického jazyka. Ako vieme z kapitoly 2, Laravel nám ponúka možnosť vytvárania databázových schém pomocou migrácií, kde je rovnako dobrým zvykom uvádzať názvy tabuliek a ich atribútov v anglickom jazyku. V prípade tabuliek sa doporučuje uvádzať ich názvy v množnom čísle. Pre zjednodušenie implementácie podpory verziovania dát v aplikácii sme sa rozhodli pre transformáciu vzťahov medzi množinami entít *Formácie* a množinami entít *začína v / končí v*, vzťahov *má ang. názov* a *má slov. názov*, ktoré sú v kardinalite *1 : 1* a vzťahu *má iné názvy*, ktorý je v kardinalite *1 : n* na atribúty tabuľky *formations*. V prípade vzťahu *má iné názvy* sme sa rozhodli použiť dátový typ *json*, vďaka ktorému vieme ukladať iné názvy priamo v riadku formácie a nepotrebujeme ich získavať z inej tabuľky. Spomenutú transformáciu môžeme vidieť na obrázkoch 4.10a a 4.10b.



(a) Entitno-relačný model



(b) Relačný model

Obr. 4.10: Transformácia entitno-relačného modelu na relačný model

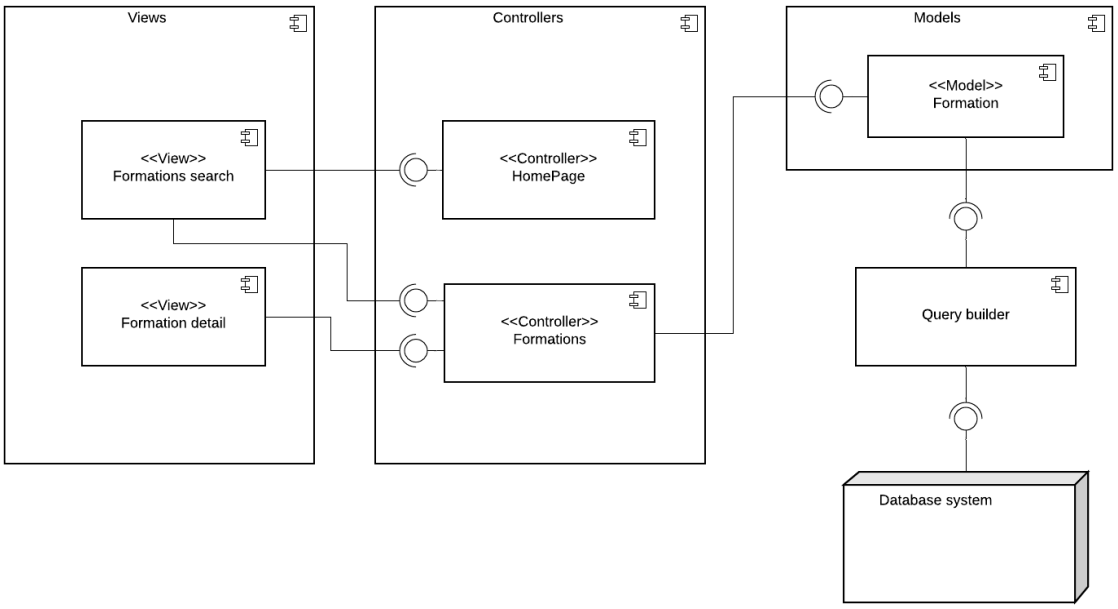
Spomenutá transformácia nám zjednodušila implementáciu vytvárania viacerých verzií formácie. Ak by sme pri spomenutých vzťahoch použili metódu transformácie na samostatné tabuľky, tak by sme pri vytváraní novšej verzie formácie museli duplikovať aj naviazané záznamy a ich atribúty. Pri našom riešení stačí duplikovať iba atribúty konkrétneho záznamu v tabuľke *formations*, čo je časovo efektívnejšie riešenie. Zvyšné časti entitno-relačného modelu sme transformovali podľa štandardných pravidiel na transformáciu entitno-relačného modelu na relačný model.

4.2 Štruktúra kódu

Po transformácii entitno-relačného modelu na relačný model si bližšie popíšeme štruktúru kódu. Kód aplikácie je rozdelený na dve hlavné časti a to administračnú časť a časť pre neprihlásených používateľov. Kód dodržiava princípy architektúry MVC (Model, View, Controller), kde sa *Model* stará o reprezentáciu dát pomocou tried, *View* o prezentáciu dát pre používateľov a *Controller* o logiku aplikácie a komunikáciu medzi *Modelom* a *View*.

Časť pre neprihlásených používateľov

V tejto časti práce popíšeme architektúru časti aplikácie pre neprihlásených používateľov. Architektúru tejto časti aplikácie môžeme vidieť na diagrame komponentov, ktorý sa nachádza na obrázku 4.11.



Obr. 4.11: Diagram komponentov pre časť aplikácie sprístupnenej pre neprihlásených používateľov

V komponente *Views* sa nachádzajú podkomponenty *Formations search* a *Formations detail*, prostredníctvom ktorých používateľ pracuje s aplikáciou. Hlavným účelom týchto komponentov je poskytovanie možnosti vyhľadávania formácií a zobrazovania ich detailu. V detaile sa nachádzajú všetky potrebné informácie o danej formácii. Spomenuté komponenty sa nachádzajú v súboroch *listing.js* a *detail.js*, ktoré sú súčasťou adresára *resources/js/rontend/formation*. Ide o javascriptové komponenty, ktoré sme vytvorili pomocou Vue.js. Na obrázku 4.12 môžeme vidieť štruktúru komponentu *formations-index* :

```
Vue.component('formations-index', {
    data() {
        formations: [],
        ...
    },
    methods: { getFormations(url) {...}},
    filters: { datetime: function (...) {...}},
});
```

Obr. 4.12: Štruktúra komponentu formations-index

Účelom komponentu *formations-index* je poskytnúť vyhľadávanie a výpis príslušných výsledkov o formáciách, ktoré sa dajú vyhľadávať na základe názvu alebo geologickej jednotky. V atribúte *data* sa nachádzajú všetky potrebné dáta s ktorými komponent pracuje. V prípade komponentu *formations-index*, to je napríklad pole *formations* v ktorom sa nachádzajú informácie o všetkých vyhľadaných formáciách na základe zadanej frázy do vyhľadávača. V atribúte *methods* sa nachádzajú všetky metódy, ktoré daný komponent poskytuje. V prípade komponentu *formations-index*, to je napríklad metóda *getFormations(url)*, ktorá odošle HTTP GET požiadavku na url adresu zadefinovanú v parametri funkcie. V prípade úspešnej odpovede zapíše dáta z odpovede do poľa *formations*. Následne dáta z tohto poľa zobrazí používateľovi. V prípade zlyhania zobrazí chybovú hlášku. V atribúte *filters* sa nachádzajú zadefinované filtre, ktoré komponent využíva. V prípade komponentu *formations-index* to je filter *datetime*, ktorý používateľovi zobrazuje systémové dátumy v čitateľnejšej podobe. Takýchto atribútov môže mať Vue.js komponent viacero, ale v našom prípade ich nevyužívame. Zoznam a účel všetkých dostupných atribútov sa nachádza v oficiálnej dokumentácii Vue.js. Pre lepšiu čitateľnosť kódu používame pre zadefinovanie používateľského rozhrania komponentu princíp *inline-template*. Vďaka tomuto princípu nemusíme písať prezentačný kód komponentu priamo v súbore, kde je zadefinovaný, ale v súboroch kde ho využívame. V našom prípade to sú *blade* súbory ktoré, ako z kapitoly 2 vieme, rozširujú html o možnosti použitia PHP kódu a vlastných konštrukcií. Komponent *formations-index* používame v

súbore *resources/views/frontend/homepage/index.blade.php* a komponent *formations-detail* v súbore *resources/views/frontend/formation/detail.blade.php*. Štruktúra *blade* súborov pre časť aplikácie sprístupnenej pre neprihlásených používateľov sa nachádza na obrázku 4.13.

```
@extends('frontend.layout.main')
@section('content')
<formations-index
    v-cloak
    inline-template>
    <div>
        ...
    </div>
</formations-index>
@endsection
```

Obr. 4.13: Štruktúra *blade* súboru s použitím komponentu formations-index

Vďaka atribútu *v-cloak* sa dáta o formáciách zobrazia až v čase, keď sú načítané, čo nám zabezpečuje, že používateľ neuvidí aplikáciu v nepožadovanom stave. Z obrázka 4.11 môžeme vidieť, že komponent *Views* využíva funkcionality, ktoré ponúka komponent *Controllers*. Tieto komponenty spolu komunikujú pomocou protokolu HTTP. Ako príklad uvedieme vyhľadávanie formácií. Z komponentu *Views* sa po stlačení tlačidla *Vyhľadať* odošle HTTP požiadavka na získanie všetkých formácií, ktoré vyhovujú zadanej fráze vo vyhľadávači. Na túto požiadavku odpovie komponent *Controllers* príslušnou HTTP odpoveďou, v ktorej sa nachádzajú nájdené formácie. V komponente *Controllers* sa nachádzajú podkomponenty *HomePage* a *Formations*. Tieto komponenty reprezentujú triedy *HomePageController* a *FormationsController*, ktoré sa nachádzajú v mennom priestore *App\Http\Controllers\Frontend*. Kostru týchto tried si ilustračne predstavíme na triede *FormationsController*, ktorá sa nachádza na obrázku 4.14.

```
class FormationsController extends Controller {
    public function formations(Request $request) {...}
    public function detail(string $formationSlug) {...}
}
```

Obr. 4.14: Kostra triedy FormationsController pre časť aplikácie sprístupnenej neprihláseným používateľom

Metóda *formations(Request $request)* slúži na vyhľadávanie formácií na základe

zadanej frázy do vyhľadávača. Táto metóda má parameter *$request* s kľúčom *formationInput*, ktorého hodnota je vyhľadávaná fráza. Pomocou databázového rozšírenia *unaccent* a operátora *ilike* sa vyhľadajú všetky formácie, ktorých názvy alebo geologické jednotky vyhovujú zadanej fráze.

Metóda *detail(string $formationSlug)* slúži na zobrazenie detailu formácie na základe požadovaného *slugu*, ktorý sa nachádza v url adrese detailu. Ak zadanému *slugu* nevyhovuje žiadna formácia, táto metóda vyvolá výnimku *FormationNotFoundException*, čo sa používateľovi zobrazí, ako stránka chybovej hlášky s HTTP kódom *404 not found*.

