

**SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY**

Evidenčné číslo: FEI-5384-46121

**ROZVRHOVÝ SYSTÉM PRE FEI - BACKEND
DIPLOMOVÁ PRÁCA**

2019

Bc. Martin Makšin

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: FEI-5384-46121

ROZVRHOVÝ SYSTÉM PRE FEI - BACKEND
DIPLOMOVÁ PRÁCA

Študijný program: Aplikovaná informatika
Číslo študijného odboru: 2511
Názov študijného odboru: 9.2.9 Aplikovaná informatika
Školiace pracovisko: Ústav informatiky a matematiky
Vedúci záverečnej práce: Mgr. Ing. Matúš Jókay, PhD.

Bratislava 2019

Bc. Martin Makšin



ZADANIE DIPLOMOVEJ PRÁCE

Študent: **Bc. Martin Makšin**
ID študenta: 46121
Študijný program: aplikovaná informatika
Študijný odbor: 9.2.9. aplikovaná informatika
Vedúci práce: Mgr. Ing. Matúš Jókay, PhD.
Miesto vypracovania: Ústav informatiky a matematiky

Názov práce: **Rozvrhový systém pre FEI – backend**

Jazyk, v ktorom sa práca vypracuje: slovenský jazyk

Špecifikácia zadania:

Cieľom práce je pokračovať v implementácii rozvrhového systému pre FEI STU BA. V tejto fáze je potrebné vytvoriť klientskú aplikáciu, ktorá umožní interakciu rozvrhára so systémom.

Úlohy:

1. Naštudujte aktuálny stav návrhu a implementácie rozvrhového systému.
2. Vytýčte ciele ďalšieho rozvoja.
3. Implementujte vami navrhnuté ciele.
4. Zhodnoťte svoj prínos pre aplikáciu rozvrhového systému.


Zoznam odbornej literatúry:

1. Bednár, D. – Jókay, M. *Tvorba užívateľského rozhrania k rozvrhovému systému pre FEI*. Diploma thesis. 2018. 47 s.
2. Račák, M. – Jókay, M. *Rozvrhový systém pre FEI*. Diploma thesis. 2017. 39 s.


Riešenie zadania práce od: 17. 09. 2018

Dátum odovzdania práce: 13. 05. 2019

Bc. Martin Makšin
študent



prof. RNDr. Otokar Grošek, PhD.
vedúci pracoviska



prof. Dr. Ing. Miloš Oravec
garant študijného programu

SÚHRN

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Študijný program:	Aplikovaná informatika
Autor:	Bc. Martin Makšín
Diplomová práca:	Rozvrhový systém pre FEI - backend
Vedúci záverečnej práce:	Mgr. Ing. Matúš Jókay, PhD.
Miesto a rok predloženia práce:	Bratislava 2019

Diplomová práca sa venuje problematike tvorby rozvrhov. Nakoľko údaje vstupujúce do tvorby rozvrhu sa v priebehu jeho vytvárania môžu meniť, v práci sa skúmajú možnosti oddelenia údajov špecifických pre jednotlivé rozvrhy. Tým je zabezpečené, že zmeny vstupných údajov neovplyvnia konzistentnosť existujúcich rozvrhov. Práca sa špeciálne zameriava na rozšírenie serverovej časti aplikácie, ktorá má na starosti tzv. backend. Backend je riešený pomocou webovej služby založenej na REST API. Rozdelenie aplikácie na klient-sku a serverovú časť umožňuje nezávislý (súbežný) vývoj oboch častí. Diplomová práca rozširuje funkcionality serverovej časti aplikácie o autentifikáciu a autorizáciu používateľov (rozvrhárov) cez Akademický informačný systém, o import údajov z tohto systému a definíciu rolí (spolu s oprávneniami), ktoré sa v systéme využívajú.

Kľúčové slová: rozvrh, django, oddelenie dát

ABSTRACT

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA
FACULTY OF ELECTRICAL ENGINEERING AND INFORMATION TECHNOLOGY

Study Programme:	Applied Informatics
Author:	Bc. Martin Makšín
Master's thesis:	Timetable system for FEI - backend
Supervisor:	Mgr. Ing. Matúš Jókay, PhD.
Place and year of submission:	Bratislava 2019

The thesis deals with the topic of creating school timetables. As data entering the timetable may change during its creation, the thesis explores the possibilities of separating timetable specific data. This ensures that changes of the input data do not affect the consistency of existing timetables. The thesis is especially focused on extending the server part of the application, which deals with so-called backend. The backend part of the application consists of a web service based on the REST API. The separation of the application into the client and server part allows an independent (simultaneous) development of both parts. The thesis extends a functionality of the server part of the application by authenticating and authorizing users (timetable creators) using the Academic Information System, importing data from this system and defining the roles (along with their permissions) used in the system.

Keywords: timetable, django, data separation

Podakovanie

Na úvod sa chcem poďakovať môjmu vedúcemu práce Mgr. Ing. Matúšovi Jókayovi, PhD. za jeho odbornú pomoc, zhovievavosť, cenné rady a pripomienky, ktoré mi boli poskytnuté počas strenutí a pomohli mi pri vypracovaní tejto diplomovej práce. Taktiež sa chcem poďakovať Mgr. Dávidovi Panczovi, PhD. za poskytnuté informácie o tvorbe rozvrhov na našej fakulte, Ing. Júliusovi Annusovi za poznatky o vývoji aplikácie eRozvrh a Bc. Adamovi Kurtákovi za bezproblémovú spoluprácu. Na záver sa chcem poďakovať rodine a priateľke za ich podporu.

Obsah

Úvod	1
1 Analýza problematiky a cieľ práce	2
1.1 Proces tvorby rozvrhu	2
1.2 Analýza softvérových riešení	3
1.2.1 ROGER	3
1.2.2 Prime Timetables	5
1.2.3 eRozvrh FCHPT	6
1.3 Tvorba rozvrhu na FEI	8
1.4 Cieľ práce	8
1.5 Špecifikácia požiadaviek	9
2 Návrh riešenia	10
2.1 Prehľad architektúry systému	10
2.2 Úpravy aplikácie	11
2.3 Webová služba	12
2.3.1 REST	12
2.3.2 Návrh REST API	14
2.4 Autentifikácia	15
2.4.1 JWT autentifikácia	15
2.4.2 LDAP	16
2.5 Autorizácia	18
2.6 Viacvlastníková architektúra	19
2.7 Stavy	20
3 Použité technológie	22
3.1 Python	22
3.2 Django	22
3.3 PostgreSQL	23
3.4 Git	23
3.5 Vývojové prostredie	24
4 Implementácia a testovanie	25
4.1 Zmeny v aplikácii	25
4.2 Autentifikácia	26

4.3	Práca s API	27
4.4	Import údajov	28
4.5	Viacvlastníková architektúra	29
4.6	Roly a právomoci	30
4.7	Stavy	30
4.8	Dokumentácia API	31
4.9	Testovanie	32
Záver		34
Zoznam použitej literatúry		35
Prílohy		I
A Štruktúra elektronického nosiča		II
B Technická dokumentácia		III

Zoznam obrázkov a tabuliek

Obrázok 1	Úvodná obrazovka pri vytváraní nového rozvrhu [5]	4
Obrázok 2	Zobrazenie kolízií v programe Prime Timetables [6]	6
Obrázok 3	Pôvodná architektúra – diagram balíčkov (zoskupení tried) [1] .	10
Obrázok 4	JSON Web Token s použitím HMAC-SHA256 [14]	17
Obrázok 5	Ukážka stromovej štruktúry [17]	17
Obrázok 6	Modely prístupu k dátam vo viacvlastníkovej architektúre [18] .	19
Obrázok 7	Stavový diagram pre schémy	20
Obrázok 8	Stavové diagramy pre rozvrhy.	21
Obrázok 9	Spracovanie volania API v Django Rest Framework (DRF) [27] .	27
Obrázok 10	Ukážka zadefinovania zdieľaných a privátnych aplikácií	29
Obrázok 11	Ukážka zadefinovania prechodov pre rozvrhy	31
Obrázok 12	Vyskakovacie okno pre prihlásenie	32
Obrázok 13	Ukážka volania GET /timetables/	33

Zoznam skratiek

ACID	Atomicity, Consistency, Isolation, Durability
AIS	Akademický informačný systém
API	Application programming interface
CRUD	Create, Read, Update, Delete
CSV	Comma-separated values
DRF	Django Rest Framework
DRY	Don't repeat yourself
ECDSA	Elliptic Curve Digital Signature Algorithm
FEI	Fakulta elektrotechniky a informatiky
HATEOAS	Hypermedia as the Engine of Application State
HMAC	Hash-based message authentication code
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IANA	Internet Assigned Numbers Authority
ID	Identifikátor
JSON	JavaScript Object Notation
JWT	JSON Web Token
LDAP	Lightweight Directory Access Protocol
LTS	Long term support
MVCC	Multiversion concurrency control
ORM	Object-relational mapping
PHP	Hypertext Preprocessor
REST	Representational state transfer
RSA	Rivest–Shamir–Adleman algorithm
SaaS	Software as a service
SOAP	Simple Object Access Protocol
SPA	Single-page application
SQL	Structured Query Language
STU	Slovenská technická univerzita
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator

W3C	World Wide Web Consortium
WSDL	Web Services Description Language
XML	eXtensible Markup Language

Zoznam výpisov

1	Ukážka použitia knižnice Python Decouple	26
2	Ukážka premapovania LDAP atribútov do databázy	26
3	Ukážka nastavenia JWT autentifikácie	31

Úvod

Každá škola potrebuje rozvrh pre svoj správny chod. Forma a komplexnosť tvorby rozvrhu sa líši v závislosti od typu školy, premenlivosti vstupných údajov a dostupných nástrojov, ktoré sa pri vypracovaní rozvrhu používajú. V prípade univerzít je nutné vytvoriť pre každé výučbové obdobie semestrálny a skúškový rozvrh. Ich prípravu komplikuje množstvo špecifických faktorov ako napr. skupiny vs individuálny zápis, opakujúci študenti, priestorové možnosti, či požiadavky pedagógov.

V dnešnej dobe sa na zjednodušenie tvorby rozvrhov používajú rôzne softvérové riešenia, ktoré v niektorých prípadoch ponúkajú aj generovanie rozvrhu. Ich problémom občas býva to, že neumožňujú zadať špecifické požiadavky na rozvrh. Rozhodli sme sa ísť vlastnou cestou a v tejto práci sa venujeme ďalšiemu rozvoju aplikácie Elisa.

Na úvod sme sa oboznámili s predchádzajúcimi prácami. Serverovú časť sme naštuodovali z prác Ing. Emílie Knaperekovej [1] a Ing. Martina Račáka [2], preverili sme aj technologický posun Ing. Dávida Bednára v minuloročnej práci [3], ktorá bola zameraná na klienta.

V prvej kapitole sa venujeme analýze tvorby rozvrhov a existujúcich softvérových riešení, definujeme ciele a špecifikujeme požiadavky. Zhodnoteniu stavu predošlého riešenia, návrhu úprav a dôležitým pojmom sa venujeme v kapitole 2. V tretej kapitole stručne predstavujeme použité technológie. Napokon kapitola 4 predstavuje súhrn implementovaných zmien, vrátane technického popisu.

V práci používame niektoré cudzojazyčné výrazy, pre ktoré sme nenašli vhodný slovenský ekvivalent, alebo ktoré je kvôli porozumeniu lepšie nechať v pôvodnom jazyku. Pre takéto výrazy používame písmo štýlu *italic*. Na zvýraznenie dôležitých pojmov používame písmo štýlu **bold**. Príkazy od bežného textu odlišujeme písmom **technického štýlu**.

1 Analýza problematiky a cieľ práce

V tejto kapitole sa najprv venujeme analýze procesu tvorby rozvrhu v univerzitnom prostredí. Ide o náročný proces, pri ktorom je nutné spracovať obrovské množstvo vstupných dát. Spočiatku museli rozvrhári zostavovať rozvrhy manuálne, s príchodom moderných technológií sa ich tvorba zjednodušila.

V súčasnosti existuje viacero softvérových riešení, z ktorých si niekoľko v tejto kapitole predstavuje. Tie ponúkajú príjemné grafické rozhranie na úpravu vstupných dát, ich vizualizáciu a možnosti na tvorbu alebo generovanie rozvrhu. Generovanie rozvrhov je NP-úplný optimalizačný problém, pri riešení ktorého sa využívajú napr. evolučné algoritmy, grafové algoritmy alebo rôzne heuristické prístupy. Avšak aj v prípade generovaných rozvrhov nastávajú kolízie, ktoré je nutné riešiť ručne.

