SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: FEI-5384-53920

ROZVRHOVÝ SYSTÉM PRE FEI

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 2017 **Bc. Martin Račák**

Slovenská technická univerzita v Bratislave
Ústav informatiky a matematiky

Fakulta elektrotechniky a informatiky
Akademický rok: 2016/2017
Evidenčné číslo: FEI-5384-53920

STU FEI

ZADANIE DIPLOMOVEJ PRÁCE

Študent:	**Bc. Martin Račák**
ID študenta:	53920
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**

Jazyk, v ktorom sa práca vypracuje: slovenský jazyk

Špecifikácia zadania:

V roku 2014 vznikol v rámci DP prototyp rozvrhového systému pre vysoké školy (Emília Knapereková, 2014). Cieľom tejto práce je aktualizácia projektu a pokračovanie v jeho vývoji.

Úlohy:
1. Oboznámte sa s požiadavkami kladenými na tvorbu rozvrhu pre FEI.
2. Oboznámte sa s DP Rozvrhový systém pre vysoké školy.
3. Aktualizujte stav implementácie vzhľadom na súčasné možnosti použitých frameworkov.
4. Pokračujte vo vývoji aplikácie.
5. Zhodnoťte prínos práce.

Zoznam odbornej literatúry:

1. Knapereková, E. – Gallo, O. *Rozvrhový systém pre vysoké školy.* Diplomová práca. Bratislava : FEI STU, 2014. 74 s.

Riešenie zadania práce od: 19. 09. 2016

Dátum odovzdania práce: 19. 05. 2017

SLOVENSKÁ TECHNICKÁ UNIVERZITA
V BRATISLAVE
Fakulta elektrotechniky a informatiky
Ústav informatiky a matematiky
Ilkovičova 3, 812 19 Bratislava

L.S.

Bc. Martin Račák
študent

prof. RNDr. Otokar Grošek, PhD.
vedúci pracoviska

prof. Dr. Ing. Miloš Oravec
garant študijného programu

Ďakujem Mgr. Ing. Matúšovi Jókayovi, PhD. za jeho dôvtipné rady a odborné vedenie pri písaní tejto práce. A mojej rodine za všemožnú podporu počas celého môjho štúdia.

SÚHRN

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Študijný program:	Aplikovaná informatika
Autor:	Bc. Martin Račák
Diplomová práca:	Rozvrhový systém pre FEI
Vedúci záverečnej práce:	Mgr. Ing. Matúš Jókay, PhD.
Miesto a rok predloženia práce:	Bratislava 2017

Tvorba školských rozvrhov je komplexný problém, ktorý trápi väčšinu vzdelávacích inštitúcií vrátane FEI STU. Aj z tohto dôvodu vznikol v rámci diplomovej práce Rozvrhový systém pre vysoké školy (Knapereková, 2014) prototyp rozvrhového systému, ktorý má za cieľ tento problém riešiť a z ktorého sme v našej práci vychádzali. Oboznámili sme sa s problematikou rozvrhovania a požiadavkami kladenými na tvorbu rozvrhov na FEI. Prototyp systému sme aktualizovali a navrhli sme postup jeho ďalšieho vývoja. Systém sme rozdelili na samostatnú serverovú a klientskú časť, čo zo sebou prináša viacero výhod a zvyšuje flexibilitu celého systému. Výsledkom implementácie navrhnutých zmien je prototyp webovej služby, napísanej v Pythone a frameworku Django a webového klienta, napísanom v Elme, ktoré spolu komunikujú cez REST API.