Ďalej môžeme z obrázka 4.11 vidieť, že komponent *Controllers* využíva funkcionalitu, ktorú ponúka komponent *Models*. V tomto komponente sa nachádza podkomponent *Formation*, ktorý reprezentuje trieda *Formation* v mennom priestore *App\Models*. Táto trieda reprezentuje množinu entít *Formácie*. Základná štruktúra tejto triedy bola vygenerovaná nástrojom Craftable o čom si viac povieme v ďalších častiach tejto kapitoly. Každý databázový dopyt sa vykonáva pomocou triedy modelu a *Query builder*, ktorý komunikuje s databázovým systém, ako to môžeme vidieť na obrázku 4.11. Kostra triedy *Formation*, ktorá je pre časť aplikácie sprístupnenej pre neprihlásených používateľov dôležitá, je znázornená na obrázku 4.15.

```
class Formation extends Model {
    use CreatedByAdminUserTrait;

    public function getContributorsAttribute(){...}

    public function area() {
        return $this->belongsTo(Area::class);
    }

    public function scopeVisible($query){...}
    ...
}
```

Obr. 4.15: Kostra triedy Formation pre časť aplikácie sprístupnenej neprihláseným používateľom

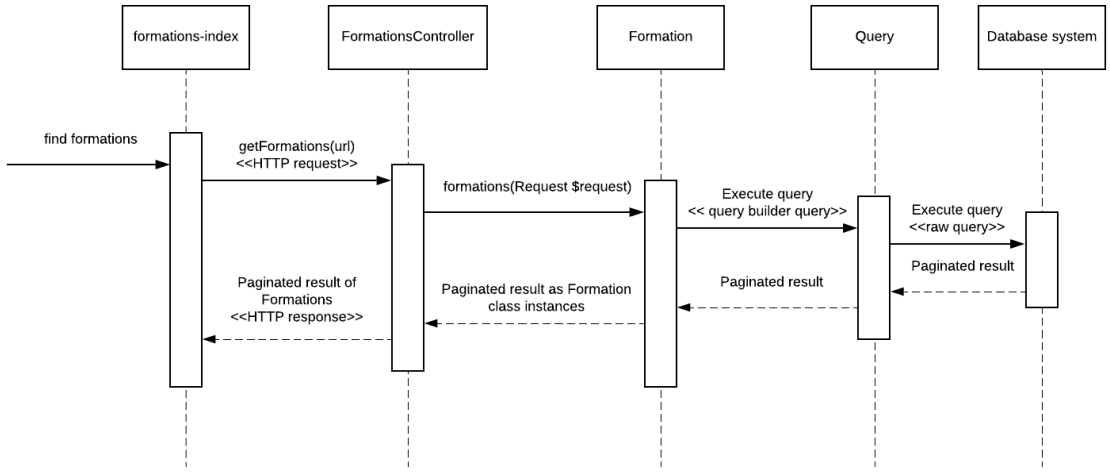
V *traite CreatedByAdminUserTrait* sa nachádza definícia vzťahu naviazaného na používateľa, ktorý danú formáciu vytvoril. Tento *trait* je sprostredkovaný nástrojom Craftable.

Metóda *getContributorsAttribute()* ponúka informácie o používateľoch, ktorí pracovali na danej formácii. Využitie tejto metódy sme mohli vidieť na obrázku 3.2 pri

výsledkoch vyhľadávania.

Metóda *area()* je príklad zadefinovania databázového vzťahu, vďaka ktorému sa vieme dostať k atribútom oblasti v ktorej sa formácia nachádza.

Úlohou metódy *scopeVisible($query)* je filtrovanie dát na databázovej úrovni. V našom prípade chceme zobrazovať a vyhľadávať iba v dátach najaktuálnejších schválených verzií formácií. Komunikáciu medzi jednotlivými komponentmi môžeme vidieť na sekvenčnom diagrame, ktorý popisuje proces vyhľadávania formácií a nachádza sa na obrázku 4.16.



Obr. 4.16: Sekvenčný diagram pre vyhľadávanie formácií

Z pohľadu adresárov je kód rozdelený do nasledujúcich celkov:

app/Http/Controllers/Frontend: V tomto adresári sa nachádzajú triedy kontrolérov, ktoré sa starajú pomocou ich implementovaných metód o zobrazovanie úvodnej stránky, vyhľadávanie formácií a zobrazovanie detailu vybranej formácie.

app/Models: V tomto adresári sa nachádzajú triedy, ktoré reprezentujú množiny entít uložené v databázovom systéme. V týchto triedach sú zadefinované aj databázové vzťahy na ostatné triedy v tomto adresári podľa kardinalít vzťahov.

routes: V tomto adresári sa nachádza súbor *web.php*, v ktorom sú zadefinované url adresy, prostredníctvom ktorých používateľ komunikuje s aplikáciou.

resources/js/frontend: V tomto adresári sa nachádzajú Vue.js komponenty pre vyhľadávanie a detail formácie. Táto časť aplikácie využíva knižnice Vue.js, Bootstrap, VeeValidate a Leaflet.

resources/views/frontend: V tomto adresári sa nachádzajú *blade* súbory, ktoré sú spolu s Vue.js komponentmi poskytované používateľom. Tieto komponenty slúžia na prezentáciu dát a používateľského rozhrania, prostredníctvom ktorého používateľ pracuje s časťou aplikácie pre neprihlásených používateľov.

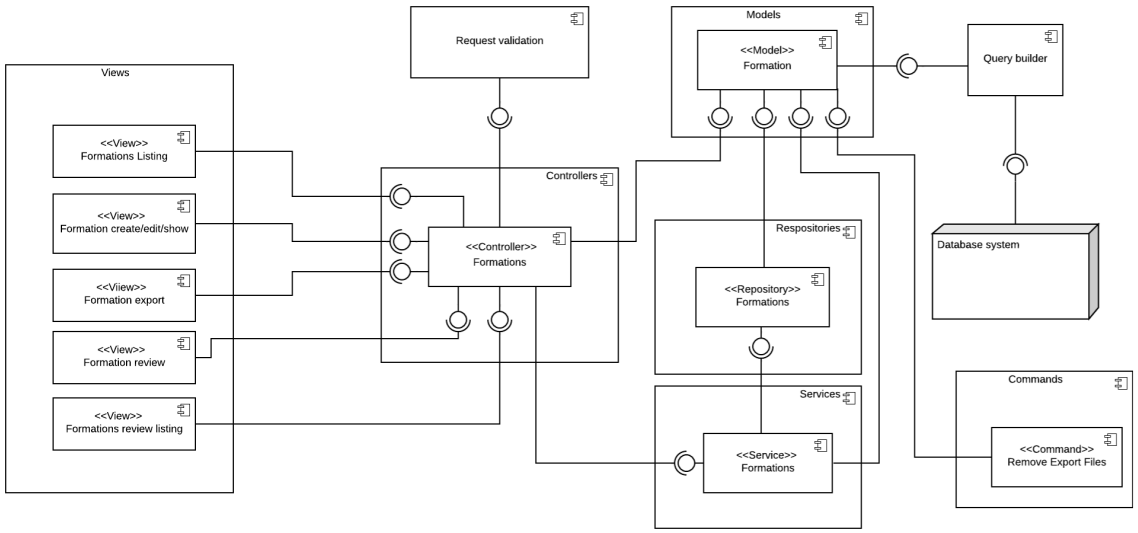
resources/images/frontend: V tomto adresári sa nachádzajú obrázky použité v časti aplikácie pre neprihlásených používateľov.

resources/sass/frontend: V tomto adresári sa nachádzajú *.scss* súbory pomocou ktorých sme zadefinovali vlastné štýly aplikácie. Výhodou použitia *scss* je možnosť zadefinovania premenných, ktoré môžu byť v prípade potreby zmenené iba na jednom mieste (napríklad, ak by sme sa rozhodli zmeniť paletu farieb aplikácie).

webpack.mix.js: V tomto súbore sa nachádza konfigurácia nástroja webpack pomocou Laravel API *laravel-mix*. Jeho účelom je napríklad komplilácia .scss súborov do .css súborov a minifikácia javascriptových súborov. Vďaka minifikovaným súborom sa naša aplikácia načítava používateľom rýchlejšie. V našom prípade slúži webpack aj na transpilovanie javascriptového kódu napísaného vo verzii ES6 do staršej verzie ES5, ktorá je podporovaná väčšinou dnešných webových prehliadačov.

Administračná časť

Po predstavení štruktúry a funkcionality kódu časti aplikácie sprístupnenej pre neprihlásených používateľov sa pozrieme na architektúru administračnej časti aplikácie. Ako z požiadaviek na aplikáciu zadefinovaných v kapitole 1 vieme, že administračná časť aplikácie je prístupná iba prihláseným používateľom a jej funkcionality sú používateľom sprístupnené na základe priradenej role. Základná štruktúra a funkcionalita bola vygenerovaná pomocou nástroja Craftable na základe atribútov množín entít z dátového modelu. Vygenerovanie základnej štruktúry a funkcionality kódu nám umožnilo sa lepšie zamerať na logiku aplikácie, splnenie všetkých požiadaviek zadefinovaných v kapitole 1, jednotkové testy, čitateľnosť kódu a modulárnosť aplikácie. Pre komplexnosť aplikácie a funkcionalít, ktoré ponúka si architektúru administračnej časti aplikácie znázornime na množine entít *Formácie*. Architektúru môžeme vidieť na diagrame komponentov, ktorý sa nachádza na obrázku 4.17.



Obr. 4.17: Diagram komponentov pre administračnú časť aplikácie

V komponente *Views* sa nachádzajú všetky podkomponenty prostredníctvom ktorých, používateľ pracuje s aplikáciou. Tieto podkomponenty komunikujú s komponentom *Controllers*, v ktorom sa nachádza podkomponent *Formations* pomocou protokolu HTTP. HTTP požiadavky sa pomocou funkcionality, ktorú ponúka komponent *Request Validation* validujú. Komplexnejšia logika aplikácie je pre čitateľnosť a modulárnosť kódu rozdelená do funkcionalít, ktoré ponúkajú komponenty *Services* a *Repositories*. Komponenty *Controllers*, *Services* a *Repositories* využívajú funkcionality, ktoré ponúka komponent *Models*. Tento komponent spolu s komponentom *Query builder* komunikuje s databázovým systémom. Účelom komponentu *Commands* je sprostredkovanie vlastných príkazov v aplikácii. Pre lepšiu ilustráciu funkcionality a štruktúry kódu sa spolu pozrieme na proces schvaľovania dát.