1.1 Proces tvorby rozvrhu

Rozvrh je základom organizácie akademického roku a proces jeho tvorby je komplexná záležitosť. Potrebujeme vziať do úvahy všetky faktory, ktoré vstupujú do tohto procesu a nájsť riešenie, ktoré ich bude najlepšie spĺňať. Spracovanie týchto faktorov zostáva na pleciach tvorcov rozvrhu. Vieme však určiť pre vstupné dáta pravidlá, ktoré musia byť splnené, aby bolo možné požadovaný rozvrh vytvoriť.

Tvorba rozvrhu začína spracovaním vstupných dát, kde sa predpokladá, že rozvrhár pozná detailnejšiu štruktúru týchto dát. Na začiatku má rozvrhár k dispozícii tieto vstupné údaje:

- zoznam pedagogických pracovníkov,
- zoznam predmetov,
- zoznam študentov a ich rozdelenie,
- zoznam pracovísk zabezpečujúcich výučbový proces,
- zoznam miestností a ich vybavenie,
- zoznam študijných programov a zameraní.

Okrem rozvrhárov vstupujú do tohto procesu aj pedagogickí pracovníci (garanti, prednášajúci). Pedagogickí pracovníci zadávajú pre predmety, ktoré vyučujú, svoje požiadavky na rozvrh. Medzi tieto požiadavky patria časové možnosti týchto pracovníkov, požiadavky

na vybavenie miestnosti, či určenie miestnosti, kde by mala prebiehať výučba. Tieto požiadavky musí rozvrhár zozbierať od jednotlivých pedagógov a následne sa snaží nájsť optimálne riešenie.

Rozvrhár alebo softvér na generovanie rozvrhu rozlišuje pri jeho tvorbe tzv. tvrdé ohraničenia a mäkké obmedzenia. Tvrdé ohraničenia musia byť splnené, aby sa mohol rozvrh označiť za platný (napr. v miestnosti nemôže naraz prebiehať viac predmetov alebo miestnosť musí mať dostatočnú kapacitu pre daný predmet). [4] Mäkké obmedzenia nemusia byť splnené, avšak ich zapracovaním sa zvyšuje hodnota výsledného rozvrhu. Sem môžeme zaradiť preferencie pedagógov (na miestnosť a/alebo čas) alebo požiadavku, aby prednáška bola pred cvičeniami. [4]

Proces prebieha v iteráciách, kedy rozvrhár postupne zverejňuje rozvrh najprv pedagogickým pracovníkom a následne aj študentom.

1.2 Analýza softvérových riešení

Na Internete existuje niekoľko softvérových riešení, ktoré umožňujú vytvárať a generovať rozvrhy pre univerzity. Pri nich je však potrebné preskúmať, ktoré operačné systémy podporujú, pod akou licenciou sa môžu používať (či sa jedná o open-source riešenie alebo o platený produkt) a funkcionality, ktorú poskytujú.

Pri funkcionalite je dôležité vedieť, do akej miery nám softvér pomôže pri tvorbe rozvrhu, aké typy rozvrhov dokážeme pomocou neho generovať alebo či umožňuje import staršieho rozvrhu a export vytvoreného rozvrhu. Popri správnej funkcionalite netreba zabúdať aj na kvalitný dizajn aplikácie a to najmä z pohľadu používateľského komfortu.

1.2.1 ROGER

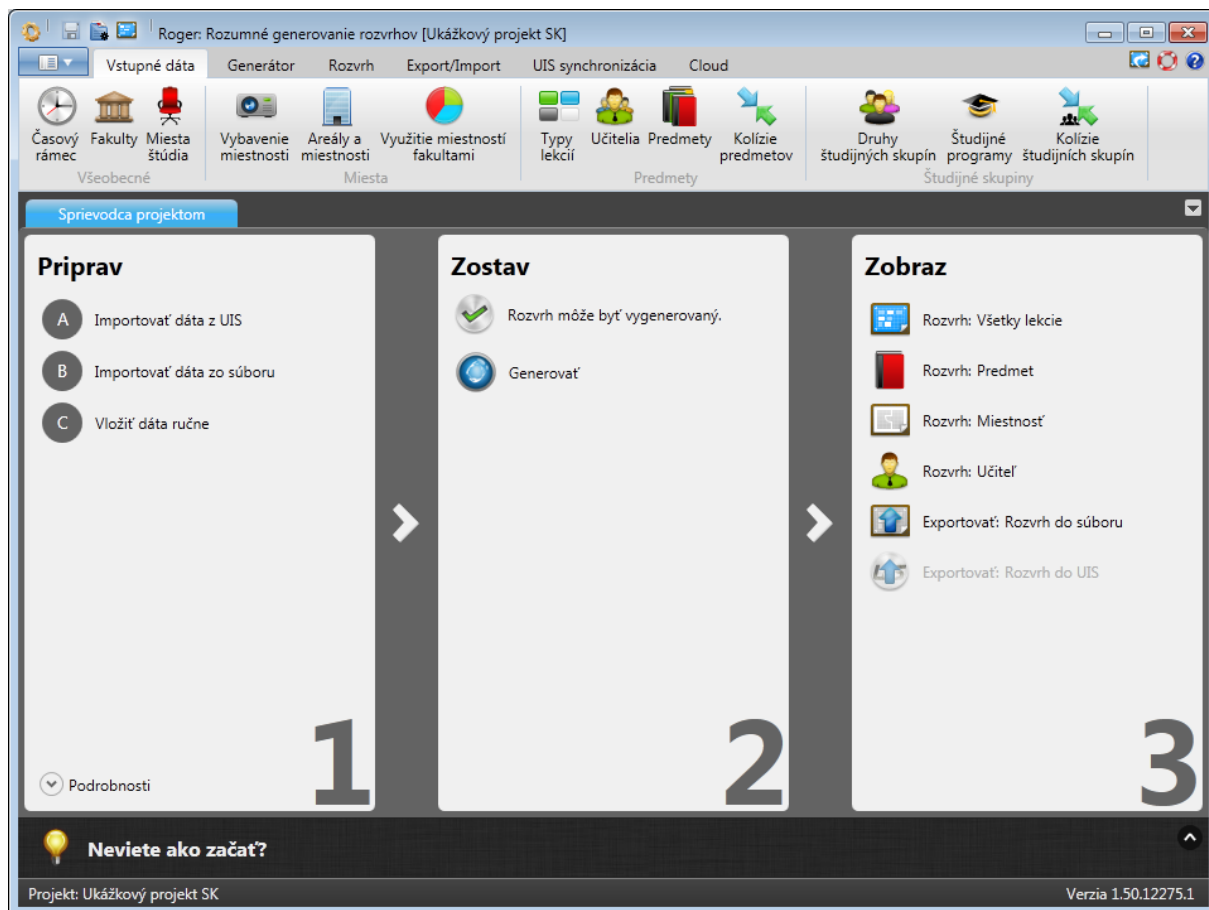
Informácie v tejto podkapitole sme čerpali z oficiálnej stránky ROGER. [5]

ROGER je program, ktorý bol vytvorený firmou IS4U. Ide o desktopovú aplikáciu, ktorá má podporu aj pre *cloud*. Tvorba rozvrhu prebieha v troch krokoch:

1. Príprava dát - dáta je možné importovať zo súboru, z AIS-u alebo vytvoriť ručne.
2. Generovanie rozvrhu - generovanie rozvrhu prebieha pomocou genetického algoritmu, ktorého autorom je Pavel Petrovič.
3. Zobrazenie rozvrhu - je možné prezeráť rôzne pohľady na rozvrh (napr. podľa miestností, predmetu), export do súboru (XML alebo XLS formát) alebo do AIS.

ROGER zo vstupných dát vygeneruje rozvrh, ktorý môže obsahovať kolízie (tie sú zobrazené červenou farbou). Pri vytváraní nového rozvrhu existuje pre rozvrhovú akciu

možnosť “zamraziť” niektoré z jej údajov, t.j. nie je umožnené ich meniť a pri ďalších výpočtoch ich algoritmus ignoruje. Pri definícii lekcií pre zvolený predmet je možné v rámci nastavení zadefinovať, či je vyučovanie spojené s iným predmetom, poprípade zadefinovať obmedzenia pre lekciiu.



Obr. 1: Úvodná obrazovka pri vytváraní nového rozvrhu [5]

Aplikácia ponúka príjemné používateľské rozhranie (Obr. 1), ktoré je štruktúrované pomocou záložiek a je v slovenskom jazyku. Vďaka podpore *cloudu* je možné zdieľať rozvrh s inými používateľmi, ktorí môžu spolupracovať na jeho vytváraní. Program je možné používať v *trial* verzii s neobmedzenou funkčnosťou, je však obmedzená na 180 lekcií, pričom pod lekciiou sa chápe rozvrhová akcia s týždennou periodicitou. Následne cena vzrastá v závislosti od počtu používateľov (zariadení, na ktorých bude aplikácia nainštalovaná) a počtu lekcií. Keďže na našej fakulte sa uskutočňuje veľké množstvo rozvrhových akcií, tak by to bolo drahé riešenie.

Výhody

- import dát z AIS-u a export dát pre AIS

- funkcionalita
- podpora cloudu
- príjemné používateľské rozhranie

Nevýhody

- cena podľa počtu rozvrhových akcií

1.2.2 Prime Timetables

Informácie v tejto podkapitole sme čerpali z oficiálnej stránky Prime Timetables. [6]

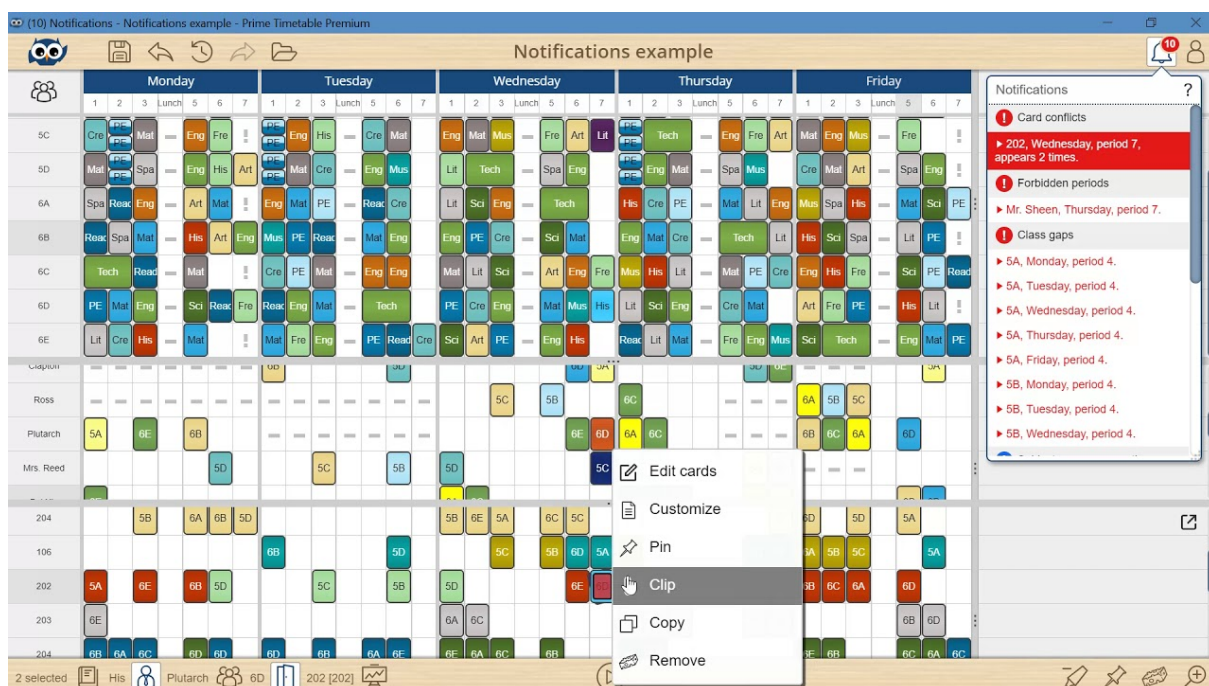
Prime Timetables je ďalším z nástrojov na tvorbu rozvrhov. Jedná sa o webovú aplikáciu, preto je ju možné používať nielen na počítači, ale aj na tablete či smartfóne. Aplikácia ponúka príjemné responzívne používateľské rozhranie, ktoré ponúka rôzne pohľady na rozvrh, možnosti filtrácie vstupných dát, podporu *drag&drop*. Vstupné dáta je možné načítať zo súboru alebo vložením do príslušných modálnych okien. Rozvrh je synchronizovaný, je možné ho zdieľať s inými používateľmi, exportovať do viacerých formátov a je dostupný aj v *offline* režime.

Pri úprave rozvrhu je možné jednotlivé rozvrhové akcie zoskupiť, pričom pri ich presúvaní sa automaticky prepočítavajú kolízie a zároveň sa ich pozícia mení vo všetkých zobrazeniach rozvrhu na obrazovke. Ukážka používateľského rozhrania je zobrazená na obrázku č.2. Rozvrh je možné vygenerovať aj automaticky a taktiež je možné “ukotviť” rozvrhové akcie, aby už nevstupovali do generovania rozvrhu. Rozvrhár určite ocení históriu zmien, ktorá mu umožňuje vrátiť späť nesprávne rozhodnutia.