Kľúčové slová: rozvrhy, rozvrhový systém, klient-server, REST, webová služba

ABSTRACT

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

Study programme:	Applied informatics
Author:	Bc. Martin Račák
Master's thesis:	Timetabling system for FEI
Supervisor:	Mgr. Ing. Matúš Jókay, PhD.
Place and year of submission:	Bratislava 2017

Creation of school timetables is a complex problem which affects most educational institutions including FEI STU. This is one of the reasons why master thesis Timetable Creator for Universities (Knapereková, 2014) contains a prototype of timetabling system which aspires to solve this problem and serves as the basis for our thesis. We familiarized ourselves with issues connected to timetabling and requirements for timetable creation at FEI. We updated the prototype and devised an approach to its following development. The system was split into an independent server and client which brings with itself several benefits and increases flexibility of the whole system. The result of implementing the proposed changes is prototype of a web service written in Python and Django framework and a web client written in Elm that communicate together through a REST API.

Keywords: timetables, timetabling system, client-server, REST, web service

Obsah

Zoznam obrázkov

Zoznam výpisov

Zoznam skratiek

ACID	Atomicity, Consistency, Isolation, Durability
AIS	Akademický Informačný Systém
AJAX	Asynchronous JavaScript And XML
API	Application Programming Interface
CSRF	Cross-site request forgery
CSS	Cascading Style Sheets
CSV	Comma-separated values
DOM	Document Object Model
DRY	Don't Repeat Yourself
FEI	Fakulta Elektrotechniky a Informatiky
GNU	GNU's not Unix
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
J2EE	Java 2 Platform, Enterprise Edition
JS	JavaScript
JSON	JavaScript Object Notation
JWT	JSON Web Token
MPTT	Modified Pre-order Traversal Tree
MVC	Model-View-Controller
ORM	Object-relational Mapping
OS	Operačný Systém
PDF	Portable Document Format
REST	Representational State Transfer
RSA	RSA kryptosystém, iniciály jeho tvorcov
SPA	Single-page Application
SQL	Structured Query Language
STU	Slovenská Technická Univerzita v Bratislave
TEA	The Elm Architecture
TLS	Transport Layer Security

UI	User Interface
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XML	Extensible Markup Language
XSS	Cross-site scripting
ÚIM	Ústav informatiky a matematiky

Úvod

Rozvrhový systém je softvér, ktorý slúži na zhotovenie rozvrhu. Vo vzdelávacích inštitúciách sa rozvrh používa na alokáciu študentov, učiteľov a miestností (priestorov) do časovo ohraničených udalostí. Vytvoriť rozvrh tak, aby sme vylúčili, alebo aspoň minimalizovali, kolízie medzi jeho jednotlivými prvkami a zároveň dodržali požadované obmedzenia je spletitý problém. Veľké požiadavky na takýto systém majú obzvlášť vysoké školy a univerzity, kde sa rozvrh tvorí spravidla niekoľko krát za rok, na semester aj skúškové obdobie, pri pomerne veľkom počte vstupných parametrov.

V našej práci sme sa oboznámili s viacerými existujúcimi rozvrhovými systémami, vrátane prototypu, ktorý vznikol ako súčasť diplomovej práce Rozvrhový systém pre vysoké školy (Emília Knapereková, 2014) [2]. Navrhli sme niekoľko možností ako ho vylepšiť a pokračovali sme v jeho vývoji s dôrazom na potreby FEI STU, kde by mal byť aj nasadený, s cieľom zvýšiť efektivitu procesu tvorby rozvrhu na fakulte.

V kapitole 1 analyzujeme problém tvorby rozvrhov a niekoľko softvérových systémov, ktoré ho riešia. Kapitola 2 obsahuje sumarizáciu požiadaviek na rozvrhový systém a opis súčasného stavu rozvrhovania na FEI. Náš návrh ďalšieho postupu pri aktualizácii a vývoji prototypu systému sme opísali v kapitole 3. Nasleduje prehľad použitých technológií v kapitole 4. Postup pri implementácii zmien a fungovanie systému sme popísali v kapitole 5.