Ako z požiadaviek na aplikáciu vieme, všetky formácie musia prejsť procesom schvaľovania dát recenzentom. Formácia sa nachádza v procese schvaľovania, ak má nastavenú hodnotu atribútu *locked* na *in_review*. Do tejto hodnoty sa atribút *locked* nastaví v prípade, ak ju tvorca odošle na schválenie. Pre zobrazenie formulára pre proces schvaľovania dát, kde sa nachádza porovnanie aktuálnej neschválenej verzie dát o formácii s jej predošlou schválenou verziou, sme vytvorili Vue.js komponent s názvom *formation-review-form*. Tento komponent sa nachádza v adresári *resources/js/admin/formation*. a jeho štruktúra sa nachádza na obrázku 4.18.

```
Vue.component('formation-review-form', {
    mixins: [AppForm],
    props: [
        'previousVersion',
    ],
    data: function() {
        return {
            form: {
                depth: '',
                genesis: '',
                ancestry_of_title: '',
                description: '',
                ...
            }
        }
    },
    methods: {
        setLockedRejected() {...},
        ...
    }
});
```

Obr. 4.18: Štruktúra komponentu formation-review-form

V komponente *formation-review-form* používame funkcionality z komponentu *AppForm*, ktorý je súčasťou Craftable a nachádza sa v každom komponente aplikácie, ktorý slúži ako formulár. V komponente *AppForm* sa nachádza konfigurácia wysiwyg editora, konfigurácia elementu pre nahrávanie súborov a podobne. V komponente *formation-review-form* používame atribút *props* do ktorého si posielame predošlú schválenú verziu formácie. V atribúte *data* si v objekte *form* uchovávame informácie o formácii odoslanej na schválenie. Ako príklad si môžeme uviesť atribút *depth*, ktorý hovorí o hrúbke formácie. Dôvodom použitia atribútu *props* pre predošlú schválenú verziu je ten, že všetky dáta uložene v atribúte *props* sa nedajú meniť. Narozdiel od verzie odoslanej na schválenie, kde chceme meniť jej status na schválený alebo zamietnutý. O zmenu statusu sa stará napríklad metóda *setLockedRejected()*, ktorá sa nachádza v atribúte methods a recenzovanej formácie nastaví atribút *locked* na hodnotu *rejected*. Pre zadefinovanie prezentačného kódu tohto komponentu sme použili princíp *inline-template*. Prezentačný kód komponentu sa nachádza v *blade* súbore *review-form.blade.php*, ktorý je súčasťou adresára *resources/views/admin/formation*. Štruktúra spomenutého *blade* súboru sa nachádza na obrázku 4.19.

```
@extends('brackets/admin-ui::admin.layout.default')
@section('title', ...)
@section('body')
    <div class="container-xl-12">
        <formation-review-form
            :action="'{{ ... }}'"
            :data="{{ $formation->toJson() }}"
            :previous-version="{{ $previousVersion->toJson()}}"
            v-cloak
            inline-template
            >

            <form method="post" ...>
                <div class="...">
                    ...
                </div>
                ...
            </form>

        </formation-review-form>
    </div>
@endsection
```

Obr. 4.19: Štruktúra *blade* súboru s použitím komponentu formation-review-form

Tento *blade* súbor rozširuje základný štýl zadefinovaný v balíčku *admin-ui*, ktorý je súčasťou Craftable a je v ňom zadefinovaná hlavička, miesto, kde sa má umiestniť navigácia, pätička a miesto, kde sa majú rozšírené *blade* súbory zobrazovať. Ako môžeme ďalej z obrázka 4.19 vidieť, do Vue.js komponentu *formation-review-form* posielame hodnoty do atribútu *props* pomocou dvojbodky. Ako príklad si uvedieme preposlanie dát o predošlej schválenej formácii, ktorú do komponentu posielame pomocou syntaxe *:previous-version="{{ $previousVersion->toJson()}}"*. Výraz, ktorý sa nachádza v zložených zátvorkách, je štandardný PHP kód, ktorý v našom prípade prevedie údaje z premennej $previousVersion do formátu *json*. Po ukončení recenzie sa odošle HTTP požiadavka, ktorú spracováva metóda *reviewUpdate(ReviewFormation $request, Formation $formation)*, nachádzajúca sa v triede *FormationsController*. Trieda *FormationsController* sa nachádza v mennom priestore *App\Http\Controllers\Admin*. Kostru metódy *reviewUpdate(ReviewFormation $request, Formation $formation)* môžeme vidieť na obrázku 4.20.

```
public function reviewUpdate(...)
{
    $sanitized = $request->getSanitized();
    DB::transaction(function ... {
        'SET TRANSACTION ISOLATION LEVEL SERIALIZABLE';
        $this
            ->formationsService
            ->checkIfUserCanReviewFormation($formation);

        if ($sanitized['locked']
         === Formation::FORMATION_TYPE_APPROVED
        && $formation->previousVersion) {
            $this
            ->formationsService
            ->changeParentInChildFormations...(...);
        }
        $formation->update($sanitized);
    });
    ...
    return redirect('admin/formations');
}
```

Obr. 4.20: Kostra metódy reviewUpdate

Ako môžeme z kostry metódy vidieť, dáta v HTTP požiadavke sa najprv validujú a dajú do požadovaného formátu pomocou metódy *getSanitized()*, ktorá je súčasťou triedy *ReviewFormation*. Kostra triedy *ReviewFormation* sa nachádza na obrázku 4.21.

```
class ReviewFormation extends FormRequest
{
    public function authorize(){...}
    public function rules(): array {
        return [
            'review' => ['sometimes', 'nullable'],
            'locked' => ['required',
                Rule::in(Formation::getStatusesForReview()),
            ],
        ];
    }
    public function getSanitized(): array {
        $sanitized = $this->validated();
        $sanitized['is_visible'] = $this->getIsVisible();
        ...
    }
    public function getIsVisible(): bool {
        return $this->isLockedAndApproved();
    }
    ....
}
```

Obr. 4.21: Kostra triedy ReviewFormation

Trieda *ReviewFormation* kontroluje, či má používateľ dostatočné právomoci na recenzovanie a či sú atribúty pre recenzovanie formácie v HTTP požiadavke vo validných hodnotách. Povolené hodnoty pre atribút *locked* vieme získať zo statickej metódy *getStatusesForReview()*, ktorá sa nachádza v triede *Formation*. Povolené hodnoty pre recenzovanie sú *approved* a *rejected*. Ak by sa v HTTP požiadavke poslal iný status ako sme spomenuli, táto trieda by vyhodnotila, že dáta nie sú validné. Ďalej sa v tejto triede nachádza metóda *getSanitized()*, ktorej úlohou je vyskladať dáta z HTTP požiadavky do formátu, ktorý si kontrolér vyžaduje. Ako príklad si môžeme uviesť nastavenie atribútu *is_visible*, ktorý sa nastaví na pravdu, ak je status formácie hodnota *approved*. Tento atribút overuje metóda *getIsVisible()*. Ak sú dáta validné a máme ich v požadovanom tvare, začneme databázovú transakciu s úrovňou izolovanosti *SERIALIZABLE* ako to môžeme vidieť na obrázku 4.21. V tejto transakcii overíme, či nie sú dáta zamknuté pomocou metódy *checkIfUserCanReviewFormation($formation)*, ktorá je súčasťou triedy *FormationsService*. Viac o zamykaní dát povieme v samostatnej časti tejto práce. Ak je recenzovaná formácia schválená, chceme podformáciam pôvodnej verzie formácie zmeniť vzťah rodiča na formáciu, ktorú práve recenzent schválil. O túto funkcionalitu sa

stará metóda *changeParentInChildFormationsToNewerApprovedFormation(Formation $formation)* v triede *FormationsService*. Táto metóda v triede *FormationsService* pre lepšiu čitateľnosť a štruktúru kódu volá metódu s rovnakým názvom v triede *FormationsRepository*. Kostra metódy *changeParentInChildFormationsToNewerApprovedFormation(Formation $formation)* v triede *FormationsRepository* sa nachádza na obrázku 4.22.

```
public function changeParentInChildFormations...(...)
{
    $formation->previousVersion->update([
        'is_visible' => false
    ]);

    if ($formation->previousVersion->childFormations !== null) {
        $formation->previousVersion->childFormations
        ->each(static function ($childFormation) use ($formation) {
            if ($childFormation->newerVersion == null) {
                $childFormation->parent_id = $formation->id;
                $childFormation->save();
            }
        });
    }
}
```

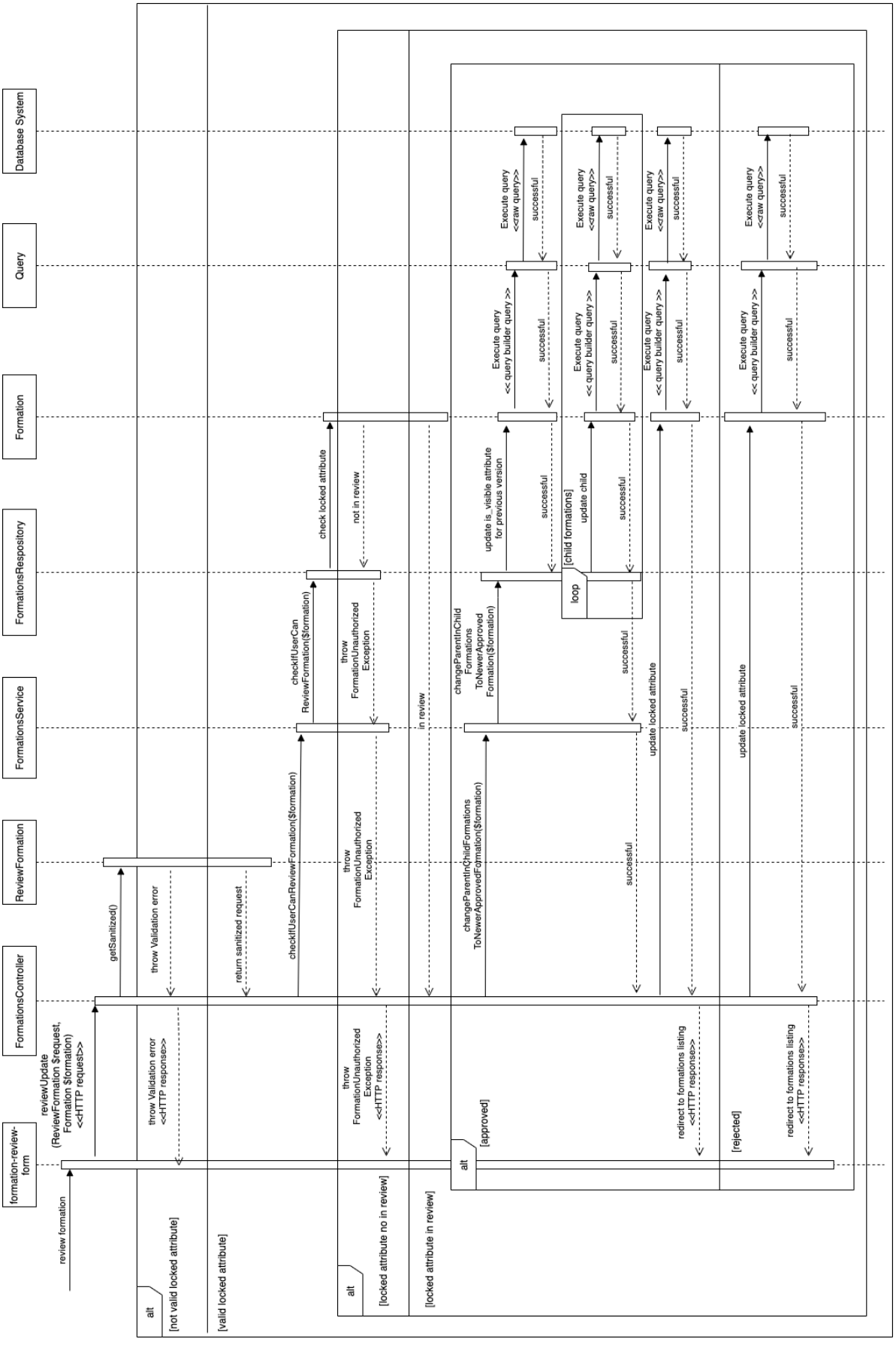
Obr. 4.22: Kostra metódy changeParentInChildFormationsToNewerApprovedFormation

Táto metóda najprv predošlej verzii formácie nastaví atribút *is_visible* na nepravdu a následne prejde všetky podformácie a zmení im atribút *parent_id* na identifikátor recenzentom schválenej formácie. V prípade zamietnutia formácie zmeníme atribút *locked* na hodnotu *rejected* čo spôsobí, že sa táto formácia zobrazí používateľovi, ktorý ju odoslal na schválenie, ako zamietnutá. V prípade komentára k zamietnutiu aktualizujeme aj atribút *review*.