Prime Timetables ponúka prehľadnú dokumentáciu a možnosť vyskúšať ho zadarmo počas 30-dňovej skúšobnej doby. Po jej expirácii je potrebné zaplatiť ročné predplatné, ktorého cena sa odvíja od počtu učiteľov. Za nevýhodu sa dá považovať tiež to, že údaje sú ukladané na serveroch, ktoré nepatria fakulte.

Výhody

- riešenie vo forme webovej aplikácie
- funkcionalita
- responzívny dizajn, príjemné používateľské rozhranie
- multiplatformová podpora
- offline režim



Obr. 2: Zobrazenie kolízií v programe Prime Timetables [6]

Nevýhody

- cena
- dáta uložené na cudzích serveroch

1.2.3 eRozvrh FCHPT

V rámci STU vznikla na Fakulte chemickej a potravinárskej technológie aplikácia eRozvrh. Jej tvorcom je Ing. Július Annus, ktorý ju vytvoril v programovacom jazyku PHP. Aplikácia ponúka jednoduché grafické rozhranie s rozsiahlou funkcionalitou pre tvorbu semestrálnych a skúškových rozvrhov. Rozvrhy sú manuálne vytvárané rozvrhárom. Dáta, s ktorými aplikácia pracuje, sú získané z AIS-u, je možný aj ich export.

Na prihlasovanie sa nepoužíva AIS, ale eRozvrh obsahuje vlastnú formu prihlásenia. Ide o vygenerovanú personalizovanú linku, ktorá obsahuje identifikátor študenta, učiteľa alebo rozvrhára a štvormiestny kód (tzv. **idc kód**). Tento kód by mohol byť z hľadiska bezpečnosti považovaný ako nedostatočný prvok ochrany, existuje však aspoň kontrola IP adresy a limit na nesprávny počet pokusov.

eRozvrh dokáže riešiť rozdielny prístup k spracovaniu študentov (cez krúžky alebo samostatne) pri vytváraní rozvrhu, problémy s opakujúcimi študentmi, kolízie predmetov či splnenie požiadaviek pre daný predmet (predmet môže požadovať ukončenie iného

predmetu). Cez univerzálny formulár sa zbierajú požiadavky pedagógov, ktoré sa potom manuálne prezerajú.

Aplikácia poskytuje tiež grafické rozhranie na vytvorenie osobného rozvrhu. Pri výbere predmetov študenti nemajú k dispozícii mená vyučujúcich. Proces tvorby osobného rozvrhu je viackolový. V prvom kole si študent iba podáva žiadosť o rozvrh a výber trvá celý deň. Následne rozvrhy študentov schvaľujú poverení ľudia, ktorí dostanú informáciu o kolízii vo forme SMS správy.

Medzi funkcionality eRozvrh-u patria tiež:

- plánovač skúšania - modul na prípravu skúškových rozvrhov,
- systém na evidenciu dochádzky - na vyhodnotenie tzv. „mŕtvych duší“ (študentov, ktorí si predmet zapísali, ale nezúčastňovali sa jeho výučby),
- rezervačný systém miestností,
- vizualizácia na tlač - ponúka farebné rozlíšenie, či možnosť aplikovať rôzne filtre a nastavenia pre tlač,
- správne zobrazovanie obmedzení v rozvrhu - napr. zmena výučby v prípade štátnych sviatkov.

eRozvrh má aj určité nedostatky. Pri implementácii riešenia nebol použitý objektovo-orientovaný prístup, ktorý ponúka PHP. Zdrojový kód je štruktúrovaný tak, že každá stránka je vo vlastnom súbore, zdieľaná je iba práca s databázou, čím vzniká množstvo duplicitného kódu. Za nevýhodu považujeme aj nevyužitie prihlásenia cez AIS.

Výhody

- import dát z AIS-u a export dát pre AIS
- množstvo funkcionalít
- rieši podobné problémy ako sú na FEI

Nevýhody

- neprehľadný zdrojový kód
- bez prihlásenia cez AIS

1.3 Tvorba rozvrhu na FEI

Tvorbou semestrálnych rozvrhov sa na FEI zaoberá Mgr. Dávid Pancza, PhD, ktorý pri práci používa zastaralejší program WinRozvrhy (vytvorený pre potreby FEI). Ako sa spomína v práci [1] mojej predchodkyne, pokúšali sa ho nahradiť za ROGER (podkapitola 1.2.1), nedopadlo to však úspešne. Vygenerovaný rozvrh nespĺňal kvalitatívne kritériá a dodatočná manuálna úprava si vyžadovala veľké úsilie.

Na FEI pracujú na tvorbe aj tzv. lokálni rozvrhári, ktorí majú na starosti spracovanie predmetov a zozbieranie požiadaviek od pedagógov na jednotlivých pracoviskách. V niektorých prípadoch dokonca namiesto zozbieraných požiadaviek odovzdávajú hlavnému rozvrhárovi priamo vypracovaný rozvrh za im priradený ústav. Tieto rozvrhy sa potom snaží hlavný rozvrhár zakomponovať do hlavného semestrálneho rozvrhu, ktorý má však vyššiu prioritu.

Údaje, ktoré sa používajú pri tvorbe rozvrhu, pochádzajú z AIS vrátane informácií o počte študentov zapísaných na jednotlivé predmety (získané z predzázpisu). Požiadavky pedagógov sú v súčasnosti spracovávané papierovou formou, čo značne spomaľuje prípravu nového rozvrhu. Pri tvorbe nového rozvrhu sa používa rozvrh z predošlého akademického roku, avšak podľa informácií z osobného stretnutia sa zhruba 70 percent mení. Rozvrh je najprv zverejnený pre pedagógov a po vyriešení problémov je zverejnený aj pre študentov¹. Nasleduje import nového rozvrhu do AIS-u a prihlasovanie študentov na cvičenia.

Z informácií, ktoré sme získali od hlavného rozvrhára vyplynulo, že existuje množstvo problémov, ktoré by bolo možné zjednodušiť vhodným softvérovým riešením.

1.4 Cieľ práce

Cieľom práce je rozšíriť aplikáciu, ktorú vytvoril Ing. Martin Račák [2] podľa prototypu Ing. Emílie Knaperekovej [1] z roku 2014. Plánujeme preveriť aktuálnosť použitých technológií, pretože súčasťou minuloročnej práce [3] neboli žiadne zmeny na strane servera. Ďalej máme v pláne dokončiť neúplne implementovanú funkcionality a rozšíriť aplikáciu o ďalšie funkčnosti vrátane nových webových služieb pre front-endovú aplikáciu. Oboznámime sa tiež s procesom tvorby rozvrhu, aby sme vedeli zohľadniť rôzne výnimky pri implementácii aplikácie.

Zvolili sme proklientsky orientovanú tvorbu API pri vytváraní RESTful webových služieb s úmyslom znížiť množstvo požiadaviek na server. Štruktúru týchto služieb sme komunikovali s tvorcom klientskej aplikácie Bc. Adamom Kurfákom, ktorá je vytváraná súbežne s našou prácou.

¹<http://aladin.elf.stuba.sk/rozvrh/>

1.5 Špecifikácia požiadaviek

Pred začiatkom práce je dôležité definovať vlastnosti a funkcionality, ktoré má aplikácia spĺňať resp. ponúkať. Tieto požiadavky boli definované už v predošlých prácach ([1] a [2]). Tieto požiadavky považujeme stále za aktuálne, preto ich iba zosumarizujeme. Okrem funkcionality pre tvorbu semestrálnych a skúškových rozvrhov by mali medzi funkcionality, ktorú má serverová strana poskytovať, patriť:

- možnosť importu dát z AIS-u, ich premapovanie na interné štruktúry aplikácie a ich následný export,
- spracovanie požiadaviek a rôznych výnimiek pri tvorbe rozvrhu,
- konfigurovateľnosť nastavení aplikácie,
- generovanie nového rozvrhu z dát pripravených rozvrhárom,
- generovanie nového rozvrhu (z dát pripravených rozvrhárom) a návrhov na jeho úpravu,
- rozlíšenie rolí a právomocí používateľov,
- modul pre rezerváciu miestností,
- verzionovanie rozvrhov vrátane zálohy dát.

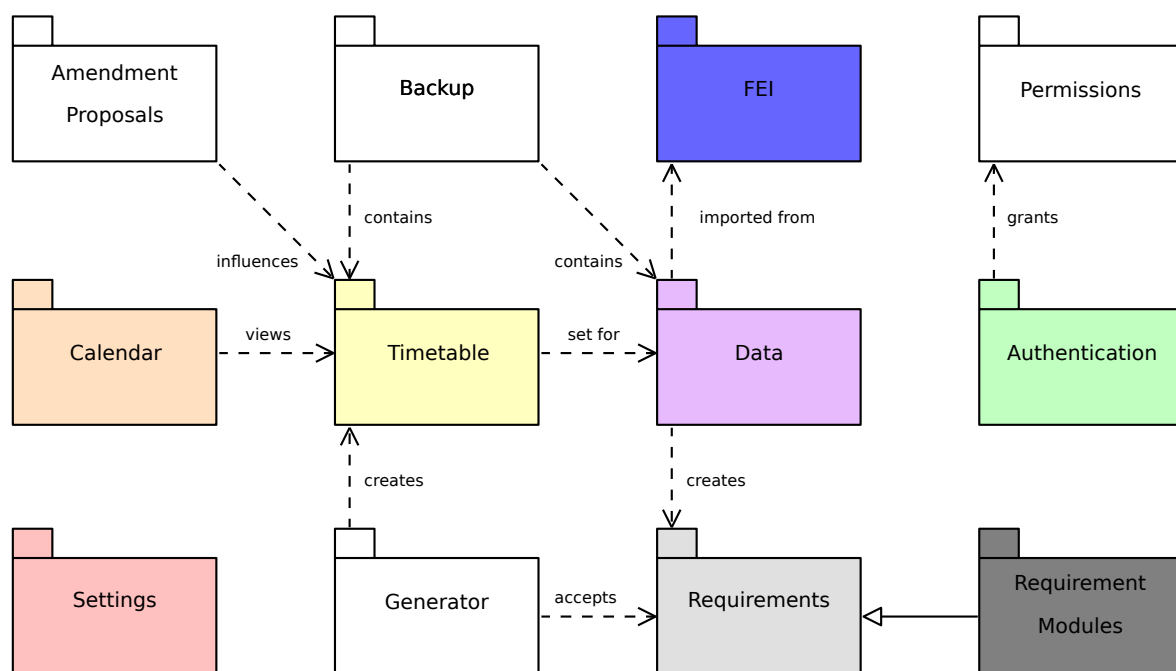
Jednoduchá a používateľsky prívetivá manuálna tvorba rozvrhov, či možnosť ich prezeráť patrí medzi požiadavky na front-endovú časť. Aby sme znížili čakanie používateľa na odozvu servera pri výpočte kolízií, ktoré nastávajú pri vytváraní rozvrhu, rozhodli sme sa presunúť túto funkčnosť na klienta.

2 Návrh riešenia

Táto kapitola obsahuje krátke zhrnutie predošlých riešení. Zároveň v nej predstavujeme návrh zmien voči súčasnemu riešeniu a detailnejšie rozoberáme dôležité pojmy a technológie, ktoré sme použili pri vypracovaní tejto práce.

2.1 Prehľad architektúry systému

Obrázok č.3 zobrazuje pôvodný návrh architektúry, ktorý bol predstavený v diplomovej práci Ing. Emílie Knaperekovej [1]. Farebne zvýraznené balíčky boli v tejto práci aspoň čiastočne implementované. Na výsledky, ktoré dosiahla, nadviazal vo svojej práci [2] Ing. Martin Račák. Ten presunul viac kompetencií na klienta, premenoval balíček Data na School, odstránil balíček Calendar (jeho úlohu prevzal klient), začal s importom vstupných údajov a upravil štruktúru ukladaných dát pre rozvrhy a rozvrhové akcie. Klientska časť dostala podobu jednostránkovej aplikácie (SPA), jej prototyp bol vytvorený pomocou programovacieho jazyka Elm².



Obr. 3: Pôvodná architektúra – diagram balíčkov (zoskupení tried) [1]

Ako posledný sa tejto téme venoval Ing. Dávid Bednár [3], ktorý riešil iba klientskú časť. Elm prototyp bol nahradený robustnejšou SPA vo framework-u Angular³.