1 Analýza problému

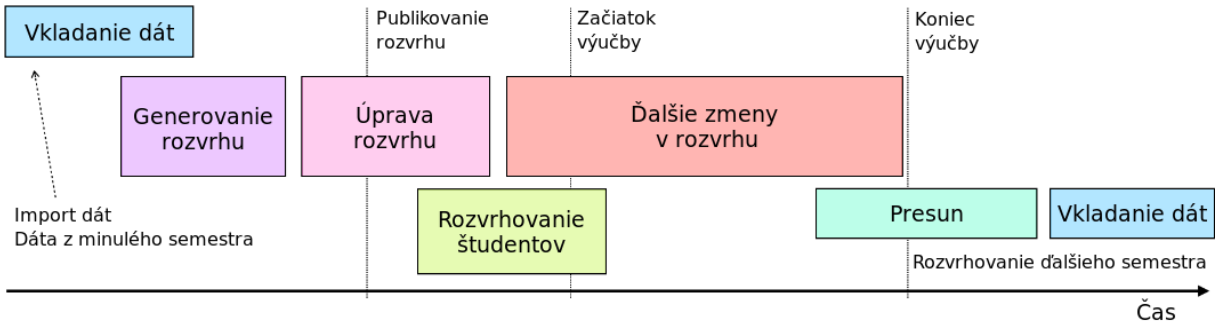
Prv než existovali počítače, školské rozvrhy zostavoval ručne rozvrhár, ktorý ich následne fyzicky zverejnil tak, aby k nim študenti a vyučujúci mali prístup. Príchod počítačov a internetu umožnil nielen ich jednoduchšiu publikáciu online, ale tiež možnosť ich automaticky generovať. Keďže sa však jedná o NP-úplný optimalizačný problém, neexistuje efektívny algoritmus na jeho absolútne riešenie v polynomiálnom čase. Generátory rozvrhov preto zvyčajne využívajú rôzne typy aproximačných algoritmov: evolučné algoritmy, lokálne vyhľadávanie, greedy algoritmy, dynamické programovanie a pod.

Osobitným problémom je spôsob, akým rozvrh prezentovať používateľom tak, aby používateľské rozhranie bolo účinné a jednoduché na používanie. Tiež je potrebné zaistiť čo najefektívnejšie vkladanie požiadaviek do systému a manuálnu úpravu rozvrhov. Zdá sa, že aj keď rozvrhový softvér je v dnešnej dobe pomerne efektívny v generovaní rozvrhov, ešte stále existuje značný potenciál na jeho zlepšenie v oblasti komunikácie s používateľmi.

1.1 Proces tvorby rozvrhu

Tvorba vzdelávacieho rozvrhu na semester alebo skúškové obdobie je optimalizačný problém, ktorého cieľom je z množiny všetkých prípustných riešení nájsť to najlepšie. Vyhľadávací priestor riešení býva obmedzený požiadavkami či už časovými (kedy), priestorovými (kde), personálnymi (kto), prípadne ďalšími. Tieto obmedzenia je možné rozdeliť na tvrdé, ktoré by rozvrh mal spĺňať úplne (vyučujúci a študenti môžu byť naraz v jednom časovom okamihu iba na jednom mieste, miestnosti majú obmedzenú kapacitu) a mäkké, pri ktorých hľadáme ich optimálnu kombináciu (vyučujúci preferujú určité časové úseky, snaha vyhnúť sa prvým a posledným časovým úsekom). [3] Tieto požiadavky sa v čase, teda aj v priebehu obdobia, no ktoré bol rozvrh zostavený, môžu meniť. Počas daného obdobia môže byť napr. potrebné do rozvrhu zaradiť predtým neplánovanú jednorázovú udalosť s vysokou prioritou (konferencia, mimovýučbová prednáška).

Zodpovednosť za tvorbu a úpravu rozvrhov máva vo vzdelávacích inštitúciach jeden alebo viacero rozvrhárov. Rozvrh môže byť generovaný automaticky, vytvorený manuálne rozvrhárom, prípadne môže ísť o kombináciu týchto dvoch prístupov. Obmedzenia na rozvrh niekedy do systému vkladajú aj zástupcovia jednotlivých oddelení inštitúcie, prípadne jednotliví vyučujúci a študenti. Na tvorbu rozvrhu na nové obdobie sa zvyčajne využívajú dáta z predchádzajúcich období, pokiaľ sú dostupné. Na obrázku 1 je znázornená schéma životného cyklu rozvrhu na semester.