Komunikáciu medzi triedami a komponentmi, ktoré sa podieľajú na procese schvaľovania dát, môžeme vidieť na sekvenčnom diagrame, ktorý sa nachádza na obrázku 4.23.

Z pohľadu adresárov je kód rozdelený do nasledujúcich celkov:

app/Console: V tomto adresári sa nachádzajú všetky nami zadefinované CLI (Commnad line interface) príkazy aplikácie. Tieto príkazy vieme spúšťať z konzoly pomocou príkazu *php artisan názov_nášho_príkazu*. Ďalej sa tu nachádza trieda *Kernel* v ktorej



Obr. 4.23: Sekvenčný diagram pre proces schvaľovania formácie

sú nami zadefinované príkazy importované a vďaka tejto triede máme umožnené spúšťanie našich príkazov z konzoly. V tejto triede sa nachádzajú aj volania plánovaných úloh alebo príkazov s definíciou pravidelnosti ich vykonávania.

app/Exceptions: V tomto adresári sa nachádzajú všetky nami zadefinované výnimky, ktoré sa môžu v aplikácii vyvolať. Trieda *Handler* je poskytovaná Laravelom a jej účelom je odchytávanie všetkých výnimiek aplikácie. V tejto triede môžeme zadefinovať, ktoré výnimky sa nemajú zapisovať do systémového logu, alebo nastavenie odosielania informácií o výnimkách do externej služby, akou môže byť napríklad nástroj *Bugsnag*.

app/Http/Controllers: V tomto adresári sa nachádzajú všetky triedy kontrolérov aplikácie. Ako už vieme, účelom týchto tried je vykonávanie logiky aplikácie a komunikácia medzi *Modelom* a *View*. Štruktúru podadresára *Frontend* sme už bližšie popísali v časti štruktúry kódu pre neprihlásených používateľov. V podadresári *Backend* sa nachádzajú kontroléry, ktorých základná štruktúra metód pre operácie vytvárania, hromadného výpisu, editácie a mazania bola vygenerovaná pomocou Craftable. My sme túto štruktúru metód rozšírili o podporu databázových vzťahov, vykonávanie viacerých databázových operácií v databázových transakciách, verziovanie, vyňatie komplexnejších častí kódu do servisov a repozitárov, proces schvaľovania dát, prezeranie vybraného záznamu tabuľky, export a import ako aj o všetky požadované funkcionality, ktoré sme si popísali v kapitole 1. V podadresári *Auth* sa nachádzajú triedy pre autentifikáciu používateľov mimo administračnej časti aplikácie, ktoré boli sprostredkované Laravelom. V našom projekte takúto autentifikáciu nevyužívame. O autentifikáciu používateľov administračnej časti aplikácie sa stará balíček *admin-auth*, ktorý je súčasťou Craftable. Triedy pre kontroléry v podadresári *Backend* sa riadia štruktúrou *Resource Controllers*, ktorá je spomenutá aj v oficiálnej dokumentácii Laravelu [6] a ich štruktúru môžeme vidieť na obrázku 4.24.

```
class ...Controller extends Controller {
    public function index(){...}
    public function create(){...}
    public function store(){...}
    public function show(...){...}
    public function edit(...){...}
    public function update(...){...}
    public function destroy(...){...}
}
```

Obr. 4.24: Štruktúra Resource Controllers podľa oficiálnej dokumentácie Laravelu [6]

app/Http/Middleware: V tomto adresári sa nachádzajú všetky triedy midlvérov, ktorých účelom je filtrovanie HTTP požiadaviek vstupujúcich do aplikácie. Tieto midlvéri sú sprostredkované Laravelom. Príkladom použitia midlvéru je napríklad trieda *VerifyCsrfToken*, ktorá poskytuje ochranu proti CSRF zraniteľnosti. Podľa [18] je CSRF typ útoku, kde útočník vykonáva nechcené akcie prostredníctvom používateľa, ktorý je na danej webovej aplikácii prihlásený. Pomocou týchto akcií môže prísť napríklad k zmene emailu a hesla prihláseného používateľa. V prípade ak CSRF token v HTTP požiadavke chýba, alebo sa token v požiadavke od používateľa nezhoduje s tokenom uloženým v sedení (z anglického session) vyhodí sa výnimka *TokenMismatchException*.

app/Http/Requests: V tomto adresári sa nachádzajú všetky triedy pre validáciu a úpravu vstupov od používateľa, ktoré prichádzajú v HTTP požiadavkách. Základnú štruktúru týchto tried nám vygeneroval Craftable a v našom prípade sme ju rozšírili o validáciu informácií o databázových vzťahoch, aplikačnú logiku (napríklad zamietnutie zmeny statusu formácie na nepovolenú hodnotu), ako aj o vyskladanie správnej štruktúry dát, pre účely metód v kontroléroch, nakoľko dáta o vzťahoch z používateľského rozhrania prichádzajú v inom formáte. V prípade nevalidnosti dát sa používateľovi vráti HTTP status *422 Unprocessable Entity* s výpisom, aké časti dát nie sú vo validnom formáte.

app/Imports: V tomto adresári sa nachádzajú triedy pre správnu funkčnosť importov. Viac o importoch si povieme v samostatnej časti tejto práce.

app/Models: V tomto adresári sa nachádzajú triedy, ktoré reprezentujú množiny entít uložené v databázovom systéme. Základnú štruktúru týchto tried nám vygeneroval Craftable. Túto štruktúru sme rozšírili o podporu databázových vzťahov na základe ich kardinalít, metód na prístup k potrebným atribútom, konštánt a rozsahov databázových dopytov, vďaka ktorým je dopytovanie pre jednotlivé časti aplikácie z pohľadu kódu čistejšie, nakoľko, ak chceme zobrazovať dáta pre prihlásených používateľov, nemusíme písať rovnaký dopyt v potrebných častiach kódu znova a opakovať sa, ale využiť dopytovanie pomocou nami vytvorených rozsahov databázových dopytov, kde sú tieto dopyty zabalené. V podadresári *Traits* sa nachádzajú špeciálne triedy, ktoré sa v PHP označujú ako *trait* a obsahujú časti kódu, ktoré sa používajú vo viacerých Modeloch za účelom predchádzaniu duplicitám v kóde.

app/Providers: V tomto adresári sa nachádzajú triedy poskytovateľov, ktorých účelom je registrácia midlvérov, url adries a podobne pri vytváraní (z anglického bootstrapping) aplikácie. Tieto triedy sú sprostredkované Laravelom.

app/Services: V tomto adresári sa nachádzajú triedy, ktoré ponúkajú komplexnejšie funkcie pre kontroléry. Tento adresár a jeho triedy sme vytvorili pre lepšiu čitateľnosť a modularitu kódu. Metódy týchto tried voláme v príslušných kontroléroch pomocou techniky Injektovania Závislostí (z anglického Dependency Injection).

app/Respositories: V tomto adresári sa nachádzajú triedy a metódy pre komplexnejšiu logiku aplikácie. Tieto triedy a ich metódy voláme v príslušných Servisoch pomocou techniky Injektovania Závislostí (z anglického Dependency Injection).

config: V tomto adresári sa nachádzajú konfiguračné súbory aplikácie a knižníc, ktoré používame. Ako príklad si môžeme uviesť súbor *locking.php* v ktorom sa nachádza kľúč *locked_time*, ktorého hodnota hovorí o čase v sekundách, ako dlho má byť formácia zamknutá pri akcii editácie. Ako príklad konfiguračného súboru sprostredkovaného Laravelom si môžeme uviesť súbor *database.php*, ktorý si v sebe uchováva informácie o databázovom spojení.

docker, harbor a docker-compose.yml: Adresár docker a súbory harbor a docker-compose.yml nám sprostredkoval Harbor a nachádzajú sa v nich všetky potrebné súbory, informácie a skripty pre spúšťanie docker kontajnerov prostredníctvom harboru. V súbore docker-compose.yml si napríklad môžeme zmeniť verziu jazyka PHP používanej v aplikácii. Podporované verzie jazyka PHP môžeme nájsť v adresári *docker/php*.

node_modules: Adresár node_modules obsahuje všetky balíčky nainštalované pomocou balíčkovacieho systému npm.

resources: Tento adresár obsahuje všetky obrázky, neskompilované scss súbory, javascriptový kód a komponenty, prezentačný kód v *blade* súboroch a multijazykové súbory pre preklady aplikácie. Základná štruktúra a funkcionalitu *blade* a javascriptových súborov nám vygeneroval Craftable. Túto štruktúru sme rozšírili o vlastné štýly, nové komponenty pre proces schvaľovania dát, podporu vyberania dát o vzťahoch vo formulároch aplikácie zo zoznamu a podobne. Do multijazykových súborov sme pridali nové kľúče a hodnoty podľa potrieb funkcionality. Ako príklad si môžeme uviesť preklady pre importovanie stratigrafických intervalov.

routes: V tomto adresári sa nachádza súbor *web.php*, v ktorom sú zadefinované url adresy, prostredníctvom ktorých používateľ komunikuje s aplikáciu. Nakoľko je administračná časť aplikácie sprístupnená iba prihláseným používateľom url adresy sú chránené pomocou midlvéru, ktorý overuje, či je používateľ prihlásený. Ak používateľ nie je prihlásený midlvér ho presmeruje na obrazovku prihlasovacieho formulára, ktorú sme

mohli vidieť na obrázku 3.5. Adresy pre základné operácie vytvárania, hromadného výpisu, editácie a vymazávania vygeneroval Craftable. Do tejto vygenerovanej štruktúry sme pridali url adresy pre export formácií, proces schvaľovania, nastavovanie formácii informáciu o jej validnosti, importovanie intervalov, prezerania konkrétneho záznamu danej tabuľky a podobne. V tomto adresári sa taktiež nachádzajú súbory *api.php, console.php, channels.php*, ktoré sú sprostredkované Laravelom. Súbor *api.php* sa používa v prípade, ak aplikácia poskytuje api rozhranie.

storage: V tomto adresári sa nachádzajú všetky súbory medií, vyrovnávacej pamäte, sedení (z anglického sessions) a systémových logov aplikácie.

tests: V tomto adresári sa nachádzajú jednotkové testy. Viac o jednotkových testoch povieme v samostatnej časti tejto práce.

vendor: Tento adresár obsahuje všetky balíčky nainštalované pomocou balíčkovacieho systému composer.