²<https://elm-lang.org/>

³<https://angular.io/>

2.2 Úpravy aplikácie

V našej práci sa sústreďujeme iba na serverovú časť aplikácie, nadviazali sme teda na výsledok, ktorý dosiahol Ing. Martin Račák. Ten vo svojej práci [2] vytvoril novú webovú aplikáciu v programovacom jazyku Python 3, ktorá obsahovala túto funkcionálnosť:

- JWT autentifikáciu,
- vlastný príkaz pre import údajov zo súborov v CSV formáte,
- triedy reprezentujúce dátové modely pre importované údaje, rozvrhy a rozvrhové akcie,
- základné RESTful API s vygenerovanou dokumentáciou pre vytvorené triedy.

Import údajov bol implementovaný iba čiastočne, preto ho plánujeme dokončiť, aby sme vedeli pracovať s reálnymi údajmi.

Keďže pri tvorbe nového rozvrhu môže prichádzať k zmenám pri predmetoch (pridanie, vymazanie, zmena názvu a pod.), aplikácia musí byť schopná držať oddelene rozvrhové dáta pre starý a nový rozvrh. Ako jedna z možností sa núka použiť tzv. hierarchický databázový model⁴. Záznamy v databáze o predmetoch by sa teda cez cudzí kľúč odkazovali na rozvrh v tabuľke rozvrhov, ku ktorému sú priradené. Takýto spôsob ukladania dát sme vyhodnotili ako nevhodný, keďže predmety by sa duplikovali v rámci jednej tabuľky, čo by mohlo mať časom dopad na odozvu databázových dopytov.

Druhou možnosťou bolo vytvárať kópie databázových tabuliek, ktoré obsahujú informácie o rozvrhu. Dôležité v tomto prípade bolo, aby sa tieto kópie dali vytvárať za behu programu, keďže akcia na vytvorenie nového rozvrhu je vyvolaná rozvrhárom. Django, ktoré používame, pristupuje k databázovým tabuľkám pomocou ORM mapovača (modely sú naviazané na konkrétnu tabuľku).

Hľadali sme teda spôsob, ako vytvárať dynamické Django modely. Tejto problematike sa venovali knižnice **Runtime Dynamic Models**⁵ a **django-dynamic-model**⁶. Prvá z nich sa nevyvíja od roku 2011, preto sme ju vylúčili. Druhá knižnica patrí medzi nové (začiatok vývoja 2018), bohužiaľ pre nás ešte nepodporuje všetky dátové typy atribútov v Django modeloch, ktoré v aplikácii používame.

Nakoniec sme našli riešenie, ktoré oddeľuje dáta na úrovni databázových schém. Ide o multivlastníkovú architektúru, ktorú detailnejšie predstavujeme v podkapitole 2.6.

⁴<http://people.cs.pitt.edu/~chang/156/14hier.html>

⁵<https://dynamic-models.readthedocs.io/en/latest/index.html>

⁶<https://github.com/rvinzent/django-dynamic-models>

Aplikáciu sme sa rozhodli doplniť o samostatné moduly pre autentifikáciu a import údajov z AIS. Tie by mali obsahovať potrebnú logiku na premapovanie externých objektov na nami používané Django objekty. To umožní tieto moduly jednoducho vymeniť za iné. Taktiež sme sa rozhodli API o nové zdroje pre ukladanie požiadaviek pedagógov, rozvrhových kolízií a schém, ktorými reprezentujeme semestrálny rozvrh.

2.3 Webová služba

World Wide Web Consortium (W3C) definuje webovú službu (preložené z anglického spojenia *web service*) ako softvérový systém, navrhnutý podporovať interoperabilnú interakciu medzi zariadeniami cez sieť. [7] Interoperabilitu chápeme ako schopnosť rôznych systémov vzájomne spolupracovať, poskytovať si služby. Webová služba obsahuje rozhranie popísané v strojovo spracovateľnom formáte (napr. WSDL). Pri webových službách rozlišujeme dve hlavné skupiny [8]:

- webové služby spĺňajúce REST - hlavnou úlohou služby je spracovávať webové zdroje v XML formáte pomocou jednotnej množiny bezstavových operácií,
- svojvoľné - webová služba môže zverejniť akúkoľvek množinu operácií.

Obe triedy webových služieb používajú URI na identifikáciu zdrojov, pričom využívajú webové protokoly ako HTTP alebo SOAP 1.2.

V tejto práci sme sa rozhodli použiť architektúru REST na vytváranie webových služieb (detailnejšie popísaný v 2.3.1). Tieto webové služby umožňujú separáciu klienta a servera, sú ľahko integrovateľné, sú platformovo nezávislé a podporujú množstvo formátov (JSON, HTML alebo XML). REST API je v súčasnosti populárne aj medzi spoločnosťami ako Google, Facebook alebo Twitter.

2.3.1 REST

Representational state transfer (REST) je architektonický štýl pre návrh prepojených aplikácií, ktoré medzi sebou komunikujú cez sieť pomocou HTTP protokolu. Roy Thomas Fielding ho prvýkrát odprezentoval vo svojej dizertačnej práci [9]. Ide o spôsob správy vzťahu medzi klientom a serverom, pričom sa zameriava na rýchlosť a výkon začlenením znovupoužiteľných komponentov.

Roy Thomas Fielding definuje vo svojej práci [9] množinu obmedzení, ktoré sa používajú pri jeho aplikovaní. Radíme medzi ne:

- separáciu záujmov klienta a servera

- oddelenie záujmov používateľského rozhrania od záujmov serverovej časti zlepšuje prenosnosť používateľského rozhrania medzi rôznymi platformami a zvyšuje škálovateľnosť zjednodušením serverových komponentov,

- **bezstavovosť**

- každá požiadavka klienta na server dostáva v odpovedi rovnaké dáta, server si neukladá žiadne dáta navyše, manažment sedení (preložené z anglického spojenia *session management*) prebieha na strane klienta,
- nevýhodou je možné zníženie sieťovej výkonnosti kvôli opakovaným volaniam servera,

- **využívanie cache pamäti**

- vyžaduje sa implicitné alebo explicitné označenie požiadaviek, ktoré sa majú resp. nemajú ukladať v cache pamäti,
- odpoveď servera na určité volanie je uložená v cache pamäti klienta, opätovne sa použije pri rovnakých volaniach,
- používa sa zlepšenie sieťovej efektívnosti, škálovateľnosti a odozvy aplikácie pre používateľa, nevýhodou môže byť v prípade premenlivých dát ich neaktuálnosť,

- **jednotné rozhranie**

- oddeluje sa rozhranie od jeho implementácie, pričom rozhranie musí dodržiavať predpísané štandardy (identifikácia zdrojov, manipulácia so zdrojmi cez ich reprezentácie, samopopisné správy a hypermedia ako aplikačný stav (HATE-OAS)).

Kľúčový princíp REST-u zahŕňa rozdelenie API do logických zdrojov. Pod pojmom API chápeme kontrakt, vďaka ktorému je umožnená komunikácia medzi aplikáciami. Každá metóda zadaná v API má presne určenú štruktúru vstupných a výstupných dát.

Za zdroj môžeme považovať ľubovoľnú entitu, ktorú dokážeme identifikovať, napr. dokument, obrázok alebo dočasná služba (preložené z anglického spojenia *temporal service*). K zdrojom pristupujeme prostredníctvom priradených URL adries, ktoré sú jedinečné a majú tvar podstatného mena v množnom čísle (napr. `/tickets`). Manipulácia s nimi prebieha pomocou HTTP metód GET, POST, PUT, PATCH alebo DELETE, ktoré umožňujú vykonávať nad zdrojmi CRUD operácie.

Na prenos údajov medzi klientom a serverom sa používajú najčastejšie formáty JSON, HTML a XML. V našom prípade sme na reprezentáciu zdrojov použili formát JSON, ktorý je samopopisný a ľahký na pochopenie.

2.3.2 Návrh REST API

Pri definovaní API je nutné dodržiavať určité konvencie. Najprv je dôležité identifikovať objekty, ktoré budú predstavené ako zdroje. Pri pomenovaní zdrojov sa používajú zmysluplné podstatné mená v množnom čísle, nie slovesá. Za reprezentáciu zdroja sa považuje stav zdroja v akomkoľvek momente [10].

Odpoveď servera na požiadavku vracia okrem dát aj HTTP status kód, ktorým klienta informuje o spracovaní požiadavky. V prípade POST volania vracia status 201 **Created**, pri chybné vyskladanej požiadavke zo strany klienta 400 **Bad Request** alebo pri neočakávanej chybe servera 500 **Internal Server Error**.

K zdrojom pristupujeme pomocou HTTP sloviess (metód) POST, GET, PUT, PATCH a DELETE. Na príklade zdroja pre predmety si ukážeme, aký je ich význam:

- GET /courses/ - vracia zoznam všetkých predmetov,
- GET /courses/21/ - vracia zvolený predmet na základe identifikátora,
- POST /courses/ - vytvára nový predmet, v odpovedi sa posiela vytvorený zdroj doplnený o ID
- PUT /courses/21/ - kompletne nahrádza predmet s ID 21, v odpovedi sa posiela upravený zdroj
- PATCH /courses/21/ - čiastočne upravuje predmet s ID 21, v odpovedi sa posiela upravený zdroj
- DELETE /courses/21/ - slúži na zmazanie konkrétného predmetu.

Pre jednotlivé zdroje môžeme určiť, ktoré z HTTP metód implementujeme v rámci nášho API. Pomocou tzv. *query stringu*, ktorý je súčasťou URL, vieme nad zdrojmi vykonávať rôzne filtrovanie alebo vieme zvoliť zoradenie dát v odpovedi podľa vybraného parametra. Táto funkcionality sa hodí, ak chceme napr. získať predmety, ktoré patria konkrétnemu pedagógovi.

Pomocou zdrojov môžeme definovať rôzne hierarchie. URI v tomto prípade vzniká zrefazením URI pre konkrétny nadradený zdroj a URI podradeného zdroja (buď konkrétny zdroj alebo kolekcia zdrojov). Takto vieme previazať kolízie alebo rozvrhové akcie na

konkrétny rozvrh. Linka pre kolíziu s ID 5 v rozvrhu, ktorý má ID 1, vyzerá nasledovne: `/timetables/1/collisions/5/`.

Na prístup k väčšej časti zdrojov, ktoré sme zdefinovali v API, musí byť používateľ prihlásený. Po prihlásení získa JWT token, vďaka ktorému sa autentifikuje. Verejne dostupné dáta sú k dispozícii aj neprihláseným používateľom. Na zabezpečenie prenášaných dát používame HTTPS protokol.

2.4 Autentifikácia

Autentifikácia je proces overenia identity používateľa a skladá sa z 2 častí: identifikácie a autentizácie. Identifikácia používateľa prebieha na základe jedinečného kľúča, napr. login, prihlasovacie ID, a pod. Pri autentizácii sa preukazuje identita používateľa jedinečným príznakom (heslo, odtlačok prsta). V práci Ing. Martina Račáka [2] sú detailne analyzované 4 typy autentifikácie pre REST webové služby:

- HTTP basic autentifikácia,
- session manažment,
- autentifikácia na základe tokenu,
- autentifikácia na základe dotazu.

Ako najvhodnejšia bola v spomínanej práci [2] zvolená autentifikácia na základe tokenu, konkrétne JSON Web Token (JWT), ktorý detailnejšie opisujeme v podkapitole 2.4.1. V rámci našej práce sme existujúcu aplikáciu rozšírili aj o overenie používateľov pomocou prihlásenia do AIS-u, kde autentifikácia prebieha pomocou LDAP protokolu.

Na zvýšenie bezpečnosti prenášaných dát sa často používa HTTPS protokol, ktorý poskytuje šifrované spojenie. Dokážeme tak zabezpečiť ochranu proti tzv. *man in the middle* útokom (preložené z anglického spojenia ako *človek uprostred*), ktoré sa snažia odpočúvať komunikáciu medzi jej účastníkmi tak, že sa stanú aktívnym prostredníkom. Pre správne použitie HTTPS je potrebný digitálne podpísaný certifikát vydaný certifikačnou autoritou.

2.4.1 JWT autentifikácia

JSON Web Token (JWT) je otvorený štandard definovaný v RFC-7519 [11], ktorý určuje kompaktný a sebestačný spôsob bezpečného prenosu dát medzi účastníkmi vo formáte JSON [12]. Tieto dáta môžu byť považované za dôveryhodné a overené, pretože sú digitálne podpísané. Podpísať ich môžeme buď pomocou tajomstva (preložené z anglického

slova *secret*) za pomoci HMAC algoritmu alebo pomocou páru zloženého z verejného a privátneho kľúča s použitím šifrovania RSA alebo ECDSA.