Obrázok 1: Životný cyklus rozvrhu na semester [4]

1.2 Existujúce systémy

Na trhu je viacero softvérov na tvorbu školských rozvrhov. Líšia sa v mnohých znakoch, z ktorých najdôležitejšie sú:

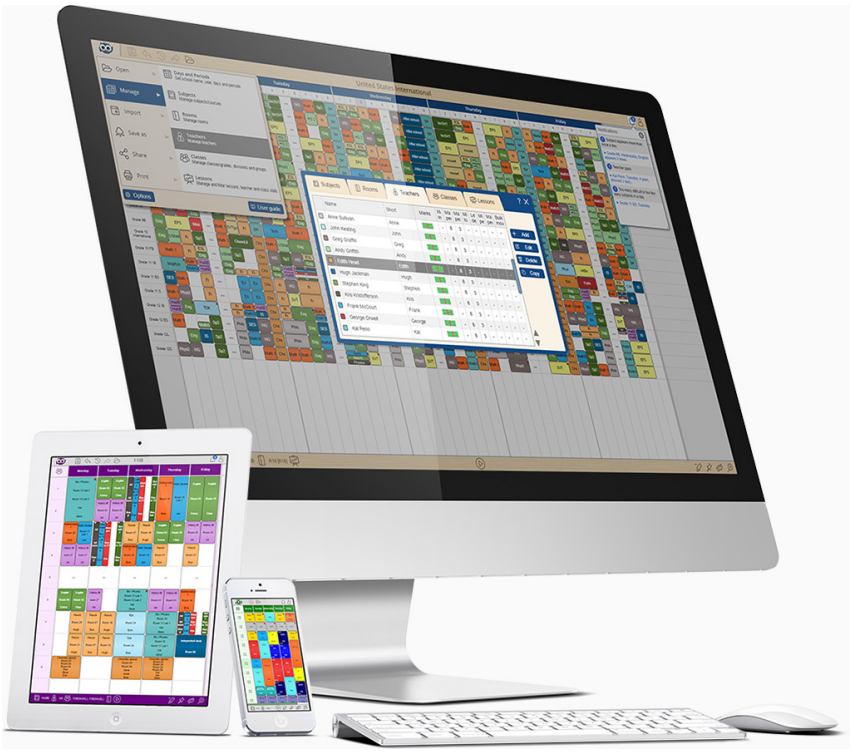
- podporovaná platforma (hardvér, OS);
- licencia (cena);
- cieľová skupina – stredné školy, univerzity, iné vzdelávacie inštitúcie atď.;
- spôsob tvorby rozvrhu – automatický, manuálny, kombinácia;
- algoritmus používaný na generovanie rozvrhu;
- typ rozvrhu – na semester, skúškové obdobie;
- druh vstupných obmedzení (požiadaviek) na rozvrh;
- import a export dát.

Autorka v práci Rozvrhový systém pre vysoké školy [2] popísala rozvrhové systémy Roger, FET a Wise Timetable. My sme sa rozhodli spomenúť niekoľko ďalších, ktoré sme zhodnotili v nasledujúcich podkapitolách.

1.2.1 Prime Timetable

Informácie v tejto kapitole sme čerpali z oficiálnej stránky Prime Timetable [5] a z vlastnej skúsenosti s jeho používaním.