.env: V tomto súbore sa nachádzajú všetky informácie o prostredí na ktorom je aplikácia spúšťaná. V tomto súbore môžeme nájsť, napríklad údaje o databáze, docker konfigurácii portov, nastavení mailového klienta a podobne. Tento súbor sa nepridáva do systému na riadenie verzií akým je napríklad git. V štruktúre projektu sa nachádza súbor .env.example, v ktorom sa nachádzajú všetky potrebné kľúče a používateľ si tieto kľúče skopíruje, vytvorí súbor s názvom .env a tieto kľúče vyplní v závislosti od prostredia na ktorom sa aplikácia spúšťa. Tento súbor pridávame aj do gitu.

phpunit.xml: V tomto súbore sa nachádza konfigurácia pre jednotkové testy aplikácie.

4.3 Zamykanie dát

V tejto časti práce popíšeme proces zamykania dát, ktorý vychádza z požiadavky uvedenej v kapitole 1, ktorá hovorí, že počas doby schvaľovania dát má byť formácia uzamknutá pre akciu editácie. Predtým ako popíšeme nami implementované riešenie, uvedieme pojem aplikačné transakcie.

4.3.1 Aplikačné transakcie

Podľa [17] sú aplikačné transakcie také transakcie, ktoré trvajú príliš dlho na to, aby im boli pridelené zámky, ktoré potrebuje iná transakcia. V závislosti od konkrétneho

použitia môže príliš dlho znamenať minúty, hodiny, dni, dokonca týždne. Aplikačné transakcie sa zväčša skladajú z viacero menších databázových transakcií, ktoré nám zabezpečujú ACID vlastnosti (Atomicita, Konzistentnosť, Izolovanosť, Trvanlivosť). Aplikačné transakcie si musíme v aplikácii implementovať samostatne v závislosti od konkrétnej logiky požadovanej funkcionality. Pri implementácii aplikačných transakcií musíme zabezpečiť aj funkcionalitu v prípade zamietnutia transakcie *(abort)* alebo vrátenia databázy do pôvodného stavu *(roolback)*.

4.3.2 Implementácia zamykania dát

Z kapitoly 1 vieme, že počas doby schvaľovania dát má byť formácia uzamknutá pre akciu editácie. To však nie je jediný prípad, kedy si želáme aby boli dáta uzamknuté. Predstavme si situáciu ak by akciu editácie už schválenej formácie chcelo vykonať viacero používateľov. Nakoľko chceme povoliť vytvoriť iba jednu novšiu neschválenú verziu formácie, tak by mohlo jednoducho nastať, že by sa táto akcia editácie podarila iba používateľovi, ktorý by ukončil editáciu ako prvý. Ostatným používateľom by táto akcia padla na chybe integritného obmedzenia, čo nechceme dovoliť z dôvodu, aby daný používateľ nevypĺňal dáta zbytočne, nakoľko môže proces schvaľovania formácie trvať dlhšiu dobu (v prípade, ak sú časové kapacity recenzenta vyčerpané, to môže trvať dni až týždne) a v konečnom dôsledku by musel začať akciu editácie pre novšiu verziu dát. V takomto prípade chceme v čase, keď začne používateľ akciu editácie danú formáciu zamknúť.

Čo ale v prípade situácie ak používateľ, ktorý začal akciu editácie formácie túto akciu už nikdy neukončí? Pre riešenie popísaného problému sme sa rozhodli použiť aplikačné transakcie, v ktorých používame databázové transakcie s vhodnou úrovňou izolovanosti. Vždy ak chce používateľ začať akciu editácie dát formácie, metódy merania alebo profilu sa chceme pozrieť či nezačal túto akciu už iný používateľ. Implementáciu aplikačných transakcií pri procese zamykania popíšeme na množine entít *Formácie* a akcii editácie.

O proces editácie sa stará metóda *edit(Formation $formation)* v triede *FormationsController*, ktorá sa nachádza v mennom priestore *App\Http\Controllers\Admin* a jej účelom je používateľovi zobraziť formulár pre editáciu zvolenej formácie. Pre lepšiu čitateľnosť ukážeme zjednodušenú kostru tejto metódy, ktorá sa nachádza na obrázku 4.25.

```
public function edit(Formation $formation) {
    $this->authorize('admin.formation.edit', $formation);
    ...
    $response = DB::transaction(function () ...) {
        'SET TRANSACTION ISOLATION LEVEL SERIALIZABLE';
        $formationsService->checkIfUserCanEditFormation(...);
        $formation->lockForUpdate()->first();
        $formation->edited_at = Carbon::now();
        $formation->editedByAdminUser()->associate(Auth::user());
        $formation->save();
        ...
        return $formationsServise->getFormWithResponse(...);
    })
    return $response;
}
```

Obr. 4.25: Kostra metódy edit

Na začiatku metódy *edit(Formation $formation)* sa overí, či má používateľ dostatočné právomoci pre editáciu formácie. Následne začneme databázovú transakciu, ktorej nastavíme úroveň izolovanosti *SERIALIZABLE*. Dôvodom zvolenia tejto úrovne izolovanosti je, že nás chráni pred serializačnou anomáliou, čo nám zaručuje, že výsledok spustenia viacerých transakcií naraz (viac používateľov chce začať akciu editácie v rovnaký čas) bude taký, ako keby boli spustené za sebou v poradí. Metóda *checkIfUserCanEditFormation($formation)*, ktorá sa nachádza v triede *FormationsService* overuje, či nie je vybraná formácia zamknutá. Trieda *FormationsService* je v triede *FormationsController* použitá pomocou techniky Injektovania Závislostí (z anglického Dependency Injection). Zjednodušená kostra metódy *checkIfUserCanEditFormation($formation)* v triede *FormationsService* sa nachádza na obrázku 4.26.

```
public function checkIfUserCanEditFormation(...): void
{
    formationsRepository->checkIfUserCanEditFormation(...);
}
```

Obr. 4.26: Kostra metódy checkIfUserCanEditFormation v triede FormationsService

Ako môžeme z kostry vidieť, logika tejto metódy sa pre lepšiu čitateľnosť a štruktúru kódu nachádza v triede *FormationsRepository*. Trieda *FormationsRepository* je v triede *FormationsService* rovnako použitá pomocou techniky Injektovania Závislostí (z

anglického Dependency Injection). Zjednodušená kostra metódy *checkIfUserCanEditFormation($formation)* v triede *FormationsRepository* sa nachádza na obrázku 4.27.

```
public function checkIfUserCanEditFormation(...): void
{
    if ($formation->newerVersion !== null) {
        throw new FormationUnauthorizedException(...);
    }
    if ($formation->locked ===
    Formation::FORMATION_TYPE_IN_REVIEW) {
        throw new FormationUnauthorizedException(...);
    }
    if ($formation->locked !== Formation::FORMATION_TYPE_APPROVED
        && $formation->createdByAdminUser->id
        !== Auth::user()->id) {
        throw new FormationUnauthorizedException();
    }
    if ($formation->edited_at !== null) {
        if (editedAtIsLessThanLockedTimeFromConfig(...)
         && $formation->editedByAdminUser === null) {
            throw new FormationUnauthorizedException(...);
        }

        if (editedAtIsLessThanLockedTimeFromConfig(...)
        && $formation->editedByAdminUser !== null) {
            if ($formation->editedByAdminUser->id
            !== Auth::user()->id) {
                throw new FormationUnauthorizedException(...);
            }
        }
    }
    $this->checkIfUserCanEditForParentAndChildFormations(...);
}
```

Obr. 4.27: Kostra metódy checkIfUserCanEditFormation v triede FormationsRepository

Ako môžeme vidieť, táto metóda sa rozhoduje, či je daná formácia zamknutá, ak spĺňa jednu z nasledujúcich podmienok:

- ak má formácia novšiu verziu (aplikácia umožňuje vytváranie maximálne jednej novšej verzie pre konkrétny záznam v tabuľke),

- ak je formácia odoslaná na schválenie,
- ak formácia ešte nie je schválená a vytvoril ju iný používateľ (z dôvodu neželaných zmien dát pre používateľa, ktorý danú formáciu vytvoril),
- ak formáciu začal editovať iný používateľ a neprekročil zadefinovaný čas na uloženie (globálna konštanta *locked_time*, ktorej hodnota udáva čas v sekundách),
- ak má formácia priameho rodiča alebo deti (podformácie) a niekto ich edituje.

V prípade splnenia niektorej z uvedených podmienok sa vyvolá výnimka *FormationUnauthorizedException* s príslušnou hláškou. Hlášky sú prekladateľné a používateľovi sa pre túto výnimku zobrazí stránka s HTTP kódom *403 forbidden*, kde sa nachádza hláška s dôvodom, prečo je formácia zamknutá pre akciu editácie. Globálna konštanta *locked_time* je uložená v konfiguračnom súbore *locking.php*. Zavedením tejto konštanty sme vyriešili problém ak by používateľ začal akciu editácie, ale nikdy ju už neukončil. Ak formácia nie je zamknutá, do databázy sa uložia informácie o čase editovania a používateľovi, ktorý akciu editácie vykonal, ako sme mohli vidieť na obrázku 4.25. Zamykanie dát funguje obdobne aj pre iné akcie formácií, ktoré sa môžu líšiť v podmienkach zamykania (pri recenzovaní nechceme povoliť recenzovať formáciu, ktorá nie je v statuse *in_review*), taktiež je obdobne implementované aj pre množinu entít *profily* a *merania veku*, kde sa pozeráme na dáta o naviazaných formáciách. Záznamy tabuliek *profiles* a *measurements_of_ages* zamykáme vtedy, ak sú zamknuté aj dáta o naviazanej formácii.

4.4 Exportovanie a importovanie dát

Ako z požiadaviek na aplikáciu vieme, aplikácia umožňuje importovanie stratigrafických intervalov vo formáte xslx a exportovanie dát formácií vo formáte pdf. V tejto časti práce si popíšeme implementáciu týchto požiadaviek.