Ide o reťazec, ktorý sa skladá z 3 častí: **hlavičky** (preložené z anglického slova *header*), **posielaných dát** (preložené z anglického slova *payload*) a **podpisu** (preložené z anglického slova *signature*). V hlavičke sa špecifikuje typ tokenu a šifrovací algoritmus, pričom vznikne JSON objekt, ktorý je kódovaný pomocou **Base64Url**. Posielané dáta sú vyskladané z **tvrdení** (preložené z anglického slova *claims*), ktoré predstavujú tvrdenia o nejakej entite (typicky o používateľovi) a ďalších dát. Tvrdenia delíme na:

- **registrované** - patria do množiny preddefinovaných tvrdení, ktoré síce nie sú povinné, ale odporúča sa ich použiť, napr. čas expirácie,
- **verejné** - tvrdenia vytvorené ľubovoľne používateľmi JWT tokenu. Odporúča sa ich zadefinovať v IANA JSON Web Token⁷ registri alebo ako URI, ktorá obsahuje menný priestor odolný voči kolíziám [12],
- **súkromné** - vlastné tvrdenia, ktoré si dohodnú aplikácie medzi sebou, nie sú nikde definované, preto je dôležité ich používať s opatrnosťou (nemali by mať rovnaký názov ako registrované alebo verejné tvrdenia [13]).

Posielané dáta sú taktiež kódované pomocou **Base64Url**. Zreťazenie hlavičky a prenášaných dát cez bodku a tajomstvo používame ako vstup do šifrovacieho algoritmu, ktorý je definovaný v hlavičke, čím vznikne **podpis**. Výsledný reťazec má nasledovnú štruktúru:

`<header>.<payload>.<signature>`

Jednotlivé časti JWT tokenu vrátane výsledného reťazca sú zobrazené na obr. 4.

JSON Web Token (JWT) sa využíva najmä pri autorizácii. Po úspešnom prihlásení sa pre používateľa vygeneruje jedinečný token, ktorý bude zahrnutý v každej následnej požiadavke používateľa na server, čím vieme overiť napr. prístupové práva pre danú URL. Ďalším spôsobom využitia je bezpečná výmena informácií. Vďaka digitálnemu podpisu vieme určiť identitu odosielateľa a zároveň vieme overiť, že správa nebola modifikovaná, pretože podpis sa vypočítava z hlavičky a posielaných dát.

2.4.2 LDAP

Lightweight Directory Access Protocol (LDAP) je otvorený protokol (definovaný v RFC 2251 [15]), ktorý slúži na ukladanie a získavanie dát z hierarchickej adresárovej štruktúry. Adresárový server (preložené z anglického spojenia *directory server*) predstavuje špeciálny

⁷<https://www.iana.org/assignments/jwt/jwt.xhtml>

záznam odvodený. Zároveň hodnota sa musí zhodovať so zadaným typom pre daný kľúč. Záznam identifikujeme podľa jedinečného mena (preložené z anglického spojenia *distinguished name* (skrátene **dn**)), ktorý interpretujeme sprava doľava. Pre Barbaru Jenson z obrázka 5 vyzerá nasledovne:

`dn=cn=Barbara Jenson, ou=Sales, o=Acme, st=California, c=US.`

Na rozdiel od relačných databáz LDAP server spracúva údaje, ktoré sa po zápise takmer nemenia. Využíva sa preto najmä v prípadoch, keď je potrebné rýchle získavanie dát. Poskytuje rôzne optimalizácie pre prácu s veľkými dátovými sadami a je vhodný na autentifikáciu používateľov, či rôzne vyhľadávania.

2.5 Autorizácia

Autorizácia je proces, pri ktorom sa zisťuje, či má osoba právomoc na vykonanie operácie. Nastáva hneď po autentifikácii a jej úlohou je priradiť prihlásenému používateľovi jeho roly a právomoci. V informatike sa používa na obmedzenie prístupu k dátam, častiam softvéru, či iným dôležitým aktívam.

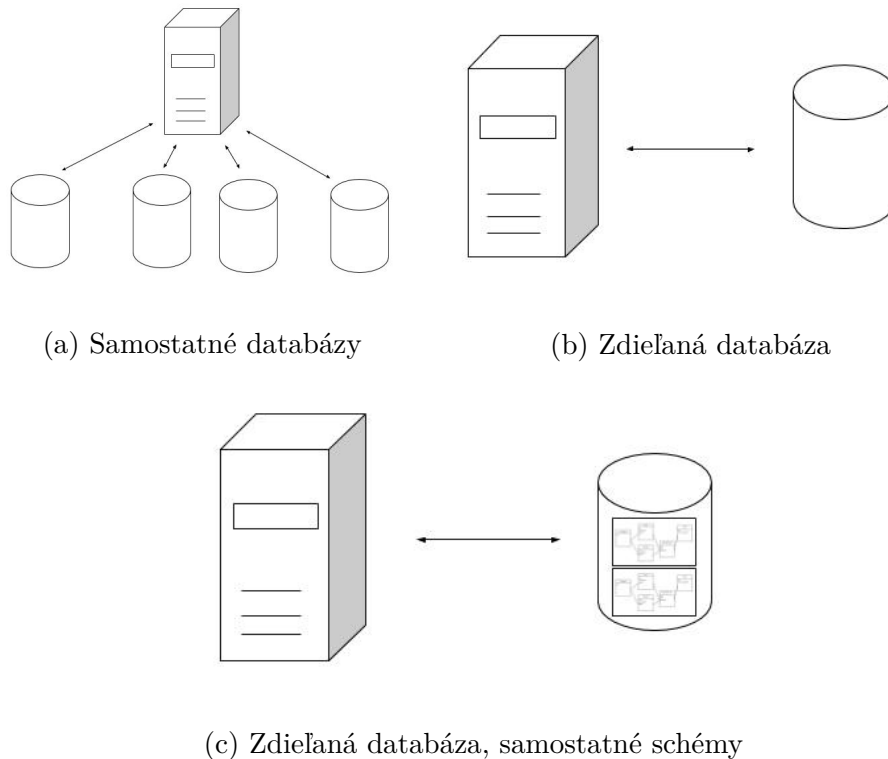
V procese tvorby rozvrhu sme identifikovali nasledovné roly (zoradené podľa oprávnení od najvyšších):

- hlavný rozvrhár,
- lokálny rozvrhár,
- pedagóg,
- študent,
- neprihlásený používateľ.

Hlavný rozvrhár má mať možnosť importovať vstupné dáta, môže ich manuálne modifikovať, môže vytvoriť hlavný rozvrh, zlúčiť rozvrhy lokálnych rozvrhárov do hlavného rozvrhu alebo zverejniť rozvrh pre učiteľov resp. pre verejnosť. Lokálny rozvrhár dokáže vytvoriť subrozvrh pre svoj ústav a vyplňať požiadavky jemu priradeným pedagógom. Pedagóg je oprávnený vyplňať požiadavky pre svoje predmety, vidí iba zverejnený rozvrh (svoje/všetky predmety).

Študent vidí iba zverejnený rozvrh (svoje/všetky predmety), neprihlásený používateľ vidí iba zverejnený rozvrh (všetky predmety). V nami vyvíjanej verzii aplikácia nebude rozlišovať medzi študentom a anonymným používateľom.

V aplikácii ešte existuje špeciálna rola superadmina, jeho úlohou je však iba spustiť inicializačné skripty a zdefinovať, kto bude mať rolu hlavného rozvrhára.



Obr. 6: Modely prístupu k dátam vo viacvlastníckovej architektúre [18]

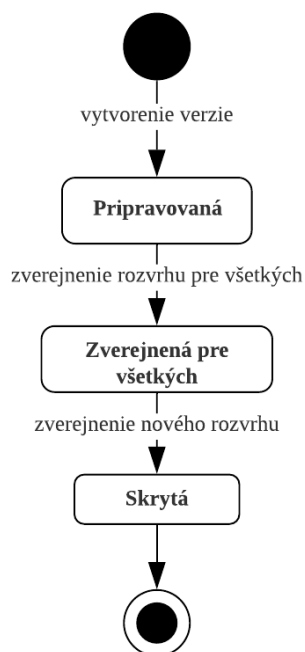
2.6 Viacvlastníková architektúra

Viacvlastníková architektúra (preložené z anglického spojenia *multi-tenant architecture*) sa používa pri tvorbe SaaS aplikácií. Viacerí vlastníci prístupujú k jednej inštancii aplikácie, ktorá umožňuje zdieľať spoločné dáta medzi všetkými vlastníkmi, a zároveň ukladá dáta patriace jednotlivým vlastníkovi oddelene. Ako sa spomína v článku [19] od pracovníkov z Microsoftu, na prístup k dátam sa používajú 3 prístupy:

- **izolovaný prístup** - každý vlastník má vlastnú databázu (obr. 6a),
- **zdieľaný prístup** - vlastníci zdieľajú jednu databázu a databázovú schému (obr. 6b).
- **poloizolovaný prístup** - vlastníci zdieľajú jednu databázu, každý vlastník má vlastnú databázovú schému (obr. 6c),

V našom prípade je najvhodnejší tretí spôsob, pretože pomocou neho vieme mať dáta o používateľoch v zdieľanej časti a rozvrhové dáta vieme oddeliť na úrovni semestrov

(schém). Vzniknú síce kópie týchto dát, ale vieme tak zabezpečiť spätnú kompatibilitu pre staršie rozvrhy (napr. zmazanie predmetu neovplyvní rozvrhové akcie v inej schéme).



Obr. 7: Stavový diagram pre schémy

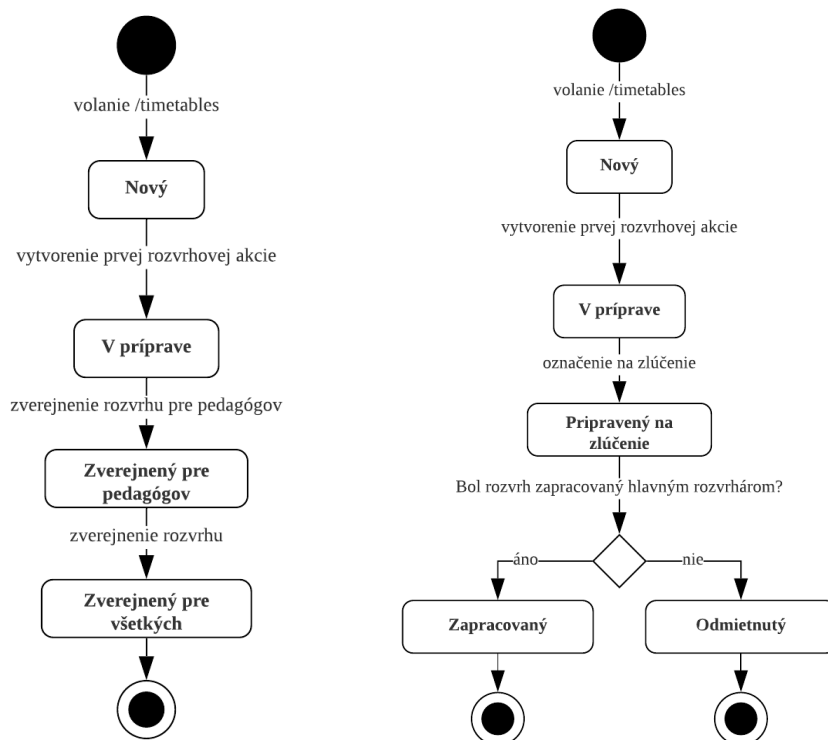
2.7 Stavy

Pri práci s aplikáciou je potrebné rozlišovať stavy, v ktorých sa nachádzajú schéma (reprezentuje semester) a vytváraný rozvrh. Znázornili sme ich pomocou UML stavových diagramov na obrázkoch č. 7 a č. 8.

Schéma (reprezentuje semestrálny rozvrh) sa po vytvorení dostane do stavu Pripravovaná (obr. č. 7). V tomto stave sú dáta dostupné iba pre rozvrhárov a pedagógov (po zverejnení pracovnej verzie rozvrhu). Zverejnením rozvrhu pre všetkých (hlavným rozvrhárom) sa rozvrhové dáta stanú viditeľnými aj pre neprihlásených používateľov. V nasledujúcom akademickom roku sa po zverejnení nového rozvrhu dostane do stavu Skrytá.

Na obrázku č.8a vidíme, do akých stavov sa môže dostať hlavný rozvrh. Prechody medzi stavmi má na starosti výhradne hlavný rozvrhár. Pri zverejnení rozvrhu pre pedagógov sa aktuálna pracovná verzia rozvrhu stáva needitovateľnou. Hlavný rozvrhár môže pokračovať buď zverejnením rozvrhu pre všetkých alebo vytvorením novej pracovnej verzie. Nová pracovná verzia môže byť odvodená od predošlej (dôjde k prekopírovaniu rozvrho-

vých dát, začína sa v stave Nový) alebo vytvoriť úplne novú.



(a) Stavový diagram rozvrhu pre hlavného rozvrhára (b) Stavový diagram rozvrhu pre lokálneho rozvrhára

Obr. 8: Stavové diagramy pre rozvrhy.