Prime Timetable je multiplatformový školský rozvrhový systém na manuálnu aj automatickú tvorbu rozvrhu. Je používaný na rozvrhovanie na základných, stredných a vysokých školách a tiež aj na iných vzdelávacích inštitúciách ako aj na plánovanie rôznych udalostí (napr. školské tábory, kurzy). Softvér je implementovaný ako webová aplikácia



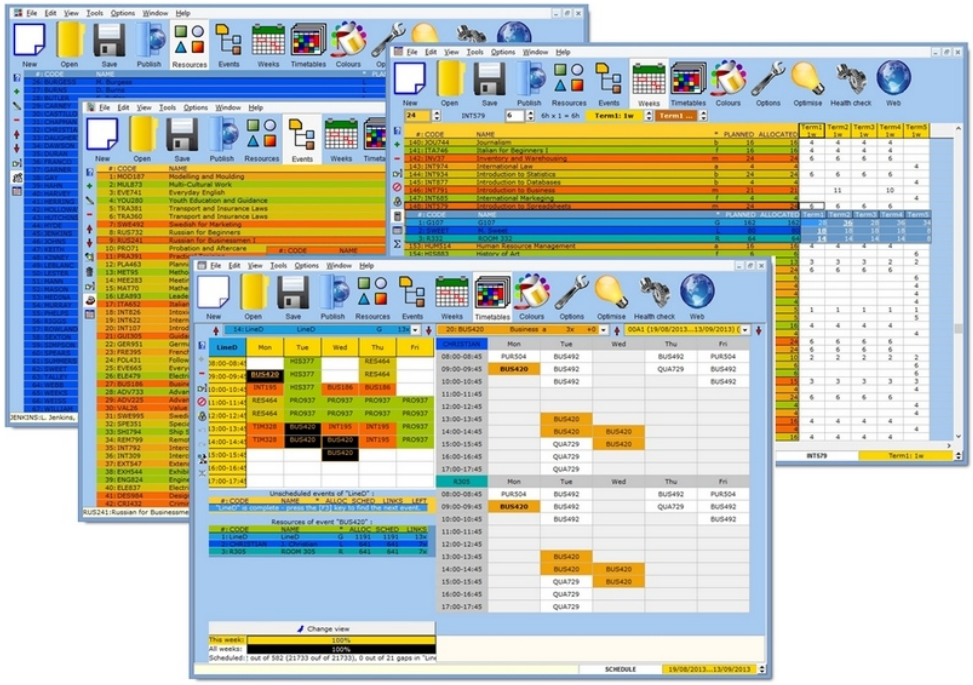
Obrázok 2: Náhľad používateľského rozhrania Prime Timetable [5]

pre moderné prehliadače, v ktorých beží klientská časť ako SPA, pričom generátor rozvrhov dosahuje najlepší výkon v prehliadači Chrome. Existuje aj zastaralá verzia aplikácie, ktorá používa Microsoft Silverlight a funguje napr. v starších verziách prehliadača Internet Explorer.

Dáta as do systému dajú vkladať ručne alebo importovať z tabuľkového editora, XML, prípadne z lokálneho súboru. K dispozícii je tiež niekoľko ukážkových súborov dát. Export je možný do tabuľkového editora, lokálneho súboru, CSV, obrázku, HTML, XML a PDF. Výsledný rozvrh sa dá aj vytlačiť, zdieľať s ostatnými používateľmi na základe rolí alebo publikovať na webe. Na úprave rozvrhu môže pracovať viacero rozvrhárov súčasne, avšak v okrajových prípadoch môže dôjsť k strate prevedených zmien. Nepodarilo sa nám zistiť aký algoritmus je použitý na generovanie rozvrhu.

Tento systém je proprietárny a jeho používanie je spoplatnené formou ročných plánov, ktorých cena sa odvíja od počtu učiteľov (používateľov). K dispozícii je tiež 30-dňová skúšobná doba, ktorá je zadarmo a aj demo verzia aplikácie, v ktorej sa neukladajú žiadne vykonané zmeny.