4.4.1 Importovanie stratigrafických intervalov

Pre importovanie stratigrafických intervalov sme sa rozhodli použiť funkcionality knižnice *maatwebsite/excel*, ktorá je súčasťou Craftable. Táto knižnica si pre sprostredkovanie dát z .xslx súboru vyžaduje zadefinovať triedu, ktorú sme pomenovali *IntervalsImport* a pomocou inštancie tejto triedy a metód, ktoré ponúka, získame dáta vo forme kolekcie na mieste, kde ich potrebujeme. Táto trieda sa nachádza v mennom priestore *App\Imports* a jej kostra sa nachádza na obrázku 4.28.

```
class IntervalsImport implements ToCollection, WithHeadingRow {
    use Importable;

    public function collection(Collection $collection){}
}
```

Obr. 4.28: Kostra triedy IntervalsImport

Pomocou metód, ktoré sa nachádzajú v *traite Importable* dostaneme z importovaného .xslx súboru dáta s hlavičkami formou kolekcie. Interface *ToCollection* predpisuje zadefinovanie funkcie *collection(Collection $collection)*, ktorá je v našom prípade prázdna z dôvodu, že logiku importovania máme pre jej komplexnosť rozdelenú do viacerých tried a metód. Funkcionalita importovania sa vykonáva v triede *IntervalsController* pomocou metódy *import(ImportIntervals $request)*, ktorá sa zavolá v prípade HTTP požiadavky zo strany klienta, kde sa v tele požiadavky nachádza súbor pre importovanie. Táto metóda pracuje s metódami implementovanými v triede *IntervalsImporter* zadefinovanej v mennom priestore *App\Services\Imports*, ktorej metódy používame pomocou techniky Injektovania Závislostí (z anglického Dependency Injection). Pre lepšie pochopenie funkcionality importovania popíšeme metódy, ktoré trieda *IntervalsImporter* ponúka. Začneme metódou *validImportFile($collectionToImport)*, ktorá sa stará o validáciu hlavičiek importovaného súboru. Jej kostra sa nachádza na obrázku 4.29.

```
public function validImportFile($collectionToImport): bool {
    $requiredHeaders = ['title', 'from', 'to', 'parent_title'];
    foreach ($requiredHeaders as $item) {
        if (!isset($collectionToImport->first()[$item])) {
            return false;
        }
    }
    return true;
}
```

Obr. 4.29: Kostra metódy validImportFile

V poli $requiredHeaders sú zadefinované povinné hlavičky, ktoré musí importovaný súbor obsahovať. Ak niektorá z hlavičiek chýba, táto metóda vráti pravdivostnú hodnotu *false*. Túto metódu používame v metóde *getCollectionFromImportedFile($file)*, ktorej úlohou je získanie kolekcie dát z importovaného súboru. Kostra metódy *getCollectionFromImportedFile($file)* sa nachádza na obrázku 4.30.

```
public function getCollectionFromImportedFile($file) {
    if ($file->getClientOriginalExtension() !== 'xlsx') {...}
    try {
        ...
        $collectionFromImportedFile = (new IntervalsImport())
                                    ->toCollection($file)
                                    ->first();
        if (!$this->validImportFile(...)) {...}
        return $collectionFromImportedFile;
    } catch (Exception $e) {...}
}
```

Obr. 4.30: Kostra metódy getCollectionFromImportedFile

Táto metóda najprv skontroluje, či má importovaný súbor správnu príponu a následne pomocou inštancie triedy *IntervalsImport*, ktorej kostru sme mohli vidieť na obrázku 4.28, zapíše dáta z importovaného súboru do kolekcie *$collectionFromImportedFile*, v ktorej sa pomocou metódy *validImportFile* skontroluje, či obsahuje všetky povinné hlavičky. Ak máme dáta validné a uložené v premennej *$collectionFromImportedFile* chceme sa pozrieť na rozdiel medzi intervalmi uloženými v databáze a intervalmi v importovanom súbore na základe ich názvov. Ak sa názov intervalu v databáze nenachádza v importovanom súbore, chceme ho z databázy vymazať. O mazanie takýchto intervalov sa stará metóda *deleteNotUsedIntervals(Collection $collectionFromImportedFile, Collection $existingIntervals)*, ktorá ako parametre funkcie dostane intervaly nachádzajúce sa v importovanom súbore a intervaly uložené v databázovom systémové. Kostra tejto metódy sa nachádza na obrázku 4.31.

```
public function deleteNotUsedIntervals(...): void {
    $titlesForIntervalsInImportedFile = ...;
    $intervalsToDelete = $existingIntervals
    ->filter(static function ... {
        return !in_array(
            $existingInterval->title,
            $titlesForIntervalsInImportedFile
            );
    });
    $intervalsToDelete->each({$intervalToDelete->delete();});
}
```

Obr. 4.31: Kostra metódy deleteNotUsedIntervals

V premennej $intervalsToDelete sa nachádzajú vyfiltrované intervaly uložené v databázovom systéme, ktoré sa nenachádzajú v importovanom súbore. Intervaly z tejto premennej sa následne z databázového systému vymažú. Ako posledný krok chceme existujúce stratigrafické intervaly aktualizovať a novopridané uložiť do databázového systému. Túto funkcionalitu ponúka metóda *saveCollection(Collection $intervalsCollection)*, ktorá ako parameter dostáva kolekciu dát na importovanie a vráti počet aktualizovaných a pridaných intervalov. Kostra tejto metódy sa nachádza na obrázku 4.32.

```
public function saveCollection(...) : array {
    ...
    $intervalsCollection->each(function ... {
        $created = $this
                ->intervalsRepository
                ->createOrUpdate($interval);
        ...
    });
    return [
        'numberOfImportedIntervals' => $numberOfImportedIntervals,
        'numberOfUpdatedIntervals' => $numberOfUpdatedIntervals
    ];
}
```

Obr. 4.32: Kostra metódy saveCollection

Táto metóda využíva funkcionalitu triedy *IntervalsRepository*, ktorá sa nachádza v mennom priestore *App\Repositories*. Metóda *createOrUpdate*, ktorá je súčasťou tejto triedy, vytvára alebo aktualizuje zadaný interval. Nakoľko môžu mať intervaly vzťah rodič a dieťa, tak rieši aj problém, ak by sa v importovanom súbore nachádzalo dieťa skôr ako jeho rodič. Spomenutá funkcionalita sa vykonáva v databázovej transakcii, čo nám zaručuje, že sa buď vykoná všetko, alebo sa databáza vráti do pôvodného stavu pred importovaním.

4.4.2 Exportovanie dát formácií

Pre exportovanie formácií do formátu pdf sme sa rozhodli použiť funkcionality knižnice *barryvdh/laravel-dompdf*. Exportovanie je implementované v triede *FormationsController* pomocou metódy *bulkExport(ExportFormation $request)*. Kostra tejto metódy sa nachádza na obrázku 4.33.

```
public function bulkExport(...) : BinaryFileResponse {
    $fileNames = collect();

    if ($request->has('all')) {
        $formations = Formation::visible();
    } else {
        $formations = Formation::whereIn('id', ...);
    }

    $formations->chunk(... {
        $pdf = PDF::loadView('...', [
            'formations' => $formationsChunk
        ]);

        $fileName = ...;
        $pdf->save(...);
        $fileNames->push(...);
    });
    ...
    app(Madzipper::class)->make(...)->add(...);
    return response()->download(...);
}
```

Obr. 4.33: Kostra metódy bulkExport

Ako môžeme vidieť z kostry metódy *bulkExport()* do premennej *$formations* uložíme dáta formácií, ktorých identifikátory prídu v HTTP požiadavke používateľa. V prípade, ak používateľ zvolí možnosť exportovať všetky formácie do tejto premennej sa uložia všetky schválené formácie, ktoré dokážeme získať pomocou metódy *visible()*, ktorej funkcionalitu sme popísali pri časti aplikácie sprístupnenej pre neprihlásených používateľov (V triede *Formation* sme ju definovali ako metódu *scopeVisible()* , ale konvencia Laravelu vypúšťa prefix scope). Nakoľko môže byť takýchto formácií viacero, exportovaný pdf súbor nevytvárame pre všetky formácie naraz, ale prechádzame ich po menších častiach pomocou metódy *chunk()*, ktorá je zabudovaná v Laravel kolekciách. Táto metóda berie ako parameter veľkosť časti, podľa ktorej rozdelí kolekciu na požadované časti zadanej veľkosti a funkciu. Keďže má formácia viacero atribútov a vzťahov, rozhodli sme sa, že jedna časť bude obsahovať 50 formácií z dôvodu, aby sme nespracovávali naraz veľké množstvo dát v pamäti. Pre takúto časť vytvoríme pomocou metódy *loadView()* pdf súbor. Táto metóda má ako jeden z parametrov názov *blade* súboru, podľa ktorej sa má daný pdf súbor vytvoriť. Blade súbor vzhľadu pdf

súboru sa nachádza v *blade* súbore s názvom *pdf-export.blade.php*. Po vygenerovaní pdf súboru si daný súbor uložíme do adresára *storage/exports/pdf*. Po vygenerovaní a uložení všetkých pdf súborov pre dané formácie, tieto súbory vložíme do jedného *.zip* súboru, ktorý vytvoríme pomocou metód *make()*, ktorá ako parameter berie cestu do ktorej sa má vytvorený zip súbor uložiť. V našom prípade to je adresár *storage/exports/zip* a pomocou metódy *add()*, ktorá berie ako parameter pole s cestami k súborom, ktoré chceme do zip súboru pridať. Tieto funkcie ponúka inštancia triedy *Madzipper*, ktorá je súčasťou knižnice *madnest/madzipper*. Následne sa takýto zip súbor používateľovi odošle v HTTP odpovedi a prehliadač ho stiahne. V tejto chvíli uložené súbory v adresári *storage/exports* nepotrebujeme. Preto sme sa rozhodli vytvoriť vlastný príkaz pre mazanie súborov nachádzajúci sa v triede *RemoveExportFiles*, ktorej kostra sa nachádza na obrázku 4.34.

```
class RemoveExportFiles extends Command
{
    protected $signature = 'remove:export-files';
    protected $description = 'Remove exports files from storage';
    public function handle()
    {
        ...
        $filesInFolder = array_merge(...);
        $filesToDelete = collect();

        foreach ($filesInFolder as $path) {...}

        $filesToDelete->each(static ... {
            unlink($fileName);
        });
    }

    private function fileIsOlderThanHalfOfDay(...) : bool {...};
    ...
}
```

Obr. 4.34: Kostra triedy RemoveExportFiles

Každý vlastne zadefinovaný príkaz v Laravel aplikácii musí dediť od triedy *Command*, ktorá je súčasťou Laravelu a implementuje v sebe napríklad metódu pre spúšťanie príkazu. V premennej *$signature* sa nachádza názov príkazu, podľa ktorého ho môžeme neskôr zavolať a v premennej *$description* popis jeho funkcionality. Pri volaní príkazu sa vykonáva metóda *handle()*, v ktorej vyberieme zo spomenutých adresárov,

ktoré slúžia pre export všetky súbory. Následne prejdeme ich názvy, ktoré majú prefix dátumu a času ich vytvorenie a na základe metódy *fileIsOlderThanHalfOfDay* sa rozhodneme, či chceme daný súbor vymazať. Chceme vymazať všetky súbory, ktoré sú staršie ako polovica dňa od času, kedy sa tento príkaz zavolal. Dôvodom takejto podmienky je, že nechceme vymazať súbory, ktoré sa vytvorili v krátkej dobe pred zavolaním tohto príkazu, nakoľko môže ísť o súbory, ktoré si v danej chvíli dal používateľ exportovať a ešte ich nestiahol. Tento príkaz môžeme volať manuálne z konzoly pomocou zadania príkazu *php artisan remove:export-files*, alebo automaticky, čo si vyžaduje naša situácia. Príkazy môžeme automaticky volať v triede *Kernel*, ktorá sa nachádza v mennom priestore *App\Console* v metóde *schedule(Schedule $schedule)* pomocou zápisu *$schedule->command('remove:export-files')->daily()*. Takýto zápis nám po nakonfigurovaní služby cron podľa Laravel dokumentácie zaručuje, že sa daný príkaz vykoná každý deň o polnoci.