Rozvrh, vytvorený lokálnym rozvrhárom, prechádza stavmi znázornenými na obr. č. 8b. Keď lokálny rozvrhár usúdi, že jeho rozvrh je dokončený, označí rozvrh ako Pripravený na zlúčenie. Hlavný rozvrhár sa potom rozhodne, či dané zmeny použije vo svojom rozvrhu alebo ich zamietne.

3 Použité technológie

V tejto kapitole sú bližšie popísané technológie, ktoré sme použili pri implementácii tejto práce.

3.1 Python

Python je interpretovaný, vysoko-úrovňový, objektovo-orientovaný programovací jazyk s dynamickou sémantikou [20] vytvorený Guido van Rossumom v roku 1991. Podporuje moduly a balíky, čo nabáda k modularite aplikácií a znovupoužiteľnosti zdrojového kódu. Dôležitou vlastnosťou je tiež to, že sa dá jednoducho rozširovať pomocou programov napísaných v C/C++. V súčasnej dobe nabera na popularitu, keďže vďaka jednoduchšej syntaxi je ľahké sa ho naučiť. Python sa používa pri vývoji webových aplikácií (na strane servera), na matematické alebo vedecké výpočty, pri strojovom učení alebo na získavanie dát (preložené z anglického spojenia *data mining*).

Pri programovaní softvéru sa dodržiavajú konvencie spísané v dokumente PEP-8⁸.

3.2 Django

Django⁹ je vysoko-úrovňový webový framework napísaný v jazyku Python. Je multiplatformový, voľne dostupný a s open-source licenciou. Hlavnými cieľmi tohto frameworku sú jednoduchosť, škálovateľnosť, znovupoužiteľnosť (princíp DRY - neopakuj sa) a spoľahlivosť. Samotné Django bez externých knižníc ponúka napríklad podporu pre autentifikáciu, bezpečnosť, nástroje pre HTTP komunikáciu, objektovo-relačné mapovanie (ORM), správu databázových migrácií alebo administračný modul.

S dôrazom na princíp DRY a ušetrenie času sme pri práci použili knižnice tretích strán, z ktorých spomenieme najdôležitejšie:

- **django-import-export** - ponúka podporu pre import/export dát do rôznych formátov (napr. CSV, JSON, Excel), aktualizovaná na verziu 1.1.0,
- **django-mptt** - používa sa na správu hierarchických dát v databáze pomocou strojmovej štruktúry, aktualizovaná na verziu 0.9.0,
- **django-rest-framework** - poskytuje nástroje na tvorbu webových API,
- **psycopg2** - poskytuje podporu na prácu s PostgreSQL databázou,

⁸<https://www.python.org/dev/peps/pep-0008/>

⁹<https://www.djangoproject.com/>

- **djangoestframework-simplejwt** - poskytuje JWT autentifikáciu, náhrada knižnice `djangoestframework-jwt` kvôli kompatibilite s Django 2.2,
- **django-rest-swagger** - slúži na vygenerovanie Swagger/OpenAPI dokumentácie pre webové API,
- **django-tenants**¹⁰ - ponúka možnosti na oddelenie dát pre viacerých "vlastníkov" (preložené z anglického slova *tenant*) cez databázové schémy v SaaS aplikáciách,
- **django-python3-ldap** - podporná knižnica pre LDAP autentifikáciu,
- **django-fsm** - podporná knižnica na vytváranie stavových automatov,
- **python-decouple** - podporná knižnica na oddelenie nastavení od zdrojového kódu,
- **cx-oracle**¹¹ - ponúka podporu na prácu s Oracle databázami.

Práca môjho predchodcu [2] používala na strane servera Python 3.5 a Django vo verzii 1.11. Python sme aktualizovali na verziu 3.6 a Django na verziu 2.2, ktorá je najaktuálnejšia s dlhodobou podporou, keďže podpora Django 1.11 skončí v apríli 2020¹² (LTS).

3.3 PostgreSQL

PostgreSQL je open-source, objektovo-relačný systém pre správu databáz, ktorý používa a rozširuje jazyk SQL [21]. Beží na všetkých hlavných operačných systémoch, podporuje transakcie a dodržiava koncept ACID (od roku 2001). Kladie dôraz na rozširiteľnosť (ponúka možnosť definovať vlastné dátové typy alebo metódy) a dodržiavanie štandardov. Umožňuje ukladať komplexné dáta napr. vo formáte JSON alebo vo forme poľa údajov, ponúka spracovanie transakcií pomocou systému MVCC, dedičnosť tabuliek, vnorené transakcie a asynchrónnu replikáciu.

Verzia PostgreSQL, aktualizovaná v práci Ing. Dávida Bednára [3] na verziu 10.1 z pôvodnej verzie 9.5, bola nami zvýšená na verziu 10.7.

3.4 Git

Pri implementácii práce sme použili nástroj Git¹³. Ide o voľne dostupný distribuovaný systém s open-source licenciou pre správu obsahu [22]. Funguje na princípe jedného cen-

¹⁰<https://django-tenants.readthedocs.io/>

¹¹<https://cx-oracle.readthedocs.io/en/latest/installation.html>

¹²<https://static.djangoproject.com/img/release-roadmap.e844db08610e.png>

¹³<https://git-scm.com/>

trálneho repozitára a neobmedzeného množstva lokálnych repozitárov, ktoré sú jeho kópiou. Umožňuje to jednoduché zdieľanie údajov a zároveň vytvára množstvo záloh týchto dát.

Zmeny v lokálnom repozitári je nutné najprv potvrdiť (preložené z anglického slova *commit*)), čím sa vytvorí záznam o zmene a vzniká tak história zmien. Aby sa zmeny prejavili aj v centrálnom repozitári, je potrebné ich vložiť (preložené z anglického slova *push*). Git ponúka vytváranie vetiev (preložené z anglického slova *branch*), čím vzniká samostatná línia vývoja alebo označovanie dôležitých míľnikov pomocou tzv. značiek (preložené z anglického slova *tag*).

3.5 Vývojové prostredie

Práca je implementovaná vo vývojovom prostredí PyCharm Professional Edition na OS Windows 10 64-bit, kde bol nainštalovaný Python vo verzii 3.6. Na inštaláciu ďalších knižníc pre Python sme použili príkazový riadok (Python konzolu), konkrétne príkaz *pip* [23]. Aplikácia pracuje s PostgreSQL databázou vo verzii 10.7, pre prístup k dátam sme využili nástroj pgAdmin 4.2. Aplikáciu sme spúšťali na lokálnom HTTPS serveri, ktorý je súčasťou knižnice *django-sslserver*¹⁴. Webové služby sme testovali najmä pomocou programu Postman¹⁵, avšak používali sme aj vygenerovanú Swagger dokumentáciu. Na prácu s repozitárom Git sme použili program Sourcetree¹⁶, ktorý poskytuje intuitívne grafické rozhranie.

¹⁴<https://github.com/teddziuba/django-sslserver>

¹⁵<https://www.getpostman.com/>

¹⁶<https://www.sourcetreeapp.com/>

4 Implementácia a testovanie

Nasledujúca kapitola detailnejšie opisuje vybrané implementačné problémy a ich riešenie, pridanú novú funkcionálnosť a v závere aj testovanie aplikácie.

4.1 Zmeny v aplikácii

Keďže aplikácia je ešte vo fáze vývoja, vzhľadom na odporúčanie mnohých internetových zdrojov (napr. článok Vitora Freitas [24]) sme vytvorili vlastný model pre používateľa nazvaný **AppUser**. Dôvodom pre tento krok je to, že každá Django aplikácia je naviazaná na tzv. *User model* a prípadné rozširovanie tohto modelu v neskoršej fáze vývoja by si vyžadovalo nemalé úsilie.

Vlastný model pre používateľa sme vytvorili dedením od triedy **AbstractUser**, ktorá je súčasťou Django a obsahuje potrebné údaje o používateľovi, s ktorými Django pracuje na pozadí a zaregistrovali sme ho v *settings.py* prepísaním nastavenia **AUTH_USER_MODEL**. Pre naše potreby sme tento model ešte rozšírili o tituly pred menom a za menom.

Okrem zmien v databáze prebehla aj migrácia na novšiu verziu Django. Počas práce sme ju najprv aktualizovali na verziu 2.1 a následne na konci apríla na najnovšiu verziu 2.2. Migrácia so sebou priniesla tieto zmeny v implementácii:

- nahradili sme metódu `url()` za metódu `path()`, čím sa zjednodušila syntax pre zápis URL,
- vymenili sme knižnicu pre JWT autentifikáciu, pretože nepodporovala verziu Django 2.X,
- opravili sme chyby, ktoré vznikli pri prechode na novšie verzie tretostranných knižníc.

Keďže pri práci využívame Git repozitár, riešili sme aj vyňatie privátnych nastavení zo súboru *settings.py* (napr. tajomstvo na generovanie JWT tokenov alebo prihlasovacie údaje pre LDAP server a AIS databázu). Použili sme knižnicu **Python Decouple** [25], ktorá si do `config` objektu uloží nájdené premenné prostredia (preložené z anglického spojenia *environment variables*) a premenné zo súborov s príponou `.ini` alebo `.env`, ktoré sa nachádzajú v projekte.

Na výpise č.1 vidíme nastavenie dvoch premenných **SECRET_KEY** a **DEBUG**. Ak v `config` objekte nie je nastavená hodnota premennej **SECRET_KEY**, aplikácia pri spustení skončí chybou. V prípade premennej **DEBUG** chyba pri spustení aplikácie nenastane, pretože sme

zadefinovali prednastavenú hodnotu pomocou atribútu `default`. Keďže všetky premenné v `config` objekte sú uložené ako reťazce, pomocou atribútu `cast` ich dokážeme pretypovať.

```
SECRET_KEY = config('SECRET_KEY')
DEBUG = config('DEBUG', default=False, cast=bool)
```

Listing 1: Ukážka použitia knižnice Python Decouple

4.2 Autentifikácia

V aplikácii sa využívajú dva typy overenia identity používateľa: **LDAP autentifikácia** pomocou prihlasovacích údajov do AIS-u a **JWT autentifikácia**.

Pre prihlásenie do aplikácie sa používa LDAP autentifikácia, ktorú sme implementovali s použitím knižnice `django_python3_ldap` [26]. V súbore `settings.py` sme nastavili tieto parametre:

- `LDAP_AUTH_URL` - URL LDAP servera,
- `LDAP_AUTH_SEARCH_BASE` - reťazec, ktorým definujeme, na akej úrovni budeme v stromovej štruktúre hľadať používateľa,
- `LDAP_AUTH_USER_LOOKUP_FIELDS` - n-tica, pomocou ktorej definujeme polia, na základe ktorých hľadáme zhodu používateľa medzi LDAP serverom a našou databázou.

LDAP autentifikácia využíva klient-server model. Keď sa používateľ snaží prihlásiť, jeho prihlasovacie údaje sú odoslané na LDAP server. Ak je prihlásenie úspešné, údaje používateľa sú načítané z LDAP servera a uložené v lokálnej databáze v tabuľke pre model `AppUser`. Na premapovanie atribútov z databázy LDAP do lokálnej databázy aplikácie sa používa nastavenie `LDAP_AUTH_USER_FIELDS` zobrazený vo výpise č.2. Tento záznam sa vytvorí pri prvom prihlásení a je synchronizovaný pri každom ďalšom prihlásení. Knižnica `django_python3_ldap` ponúka ešte možnosť synchronizácie všetkých LDAP používateľov cez príkaz: `/manage.py ldap_sync_users`.

```
LDAP_AUTH_USER_FIELDS = {
    "username": "uid",
    "first_name": "givenName",
    "last_name": "sn",
    "email": "mail",
}
```

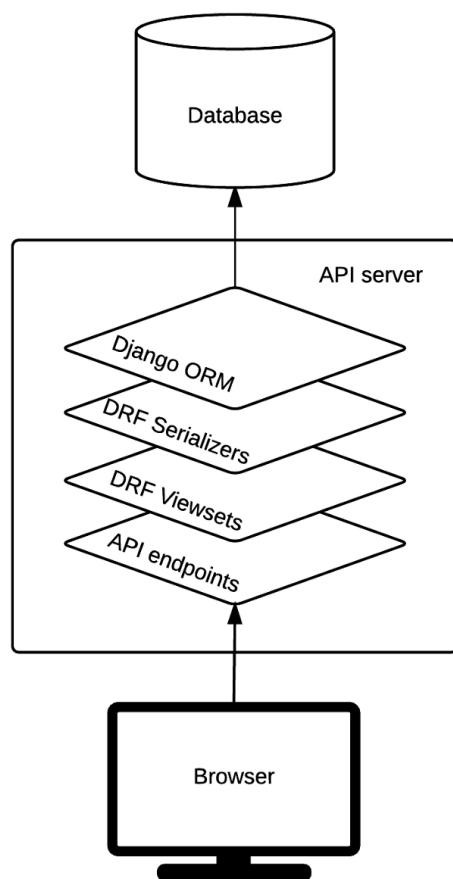
Listing 2: Ukážka premapovania LDAP atribútov do databázy

Po overení používateľa cez LDAP sa pre používateľa vygeneruje dvojica tokenov: **prístupový token** (preložené z anglického spojenia *access token*) a **obnovovací token**

(preložené z anglického spojenia *refresh token*). Prístupový token posiela front-endová aplikácia v každej svojej požiadavke na server, aby bolo možné overiť identitu používateľa. V prípade, že platnosť vyprší, použije sa obnovovací token na získanie nového prístupového tokenu. Po vypršaní obnovovacieho tokenu je používateľ odhlásený z aplikácie a musí sa opätovne prihlásiť.