Medzi prednosti Prime Timetable patrí jeho príjemné, responzívne používateľské rozhranie, prístup k rozvrhom aj offline a jednoduchosť zdieľania výsledných rozvrhov. Generovanie rozvrhov na strane klienta môže byť výhodou alebo aj nevýhodou, v závislosti



Obrázok 3: Náhľad používateľského rozhrania Mimosa [7]

od výkonu zariadenia a počtu vstupných parametrov do generátora (veľkosti vyhľadávajúceho priestoru). Jasnou nevýhodou je proprietárna softvérová licencia, ktorá neumožňuje úpravu aplikácie priamo inštitúciou na základe jej špecifických požiadaviek a tiež nutnosť používať a ukladať dáta na server poskytovateľa. Po vyskúšaní demo verzie programu sme nadobudli presvedčenie, že je vhodný skôr pre menšie inštitúcie (základné a stredné školy) a len s ťažkosťami dokáže naplniť požiadavky vysokej školy.

1.2.2 Mimosa

Informácie v tejto kapitole sme čerpali z oficiálnej stránky Mimosa [6].

Mimosa je softvér na tvorbu rozvrhov a plánovanie udalostí, používaný v rôznych druhoch škôl, univerzít a firiem. Môže byť použitý v každom prostredí, kde je potrebné efektívne rozvrhovať komplexnú kombináciu obmedzených zdrojov. Podporuje tvorbu rozvrhov manuálne, automaticky alebo kombináciu obidvoch metód. Rozlišuje rozvrh na výučbové obdobie (opakovanie udalostí) a skúškové obdobie (jednorázové udalosti).

Importovať a exportovať údaje je možné cez systémovú schránku (clipboard) vo formáte tabuľkového editora alebo pomocou súborov vo formáte CSV, iCalendar, vCalendar alebo ako textové či binárne súbory Mimosa (vlastný formát). Výsledný rozvrh môže byť vytlačený alebo publikovaný ako webová stránka.

Je to proprietárna desktopová aplikácia, určená pre OS Windows. Je implementovaná

v jazyku Borland Delphi a využíva vlastnú databázu. Používateľ platí za licenciu jednorázovo a získa tým doživotné bezplatné aktualizácie programu a bezplatnú podporu cez email a telefón. Dostupná je aj freeware verzia softvéru Mimosa, ktorá obsahuje základné funkcie komerčnej verzie s obmedzeným počtom záznamov a tiež aj bezplatná 60-dňová skúšobná verzia s takmer plnou funkcionalitou.

Výhodou programu Mimosa je jeho obsiahla funkcionalita, vyspelosť a flexibilita, ktorú nadobudol aj vďaka viac ako 15-ročnému vývoju a nasedeniu v mnohých inštitúciách. Nevýhodou je jeho proprietárna licencia, naviazanosť na jednu platformu a zastaralé používateľské rozhranie. Tým, že sa jedná o desktopovú aplikáciu, umožňuje iba značne obmedzenú kolaboráciu pri úprave rozvrhu.

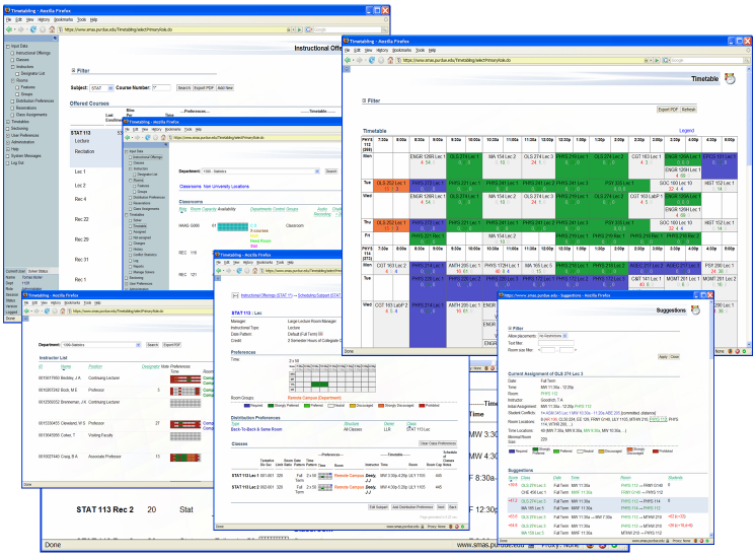
1.2.3 UniTime

Informácie v tejto kapitole sme čerpali z oficiálnej stránky UniTime [8] a z vlastnej skúsenosti s jeho používaním.