4.5 Ostatné vybrané funkcionality

V tejto časti práce si popíšeme funkcionality, ktoré sú nad rámec požiadaviek zadefinovaných v kapitole 1, ale nemali by chýbať v žiadnom komplexnejšom softvérovom projekte.

Generátor dát

Aplikácia ponúka generátor náhodných dát pre množiny entít, ktoré dátový model obsahuje. Účelom tohto generátora je sprostredkovať dáta pre používateľov, ktorí sa rozhodnú inicializovať si aplikáciu na lokálnom prostredí (príkladom môže byť programátor, ktorý by chcel pridať novú funkcionalitu), aby nevideli aplikáciu bez dát, respektíve aby nemuseli importovať dáta z produkčnej verzie aplikácie. Ďalším príkladom, kde sa používa generátor dát sú jednotkové testy, ktoré popíšeme v ďalšom bode tejto časti práce. V súboroch *ModelFactory.php* a *UserFactory.php* sa nachádzajú definície náhodných hodnôt atribútov pre množiny entít dátového modelu. Základnú štruktúru kódu v spomenutých súboroch nám vygeneroval nástroj Craftable podľa atribútov v databázových tabuľkách. Túto funkcionalitu sme rozšírili o podporu databázových vzťahov a v prípade niektorých atribútov o validnejšie hodnoty. Ako príklad ukážeme pôvodnú kostru súboru *ModelFactory.php* na množine entít *Formácie*, ktorá sa nachádza na obrázku 4.35 a jej účelom je vygenerovanie atribútov pre jeden záznam v tabuľke *formations*.

```
...
$factory->define(App\Models\Formation::class, ... {
    return [
        'slovak_title' => $faker->sentence,
        'genesis' => $faker->text(),
        'end_in_interval_id' => $faker->randomNumber(5),
        'slug' => $faker->sentence,
        'locked' => $faker->sentence,
        ...
    ];
});
...
```

Obr. 4.35: Kostra pôvodného generátora dát pre množinu entít Formácie

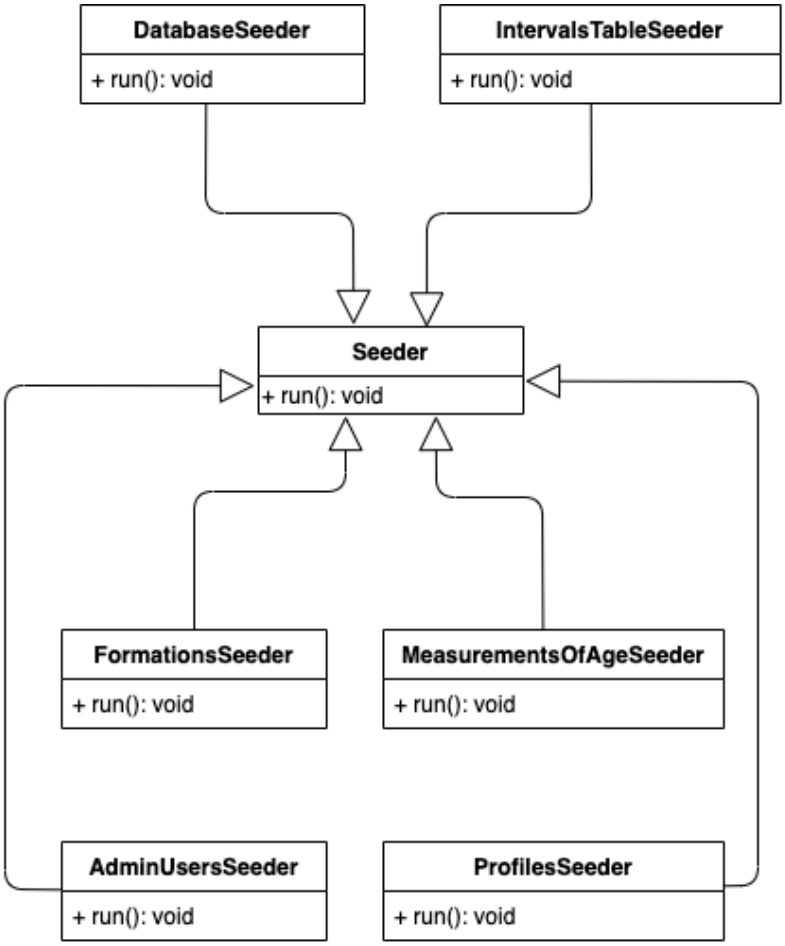
Ako môžeme z ukážky vidieť, všetky dáta sú náhodné. Tento prístup nám nevyhovuje napríklad z dôvodu, že pri atribúte *slug*, ktorý slúži pre lepšie zapamätanie url adresy detailu formácie pre neprihlásených používateľov, chceme hodnotu podľa slovenského názvu. Ďalším príkladom je atribút *locked*, kde chceme mať jednu z hodnôt *new*, *approved*, *rejected* alebo *in_review*. Vykonané zmeny ukážeme na upravenej kostre súboru *ModelFactory.php* na množine entít *Formácie*, ktorá sa nachádza na obrázku 4.36.

```
...
$factory->define(App\Models\Formation::class, ... {
    $slovakTitle = $faker->sentence;
    return [
        'genesis' => $faker->text(),
        'end_in_interval_id' => $faker->randomNumber(5),
        'slug' => Str::slug($slovakTitle),
        'locked' => Arr::random(
            Formation::getAllStatuses()
            ),
        ...
    ];
});
...
```

Obr. 4.36: Kostra generátora dát pre množinu entít Formácie

Dôležitou úpravou vo vygenerovanom kóde je napríklad aj atribút *end_in_interval_id*,

ktorému je pridelená hodnota *$faker->randomNumber(5)*, čo môže spôsobiť problém v prípade, ak by neexistoval interval s týmto identifikátorom. Ako ďalší príklad, kedy nepotrebujeme úplne náhodné dáta je situácia, ak je formácia už schválená. V takomto prípade jej chceme priradiť náhodného recenzenta, ktorý danú formáciu schválil. Pre podporu funkcionality databázových vzťahov a validnejších dát sme vytvorili triedy *Seeder*, ktorých triedny diagram môžeme vidieť na obrázku 4.37.



Obr. 4.37: Triedny diagram pre generátor dát

Abstraktná trieda *Seeder* a trieda *DatabaseSeeder*, ktorá ju rozširuje sú sprostredkované Laravelom. Metóda *run()* v triede *DatabaseSeeder* je vstupným bodom pri generovaní dát. V tejto metóde voláme inštancie nami vytvorených tried *Seeder*, ktorých funkcionalitu popíšeme na triede *FormationsSeeder*. Trieda *FormationsSeeder* v metóde *run()* implementuje podporu databázových vzťahov a validnosť dát na základe aplikačnej logiky aplikácie. Kostra tejto metódy sa nachádza na obrázku 4.38.

```
public function run() : void {
    ...
    $reviewerRole = Role::where('name', 'Reviewer')->first();
    $reviewers = AdminUser::whereHas('roles', .... {
        $query->where('role_id', $reviewerRole->id);
    })->get();

    collect(range(1, 100))->each(static function ($i) use (
            ...
            $reviewers,
            ...
        ) {
        $lockedStatus = Arr::random(Formation::getAllStatuses());
        $formation = factory(Formation::class, 1)
        ->create([
            ...
            'locked' => $lockedStatus,
            'approved_by_admin_user_id' => (
                $lockedStatus === ...FORMATION_TYPE_APPROVED
                ?
                $reviewers->random()->id : null
            ),
            ...
        ]);
        ...
        $formation->fossilises()->sync([
            $fossilises->random()->id
            ]);
        ...
    });
}
```

Obr. 4.38: Kostra metódy run

Ako z kostry môžeme vidieť, najprv vyberieme rolu, ktorá patrí recenzentovi a následne si zoberieme všetkých recenzentov, ktorí sa nachádzajú v databáze. Následne chceme vygenerovať 100 formácií, kde sa na základe vygenerovaného statusu formácie rozhodneme, či jej pridelíme recenzenta. Ak je formácia schválená, tak jej pridelíme náhodného recenzenta z premennej *$reviewers*. Ako môžeme vidieť v tejto triede voláme inštanciu triedy *Factory*, kde ako parameter zadávame názov Modelu, pre ktorý chceme generovať dáta a počet záznamov. Takéto volanie sa odkazuje na definíciu *fac-*

tory pre daný model, ktorej ukážku sme mohli vidieť na obrázku 4.36. Na takejto inštancii zavoláme metódu *create()*, kde ako parameter zadávame pole atribútov. V tomto poli nemusíme uvádzať všetky atribúty modelu, nakoľko sa nachádzajú už v definícii danej *factory*. Do tohto poľa dávame iba atribúty, ktoré chceme mať odlišné. Príkladom je aj atribút *end_in_interval_id* ktorému nastavujeme náhodný interval. Následne pre vygenerovanú formáciu chceme vygenerovať aj $m : n$ vzťahy, ktoré ako vieme, nie sú súčasťou tabuľky *formations*. Takéto vzťahy formácií priradíme pomocou metódy *sync()*, ktorá ako parameter, berie pole identifikátorov záznamov, ktoré chceme naviazať. Takýto princíp používame aj pre iné vzťahy a atribúty formácie a obdobne sú štruktúrované aj triedy *AdminUsersSeeder*, *MeasurementsOfAgeSeeder* a *ProfilesSeeder*. Trieda *IntervalsTableSeeder* je špeciálnym prípadom, nakoľko v tabuľke *intervals* chceme uchovávať intervaly určené podľa oficiálnej tabuľky pre stratigrafické intervaly. Túto tabuľku sme mali k dispozícií vo formáte xlsx, preto sme zvolili postup, že sme ju do aplikácie naimportovali pomocou funkcionality pre importovanie stratigrafických intervalov a následne sme tieto dáta získali z tabuľky *intervals* pomocou knižnice *orangehill/iseed*, ktorá nám vytvorila triedu *IntervalsTableSeeder*, v ktorej sa nachádzajú vkladania pomocou funkcionalít *Query buildera*.

```
class DatabaseSeeder extends Seeder
{
    public function run()
    {
        Model::unguard();
        factory(\App\Models\Publication::class, 10)->create();
        $this->call(FormationsSeeder::class);
        ...
        Model::reguard();
    }
    ...
}
```