4.3 Práca s API

Django Rest Framework (DRF) ponúka možnosti na jednoduché vytváranie REST API. Na obrázku č. 9 môžeme vidieť všetky vrstvy, cez ktoré prechádza požiadavka z prehliadača, keď chceme získať dáta zo servera.



Obr. 9: Spracovanie volania API v Django Rest Framework (DRF) [27]

API endpoints - zoznam URL adries, ktoré obsluhuje aplikácia, zväčša spravované pomocou smerovača (preložené z anglického slova *router*),

DRF viewsets - kontrolujú logiku posielaných dát, umožňujú nastaviť oprávnenia pre prístup k danému zdroju,

DRF serializers - slúžia na konverziu dát z HTTP požiadavky do Django objektov a naopak,

Django ORM - natívny Django ORM mapovač, ktorý slúži na efektívnu prácu s databázou.

Pri vytváraní API sme dedili od triedy `ModelViewSet` z knižnice Django Rest Framework (DRF), ktorá automaticky vytvorí pre daný zdroj všetky CRUD operácie. DRF umožňuje tiež vytvárať vnorené pohľady, čím vieme vytvárať napríklad požiadavky pedagógov. Použitím HTTP metódy `POST` na URL, reprezentujúcu zdroj pre požiadavky pedagóga, zároveň vytvárame nový záznam v tabuľke komentárov, na ktorý sa potom v požiadavke odkazujeme. Vtedy je nutné preťažiť metódy `create()` a `update()`, ktoré slúžia na vytváranie resp. úpravu zdrojov.

4.4 Import údajov

Pri importe údajov sme riešili 2 typy: import z CSV súborov a import dát z databázových pohľadov z AIS-u. V oboch prípadoch je dôležité dodržať správne poradie (napr. vybavenie miestností pred samotnými miestnosťami).

Pre správne fungovanie importu dát zo súborov sme doplnili ako prvý riadok hlavičky, vďaka ktorým knižnica **django-import-export** [28] dokázala tieto súbory namaľovať na naše Django modely. V prípadoch, keď neexistoval stĺpec pre ID, bolo nutné upraviť štruktúru CSV súborov pridaním prázdneho stĺpca, čím sme umožnili generovať ID automaticky.

Pri importe dát z AIS-u sme najprv zaregistrovali novú Oracle databázu, ktorú používa AIS, v súbore *settings.py* a vytvorili nový modul `fei_importexport`. Pre databázové pohľady sme vygenerovali Django modely pomocou príkazu `inspectdb`, ktorého výstup sme presmerovali do súboru *models.py* v spomínanom module.

Vygenerované Django modely sme ešte manuálne upravili ako odporúča dokumentácia [29] a pre každý z nich sme označili jeden z atribútov ako primárny kľúč. Dôležité bolo nastaviť meta atribút `managed=False`, vďaka ktorému Django nevytvára databázové tabuľky pre tieto modely. Následne sme vypublikovali URL v tvare `/zdroj/import`, ktoré premapovali údaje z databázy AIS-u do štruktúr, ktoré používa naša aplikácia.

4.5 Viacvlastníková architektúra

Na implementáciu viacvlastníkovej architektúry cez poloizolovaný prístup k dátam sme použili knižnicu **django-tenants** [30], ktorá poskytuje podporu pre prácu s databázovými schémami v PostgreSQL.

Najprv je dôležité definovať v nastaveniach aplikácie, ktoré dáta sú zdieľané medzi vlastníkmi (**SHARED_APPS**) a ktoré sa týkajú iba jednotlivých vlastníkov (**TENANT_APPS**) (zobrazené na obr. č. 10). Okrem toho je potrebné upraviť atribút **ENGINE** pri nastavení PostgreSQL databázy na `'django_tenants.postgresql_backend'` a taktiež nastaviť **DATABASE_ROUTERS** na hodnotu `'django_tenants.routers.TenantSyncRouter'`.

```
# apps synced with created tenants
TENANT_APPS = [
    'django.contrib.contenttypes',
    'timetables',
    'school',
    'requirements'
]

# apps synced with public schema
SHARED_APPS = [
    'django_tenants',
    'fei',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

Obr. 10: Ukážka zadefinovania zdieľaných a privátnych aplikácií

Knižnica **django-tenants** používa na rozhodnutie pre zvolenie správnej schémy databázové tabuľky, ktorých modely vzniknú odvodením od tried **TenantMixin** a **DomainMixin**. Trieda **TenantMixin** obsahuje iba jediný povinný atribút - názov schémy. V našej aplikácii sme vytvorili model dedením od tejto triedy s názvom **Version**. Ten sme rozšírili o používateľsky prívetivejší názov, keďže názov schémy musí spĺňať určitý formát a stav danej schémy (popísané v 2.7). Dôležitú úlohu zohrávajú aj atribúty **auto_create_schema** (slúži na automatické vytvorenie tabuliek definovaných v **TENANT_APPS** vo vzniknutej schéme, prednastavené ako pravda) a **auto_drop_schema** (príznak pre zmazanie schémy, ktorej záznam sme zmazali z tabuľky pre model **Version**, prednastavený ako nepravda). Pri databázových migráciách je nutné používať príkaz: **migrate_schemas** namiesto **migrate**

(priamo dostupné v Django), aby zmeny nastali v správnej schéme.

Rozhodovanie, ktorú schému aplikácia použije, prebieha na tzv. middleware vrstve. Knižnica **django-tenants** vyberá schému na základe domény, na ktorej beží aplikácia pomocou `django_tenants.middleware.main.TenantMainMiddleware`. V našom prípade predpokladáme, že aplikácia bude bežať na jednej doméne, museli sme teda nájsť riešenie, ktoré to zohľadní.

Riešenie ako túto situáciu vyriešiť nám ponúkla knižnica **django-tenant-schemas** [31], od ktorej je knižnica, ktorú používame, odvodená. V dokumentácii tejto knižnice sa opisuje riešenie pomocou vlastnej HTTP hlavičky. Ak klientska aplikácia potrebuje získať dáta špecifické pre určitého vlastníka, posieľa v požiadavke na server hlavičku `HTTP_TIMETABLE_VERSION`, ktorej hodnotou je názov schémy z tabuľky `fei_version`. Ak táto hlavička nie je nastavená, schéma je nastavená na hodnotu `public`.

Pri vytváraní novej verzie (schémy) semestrálneho rozvrhu je možné špecifikovať nepovinný parameter `parent_schema`. Hodnota tohto parametra definuje názov databázovej schémy, z ktorej sa dáta prekopírujú do novovytvorenej schémy pomocou vlastného príkazu `dump_tenant`. Tento príkaz na pozadí volá príkazy `dumpdata` a `loaddata`, ktoré sú súčasťou Django a slúžia na export resp. import údajov z/do databázy.

4.6 Roly a právomoci

Roly v našej aplikácii sme rozlišovali pomocou triedy `Group`, ktorá je natívnou súčasťou Django. Superadmin pri nasadení aplikácie spustí príkaz `init_app`, ktorý vytvorí 4 skupiny: hlavní rozvrhári, lokálni rozvrhári, pedagógovia a študenti. Jednotliví používatelia môžu byť zaradení do jednej alebo viacerých skupín.

Django Rest Framework (DRF) [32] ponúka možnosť definovať vlastné oprávnenia odvodením od triedy `BasePermission` z balíčka `rest_framework.permissions`. Na základe príslušnosti do jednotlivých skupín sme vytvorili tieto oprávnenia:

- `IsMainTimetableCreator` - pre hlavného rozvrhára,
- `IsLocalTimetableCreator` - pre lokálneho rozvrhára,
- `IsTeacher` - pre pedagóga.

4.7 Stavý

Správnosť prechodov medzi stavmi sme implementovali pomocou konečného automatu s použitím knižnice **django-fsm** [33]. Pre účely našej práce stačí povedať, že konečný automat je definovaný množinou stavov a prechodov medzi nimi. Pre správne fungovanie musí byť jeden z množiny stavov označený ako počiatočný a minimálne jeden ako konečný.

Použitím konečného automatu vieme predísť nesprávnym stavom požadovaných objektov v aplikácii.

Prechody zobrazené na obrázkoch č.8 a č. 7 sme zdefinovali pre modely **Version** z balíčka **fei** a **Timetable** z balíčka **timetables**. V rámci týchto modelov sme vytvorili metódy, ktoré sme označili formou anotácie **@transition**. Na obrázku č.11 môžeme vidieť, že pre zverejnenie rozvrhu pre učiteľov musí byť rozvrh v stave 'V príprave'. Po zverejnení pre učiteľov je ho možné zverejniť pre všetkých používateľov.

```
@transition(field=status, source=WORK_IN_PROGRESS, target=PUBLISHED_FOR_TEACHERS)
def publish_teachers(self):
    print("Publish working version for teachers")

@transition(field=status, source=PUBLISHED_FOR_TEACHERS, target=PUBLISH_PUBLIC)
def publish_public(self):
    print("Publish for everyone")
```

Obr. 11: Ukážka zdefinovania prechodov pre rozvrhy

Zverejniť rozvrh môže iba hlavný rozvrhár použitím linky `/timetables/<id>/publish`, ktorá interne zavolá metódu `publish_public`.

4.8 Dokumentácia API

Dokumentácia je súčasťou každého API. Jej prvá verzia bola vytvorená v predošlej práci [2] pomocou knižnice Django Rest Swagger [34]. Basic autentifikáciu sme nahradili za JWT autentifikáciu doplnením atribútu **SECURITY_DEFINITIONS** (výpis č. 3) do nastavení Swaggeru. Zmenou hodnoty atribútu **DEFAULT_PERMISSION_CLASSES** v nastaveniach pre DRF na `rest_framework.permissions.IsAuthenticated` sme upravili dostupnosť zdrojov iba pre prihlásených používateľov (okrem výnimiek ako je napr. URL pre prihlásenie, kde sme to nastavili na úrovni pohľadu).

```
'SECURITY_DEFINITIONS': {
    'api_key': {
        'type': 'apiKey',
        'in': 'header',
        'name': 'Authorization',
    },
}
```

Listing 3: Ukážka nastavenia JWT autentifikácie

JWT token potrebný pre prihlásenie je možné získať volaním metódy `/login`. Vo vyskakovacom okne pre prihlásenie (obr. 12) na stránke dokumentácie nastavíme prístupový token s predponou **Bearer** ako hodnotu atribútu *value*, čím nastavíme HTTP hlavičku

Authorization. Po úspešnom prihlásení sa zobrazia chýbajúce metódy API.

Available authorizations

Api key authorization

name: Authorization
in: header
value: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tll

Authorize

Cancel

Obr. 12: Vyskakovacie okno pre prihlásenie

Dokumentácia aktuálne neposkytuje možnosť zvoliť si databázovú schému, ktorá sa má pri volaní použiť, pretože na testovanie sme primárne používali program Postman [35].

4.9 Testovanie

Pri testovaní API sme použili voľne dostupný program Postman [35]. Ten umožňuje jednoducho vytvárať nové volania na server, ktoré sú zoskupené v tzv. zbierkach (preložené z anglického slova *collection*). Na úrovni zbierky je možné nastaviť globálne premenné, napr. URL servera. Ďalej Postman umožňuje vytvárať tzv. prostredia (preložené z anglického slova *enviroments*), čím dokážeme ľahko simulovať rôzne stavy aplikácie (napr. rôzne typy prostredia alebo používateľov). Reprezentujeme ho pomocou množiny párov kľúč-hodnota (preložené z anglického spojenia *key-value pairs*), kde kľúč reprezentuje názov premennej.

Premenné je možné používať v jednotlivých volaniach cez `{{nazov_premennej}}`. Na obrázku 13 je premenná **HOST** v URL globálna, premenné **jwt_token** a **timetable** sú premenné prostredia, ktoré reprezentuje používateľa s menom Martin.

Zbierky volaní a prostredia je možné jednoducho zdieľať s ďalšími používateľmi, rovnako je dostupný aj import resp. export vo formáte JSON.

GET timetables

timetables Examples (0)

GET {{HOST}}/timetables/ Send Save

Params Authorization Headers (9) Body Pre-request Script Tests Cookies Code Comments (0)

Headers (2)

	KEY	VALUE	DESCRIPTION***	Bulk Edit	Presets
<input checked="" type="checkbox"/>	Authorization	Bearer {{jwt_token}}	prístupový token		
<input checked="" type="checkbox"/>	Timetable-Version	{{timetable}}	požadovaná schema		
	Key	Value	Description		

Temporary Headers (7)

Body Cookies Headers (7) Test Results Status: 200 OK Time: 92 ms Size: 433 B Save Download

Pretty Raw Preview JSON