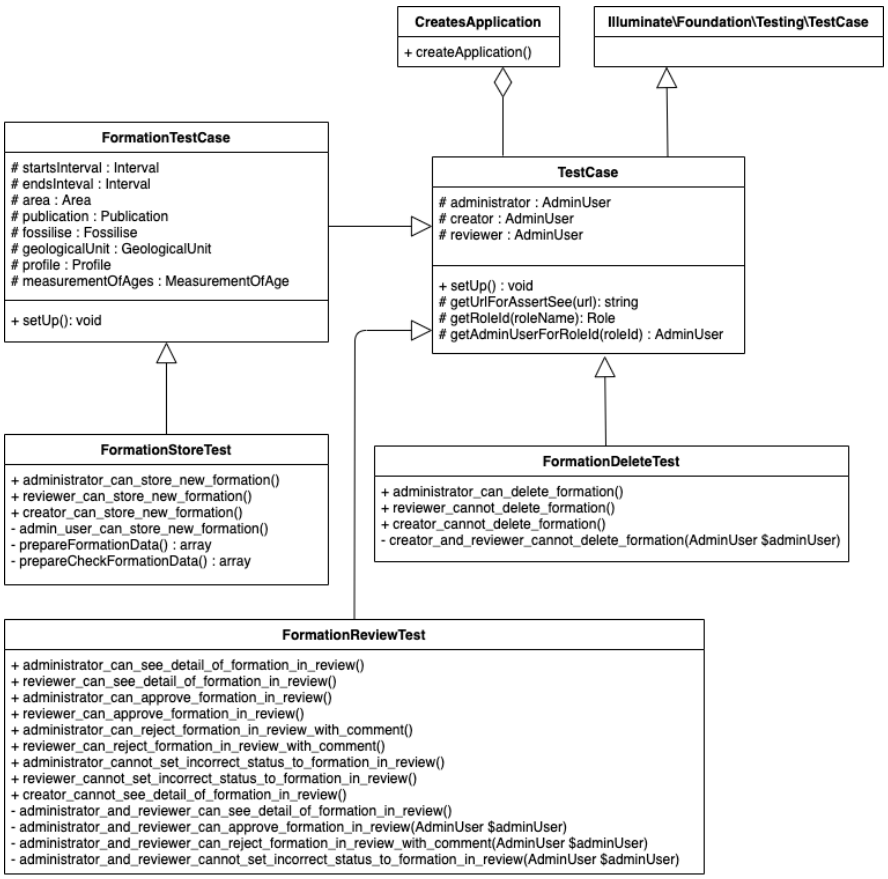
Obr. 4.39: Kostra triedy DatabaseSeeder

Metódy tried *AdminUsersSeeder*, *FormationsSeeder*, *IntervalsTableSeeder*, *MeasurementsOfAgeSeeder* a *ProfilesSeeder* voláme v triede *DatabaseSeeder*. V tejto triede voláme aj metódy triedy *Factory* s parametrami, kde sa nachádza názov modelu a počet záznamov na vygenerovanie pre množiny entít, ktoré si nevyžadovali komplexnejšie úpravy, ktoré by sme vytiahli do samostatných tried. Kostra triedy *DatabaseSeeder* sa nachádza na obrázku 4.39. Statická metóda *unguard()* nám dočasne vypína ochranu Modelu, ktorá zabraňuje masívnemu priraďovaniu dát. Po ukončení generovania dát

túto ochranu znova zapneme pomocou statickej metódy *reguard()*.

Jednotkové testy

Aplikáciu sme otestovali pomocou jednotkových testov, ktoré slúžia na overenie funkčnosti funkcionalít dôležitých časti aplikácie pre neprihlásených a prihlásených používateľov, podľa požiadaviek zadefinovaných v kapitole 1. Pomocou jednotkových testov vieme pri pridávaní nových funkcionalít, alebo pred nasadením aplikácie na produkčnú verziu overiť, či je požadovaná funkcionalita aj naozaj funkčná. Pri testovaní funkcionality pre prihlásených používateľov overujeme danú funkcionalitu z pohľadu rôznych rolí, nakoľko ako vieme, nie každá rola ma sprístupnené rovnaké právomoci pre funkcionality. Jednotkové testy realizujeme na testovacej databáze, ktorá je súčasťou nástroja Harbor. Dôvodom použitia inej databázy, ako tej na ktorej beží aplikácia je tá, že počas testovania chceme vždy vytvoriť databázu a naplniť ju testovacími dátami pomocou generátora a rovnako pomocou jednotkových testov overujeme aj funkcionality, kde sa menia atribúty záznamov v databáze. Jednotkové testy sú na sebe nezávislé a ich triedny diagram, ktorý zobrazuje implementáciu testov na množine entít *Formácie* pre akcie vytvárania, schvaľovania a vymazávania môžeme vidieť na obrázku 4.40.



Obr. 4.40: Triedny diagram pre jednotkové testy

Ako môžeme z triedneho diagramu vidieť, každý test rozširuje abstraktnú triedu *TestCase*, ktorá rozširuje abstraktnú triedu *Illuminate\Foundation\Testing\TestCase* a využíva *trait CreatesApplication*. *Trait CreatesApplication* pomocou metódy *createApplication()* vytvorí aplikáciu. Trieda *Illuminate\Foundation\Testing\TestCase* je sprostredkovaná Laravelom a jej účelom je napríklad vytvorenie aplikácie pomocou metódy *createApplication()* pred každým spustením testu. V triede *TestCase* sme v metóde *setUp()* zabezpečili, že pred každým testom sa databáza vytvorí a vygenerujú sa v nej testovacie dáta. Ďalej v tejto triede priradíme do premenných *$administrator*, *$reviewer* a *$creator* používateľov s danými rolami, prostredníctvom ktorých testy vykonávame. V prípade triedy *FormationStoreTest* sme pre predchádzanie duplicitám a lepšiu štruktúru a modulárnosť kódu vytvorili abstraktnú triedu *FormationTestCase*, ktorú rozširuje. Túto triedu sme vytvorili za účelom, že pri testovaní akcie vytvárania, alebo aktualizovania formácie si potrebujeme z databázy vytiahnuť dáta o vzťahoch, ktoré sú pri vytváraní povinné alebo ktoré chceme zmeniť. V tomto prípade si tieto dáta z databázy náhodne vytiahneme v triede *FormationTestCase* a v triede *FormationStoreTest* ich pomocou metódy *prepareFormationData()* vyskladáme do požadovanej formy pre danú funkcionalitu. Z triedneho diagramu na obrázku 4.40 môžeme vidieť, že sme sa snažili testami pokryť každú dôležitú funkcionalitu a prípady, ktoré môžu nastať , z pohľadu rôznych rolí používateľov na základe požiadaviek z kapitoly 1. Obdobný princíp sme zvolili aj pri implementovaní testov pre všetky akcie formácií, ako aj pre akcie nad množinami entít pre intervaly, profily, merania veku, administračných používateľov a časti aplikácie pre neprihlásených používateľov. Spolu sme implementovali 147 testov, kde sa spolu nachádza 555 kontrol.

Záver

Cieľom bakalárskej práce bolo vytvoriť viacpoužívateľskú webovú aplikáciu pre správu litostratigrafických dát podľa požiadaviek Katedry geológie a paleontológie Prírodovedeckej fakulty Univerzity Komenského v Bratislave. Podľa týchto požiadaviek sme vybrali vhodné technológie, prostredníctvom ktorých sme webovú aplikáciu pre správu litostratigrafických dát implementovali. Nakoľko ide o webovú aplikáciu, ktorá je prispôsobená aj pre mobilné zariadenia, jej používateľom umožňujeme pracovať s aplikáciou na ľubovoľnom zariadení a ľubovoľnom operačnom systéme. Jedinou podmienkou je internetové pripojenie a webový prehliadač. Aplikácia neponúka možnosť samoregistrácie a jej používatelia majú funkcionality priradené pomocou používateľskej role, čo zaručuje, že k administračnej časti aplikácie nebudú mať prístup nepovolené osoby. Administrátor sa sám rozhodne, aká rola je pre daného používateľa najvhodnejšia. Prihláseným používateľom je umožnené vytvárať, editovať, exportovať a importovať litostratigrafické dáta, ktoré podliehajú procesu schvaľovania. Pri procese schvaľovania dát, sa jej používateľom zobrazuje posledná schválená verzia, ktorá je uzamknutá pre editáciu, čo si vyžadovalo implementovať podporu zamykania a ukladania viacerých verzií dát. Širokej verejnosti je umožnené vyhľadávať iba v dátach, ktoré prešli procesom schvaľovania, čo zaručuje, že sú vyhľadané informácie správne a v korektnom formáte. Pri typových a referenčných profiloch formácie zobrazujeme ich geografickú pozíciu na interaktívnej mape, čím majú používatelia presné informácie o polohe a v prípade záujmu, môžu dané miesto osobne navštíviť. Keďže ide o komplexnejšiu aplikáciu, rozhodli sme sa, že pre overenie funkcionality implementujeme aj jednotkové testy, ktoré overujú funkčnosť najdôležitejších častí aplikácie.

Nakoľko je litostratigrafia špecifickou disciplínou stratigrafie, vidíme možné vylepšenie a rozšírenie aplikácie o podporu správy dát pre všetky stratigrafické disciplíny a ich jednotky s ktorými pracujú. Pri takomto rozšírení by bolo možné pridať aj nové atribúty pre používateľské role, ktoré by tvorcom umožnili pridávať a editovať iba informácie pre priradenú stratigrafickú disciplínu a rovnako recenzentom možnosť schvaľovania dát pre priradenú stratigrafickú disciplínu.

Literatúra

[1] Angular angular github repository. dostupné online: https://github.com/angular/angular. Citované: 2020-04-21.

[2] Angular documentation. dostupné online: https://angular.io/docs. Citované: 2020-04-21.

[3] Craftable documentation. dostupné online: https://getcraftable.com/docs/5.0/overview. Citované: 2020-04-21.

[4] Facebook react github repository. dostupné online: https://github.com/facebook/react. Citované: 2020-04-21.

[5] Geoscience Australia australian stratigraphic units database. dostupné online: https://www.ga.gov.au/data-pubs/datastandards/stratigraphic-units. Citované: 2020-04-14.

[6] Laravel 6 documentation. dostupné online: https://laravel.com/docs/6.x. Citované: 2020-04-15.

[7] Laravel nova. dostupné online: https://nova.laravel.com/. Citované: 2020-04-21.

[8] Packagist the php package repository. dostupné online: https://packagist.org/packages/laravel/framework. Citované: 2020-04-30.

[9] Postgresql documentation. dostupné online: https://www.postgresql.org/docs/11/index.html. Citované: 2020-04-21.

[10] React documentation. dostupné online: https://reactjs.org/. Citované: 2020-04-21.

[11] State of javascript. dostupné online: https://2019.stateofjs.com/front-end-frameworks/. Citované: 2020-04-15.

[12] Univerzita Komenského v Bratislave významné paleontologické lokality slovenska. dostupné online: http://www.paleolocalities.com/index.php/site/mapa. Citované: 2020-04-24.

[13] Vue.js 2 documentation. dostupné online: `https://vuejs.org/v2/guide/`. Citované: 2020-04-15.

[14] Vue.js vue github. dostupné online: `https://github.com/vuejs/vue`. Citované: 2020-04-21.

[15] What is a container? a standardized unit of software. dostupné online: `https://www.docker.com/resources/what-container`. Citované: 2020-04-15.

[16] J. Michalík a spol. *Stratigrafická príručka*. Veda, vydavateľstvo SAV, 2007.

[17] J. Widom H. Garcia-Molina, J. D. Ullman. *Database Systems: The Complete Book*. Pearson, 2002.

[18] The Open Web Application Security Project. Cross site request forgery (csrf). dostupné online: `https://owasp.org/www-community/attacks/csrf`. Citované: 2020-05-11.