```

1 [
2   {
3     "id": 1,
4     "major_version": 1,
5     "minor_version": 0,
6     "owner": 1,
7     "name": "Letný semester 2018/2019",
8     "created_at": "2019-05-12T19:25:31.904981+02:00",
9     "updated_at": "2019-05-12T19:25:31.904981+02:00",
10    "status": "NEW"
11  }
12 ]

```

Obr. 13: Ukážka volania GET /timetables/

Záver

V tejto práci sme nadviazali na prácu Ing. Martin Račáka [2], ktorý vytvoril webovú službu s REST API a prototyp webového klienta v jazyku Elm. Cieľom práce bolo oboznámiť sa s použitými technológiami na strane servera, preveriť ich aktuálnosť, dokončiť rozpracované funkcionality a doplniť serverovú časť o novú funkčnosť.

Na úvod sme sa stretli s tvorcom rozvrhového systému na Fakulte chemickej a potravinárskej technológie, ktorý nám predstavil aplikáciu eRozvrh. Nasledovalo stretnutie s rozvrhárom pre semestrálne rozvrhy na FEI, ktorý nás oboznámil s procesom jeho tvorby a s problémami, s ktorými sa pritom potýka. Oboznámili sme sa s predošlou prácou a našťudovali si princípy REST API.

V rámci vývoja tejto práce sme riešili najmä prácu s dátami. Dokončili sme import dát a rozšírili REST API o nové zdroje. Ďalej sme sa venovali riešeniu oddelenia rozvrhových dát, kde nás do istej miery limitoval Django ORM mapovač. Hľadali sme teda možnosti ako to vyriešiť tak, aby Django ORM mapovač ostal zachovaný, čo sa nám nakoniec podarilo pomocou viacvlastníckovej architektúry. Integrovali sme tiež prihlásenie do aplikácie pomocou prihlasovacích údajov do AIS-u a taktiež import aktuálnych dát. Pri práci sme využívali Git repozitár, ktorý sme zdieľali s autorom súbežne vyvíjanej klientskej aplikácie. Klient tak mohol pracovať s aktuálnymi službami a dátami. Vyriešili sme aj oddelenie privátnych nastavení aplikácie od zdrojového kódu.

Aplikácia v aktuálnej podobe nie je vhodná na nasadenie, keďže ešte neobsahuje kompletnú funkčnosť. Tento stav je spôsobený náročnosťou samotného procesu tvorby rozvrhov, ale aj použitím pre nás nových technológií. Podarilo sa nám však rozšíriť aplikáciu o novú funkcionality napr. import dát zo súborov alebo z AIS-u, rozdelenie dát pre jednotlivé rozvrhy, LDAP autentifikáciu a niekoľko nových služieb pre klientsku aplikáciu.

Okrem pridania ďalších funkcionalít do nášho systému, vidíme priestor na zlepšenie nami implementovaných funkčností, z ktorých spomenieme napríklad:

- asynchrónne spracovanie časovo náročných akcií na strane servera,
- úprava dokumentácie API, aby bolo možné vybrať schému,
- použitie grafových databáz.

Zoznam použitej literatúry

1. KNAPERREKOVÁ, Emília. *Rozvrhový systém pre vysoké školy*. 2014. Diplomová práca. EČ: FEI-5384-56111.
2. RAČÁK, Martin. *Rozvrhový systém pre FEI*. 2017. Diplomová práca. EČ: FEI-5384-53920.
3. BEDNÁR, Dávid. *Tvorba užívateľského rozhrania k rozvrhovému systému pre FEI*. 2018. Diplomová práca. EČ: FEI-5384-46049.
4. WOODS, Damien a TRENAMAN, Adrian. *Simultaneous Satisfaction of Hard and Soft Timetable Constraints for a University Department Using Evolutionary Timetabling*. 1999. Dostupné tiež z: <https://pdfs.semanticscholar.org/af0b/f88262ba9be1cc214b2b818d0b508bda0244.pdf>. Diplomová práca.
5. IS4U. *Roger: Rozumné generovanie rozvrhov* [online] [cit. 2018-12-03]. Dostupné z: <https://www.rozvrhy.eu/sk/uvod>.
6. ŠTULIĆ, Danijel. *Prime Timetable* [online] [cit. 2018-12-03]. Dostupné z: <https://www.primetimetable.com/>.
7. W3C. *Web Services Glossary* [online] [cit. 2019-05-12]. Dostupné z: <https://www.w3.org/TR/ws-gloss/>.
8. W3C. *Web Services Architecture* [online] [cit. 2019-05-12]. Dostupné z: <https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest>.
9. FIELDING, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures* [online]. 2000 [cit. 2019-05-03]. Dostupné z: <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>. Dizertačná práca.
10. *What is REST* [online] [cit. 2019-05-12]. Dostupné z: <https://restfulapi.net/>.
11. JONES, M., BRADLEY, J. a SAKIMURA, N. *JSON Web Token (JWT)* [online]. RFC Editor, 2015 [cit. 2019-04-27]. ISSN 2070-1721. Dostupné z: <https://tools.ietf.org/html/rfc7519>. RFC. RFC Editor.
12. *Introduction to JSON Web Tokens* [online] [cit. 2019-05-07]. Dostupné z: <https://jwt.io/introduction/>.
13. *JSON Web Token Claims* [online] [cit. 2019-05-13]. Dostupné z: <https://auth0.com/docs/tokens/jwt-claims>.

14. *JSON Web Token* [online] [cit. 2019-05-07]. Dostupné z: <https://artsy.github.io/blog/2016/10/26/jwt-artsy-journey/>.
15. WAHL, M., HOWES, T. a KILLE, S. *Lightweight Directory Access Protocol (v3)* [online]. RFC Editor, 1997 [cit. 2019-05-11]. Dostupné z: <https://www.ietf.org/rfc/rfc2251>. RFC. RFC Editor.
16. ING. PAVEL HOROVČÁK, CSc. *Web databázy LDAP* [online] [cit. 2019-04-25]. Dostupné z: <http://omega.tuke.sk/pj/foлие/ldap.pdf>.
17. FOUNDATION, OpenLDAP. *Introduction to OpenLDAP Directory Services* [online] [cit. 2019-05-11]. Dostupné z: <http://www.openldap.org/doc/admin24/intro.html>.
18. XIMENES, Filipe. *Multitenancy: juggling customer data in Django* [online]. 2017 [cit. 2019-05-10]. Dostupné z: <https://www.vinta.com.br/blog/2017/multitenancy-juggling-customer-data-django/>.
19. CHONG, Frederick, CARRARO, Gianpaolo a WOLTER, Roger. *Multi-Tenant Data Architecture* [online]. 2016 [cit. 2019-05-10]. Dostupné z: <http://ramblingsofraju.com/wp-content/uploads/2016/08/Multi-Tenant-Data-Architecture.pdf>.
20. FOUNDATION, Python Software. *What is Python?* [online] [cit. 2019-05-05]. Dostupné z: <https://www.python.org/doc/essays/blurb/>.
21. GROUP, The PostgreSQL Global Development. *What is PostgreSQL?* [online] [cit. 2019-05-05]. Dostupné z: <https://www.postgresql.org/about/>.
22. *Git* [online] [cit. 2019-05-05]. Dostupné z: <https://git-scm.com/>.
23. *Pip - inštalačný nástroj* [online] [cit. 2019-05-05]. Dostupné z: <https://pypi.org/project/pip/>.
24. FREITAS, Vitor. *How to Implement Multiple User Types with Django* [online] [cit. 2019-03-22]. Dostupné z: <https://simpleisbetterthancomplex.com/tutorial/2018/01/18/how-to-implement-multiple-user-types-with-django.html>.
25. BASTOS, Henrique. *Python Decouple GIT repozitár* [online] [cit. 2019-05-12]. Dostupné z: <https://github.com/henriquebastos/python-decouple/>.
26. HALL, Dave. *django-python3-ldap GIT repozitár* [online] [cit. 2019-04-10]. Dostupné z: <https://github.com/etianen/django-python3-ldap>.

27. LEONTIEV, Anthony. *Django Rest Framework stack* [online] [cit. 2019-05-12]. Dostupné z: <https://medium.com/altschool-engineering/dynamic-apis-in-django-c40ae64d5c70>.
28. MIHELAC, Bojan. *Creating import export resources* [online] [cit. 2019-05-07]. Dostupné z: <https://django-import-export.readthedocs.io/en/latest/>.
29. FOUNDATION, Django Software. *Django dokumentácia* [online] [cit. 2019-05-08]. Dostupné z: <https://docs.djangoproject.com/en/2.2/>.
30. TURNER, Thomas a CARNEIRO, Bernardo Pires. *django-tenants dokumentácia* [online] [cit. 2019-04-10]. Dostupné z: <https://django-tenants.readthedocs.io/en/latest/index.html>.
31. CARNEIRO, Bernardo Pires. *django-tenant-schemas dokumentácia* [online]. 2018 [cit. 2019-04-10]. Dostupné z: <https://django-tenant-schemas.readthedocs.io/en/latest/index.html>.
32. LTD, Encode OSS. *Django REST framework* [online] [cit. 2019-05-05]. Dostupné z: <https://www.django-rest-framework.org/>.
33. TURNER, Thomas a CARNEIRO, Bernardo Pires. *django-fsm Git repozitár* [online] [cit. 2019-05-12]. Dostupné z: <https://github.com/viewflow/django-fsm>.
34. GROUP, The PostgreSQL Global Development. *What is PostgreSQL?* [online] [cit. 2019-05-05]. Dostupné z: <https://django-rest-swagger.readthedocs.io/en/latest/>.
35. *Postman* [online] [cit. 2019-05-05]. Dostupné z: <https://www.getpostman.com/>.

Prílohy

A	Štruktúra elektronického nosiča	II
B	Technická dokumentácia	III

A Štruktúra elektronického nosiča

/thesis

- zdrojové súbory tejto práce

/img

- použité obrázky

/includes

- pomocné .tex súbory

/bibliography.bib

- bibliografia

/dp_maksin.tex

- hlavný .tex súbor

B Technická dokumentácia

Zdrojový kód aplikácie je zverejnený v GitHub repozitári¹⁷.

Vývoj aplikácie prebieha na prostredí, kde je nutné mať nainštalovaný Python 3 a PostgreSQL 10. Postup pre spustenie aplikácie na lokálny vývoj je spísaný v súbore `README.md`, v ktorom je tiež zdokumentované spustenie webového servera určeného iba pre vývoj. Pre správne fungovanie aplikácie je potrebné nastaviť nasledujúce premenné:

- `SECRET_KEY` - tajomstvo na generovanie JWT tokenov,
- `ALLOWED_HOSTS` - pole reťazcov s názvami povolených domén, resp. hostiteľských mien,
- `LDAP_AUTH_URL` - URL LDAP servera,
- `LDAP_AUTH_SEARCH_BASE` - reťazec, ktorým definujeme, na akej úrovni budeme v stromovej štruktúre hľadať používateľa,
- `DB_IMPORT_NAME` - názov AIS databázy,
- `DB_IMPORT_USER` - meno používateľa pre AIS databázu,
- `DB_IMPORT_PASSWORD` - heslo používateľa pre AIS databázu,
- `DB_IMPORT_HOST` - hostiteľské meno databázového servera,
- `DB_IMPORT_PORT` - port pre AIS databázu,
- `DEBUG` - povolenie *debug* módu, nepovinná.

Tieto premenné je možné nastaviť na úrovni prostredia, kde je aplikácia nasadená alebo v súbore s príponou `.ini` alebo `.env` v umiestnenom v projekte. Bližšie informácie sa nachádzajú na stránke knižnice Python Decouple¹⁸.

V produkčnom prostredí odporúčame použiť plnohodnotný webový server napr. Apache HTTP Server alebo Nginx. Na strane servera bude pravdepodobne nutné povoliť nami definovanú HTTP hlavičku `HTTP_TIMETABLE_VERSION`, na ktorú sa aplikácia spolieha. Na

¹⁷<https://github.com/matusjokay/Elisa>

¹⁸<https://github.com/henriquebastos/python-decouple/>

zabezpečenie webovej služby je nevyhnutné použiť HTTPS protokol s digitálne podpísaným certifikátom. Bližšie informácie pre nasadenie Django aplikácií do produkcie sa nachádzajú v jeho dokumentácii¹⁹.

¹⁹<https://docs.djangoproject.com/en/2.2/howto/deployment/>