UniTime je obsiahly edukačný plánovací systém, zameraný na vysoké školy a univerzity. Podporuje tvorbu a údržbu rozvrhov na výučbové a skúškové obdobie, zdieľanie miestností na iné udalosti a rozvrhovanie študentov na jednotlivé predmety. Umožňuje viacerým univerzitným a úsekovým rozvrhárom koordinovať úsilie pri zostavovaní a úprave rozvrhu. Môže byť používaný samostatne alebo prepojený s existujúcim študentským informačným systémom. Dáta môžu byť do systému importované alebo z neho exportované vo formáte XML, pričom ich formát reflektuje interný dátový model UniTime.

Na generovanie rozvrhov využíva knižnicu CPSolver, ktorá slúži na riešenie problémov s obmedzeniami. Táto knižnica obsahuje framework založený na lokálnom vyhľadávaní, ktorý umožňuje modelovať problém s použitím primitív programovania s obmedzujúcimi podmienkami (constraint programming), teda premenných, hodnôt a obmedzení. Vyhľadávanie je založené na iteratívnom doprednom vyhľadávacom algoritme. Tento algoritmus je podobný lokálnym vyhľadávacím metódam, ale v kontraste s klasickými technikami lokálneho vyhľadávania operuje nad prípustnými, aj keď nie nevyhnutne kompletnými, riešeniami. Avšak všetky tvrdé obmedzenia musia byť splnené. Vďaka iteratívnemu charakteru algoritmu je možné generátor jednoducho spustiť, zastaviť alebo pokračovať v riešení od ľubovoľného prípustného riešenia, či už kompletného alebo nie.

Systém bol pôvodne vyvinutý ako spoločné úsilie členov fakúlt, študentov a zamestnancov univerzít v Severnej Amerike a Európe. Používajú ho viaceré inštitúcie v rôznych častiach sveta. Softvér je distribuovaný zdarma pod open source licenciou a v roku 2015 sa stal sponzorovaným projektom Apereo Foundation. Je to platformovo nezávislá webová aplikácia, implementovaná s použitím Java J2EE a SQL databázy. Jej demo verzia



Obrázok 4: Náhľad používateľského rozhrania UniTime [8]

je voľne dostupná na jej oficiálnej stránke, rovnako aj niekoľko súborov ukážkových dát. UniTime má rozsiahlu dokumentáciu a poskytuje tiež (obmedzenú) bezplatnú aj komerčnú podporu.

Medzi jeho výhody patrí robustnosť, pokrytie veľkého množstva potrieb pri rozvrhovaní a vyspelosť algoritmu na generovanie rozvrhov. Nevýhodou je používateľsky neveľmi prívetivé, neresponzívne, pomalé grafické rozhranie a absencia webového API.

2 Cieľ práce

Cieľom našej práce bolo oboznámiť sa s prácou Rozvrhový systém pre vysoké školy (Emília Knapereková, 2014) [2], vrámci ktorej vznikol prototyp rozvrhového systému, aktualizovať ho a pokračovať v jeho vývoji. Našim plánom tiež bolo zoznámiť sa s požiadavkami, ktoré sú kladené na tvorbu rozvrhu na FEI a zabezpečiť, aby ich výsledný systém spĺňal.

Zvolili sme agilný, iteratívny prístup k vývoju systému, s krátkymi cyklami spätnej väzby a so zámerom zaistiť jeho flexibilný a efektívny vývoj a adekvátnu kvalitu. Výsledný systém by mal slúžiť v prvom rade ako nástroj pre rozvrhára na manuálnu tvorbu a úpravu rozvrhu, teda poskytovať minimálne nasledujúce funkcie:

- import a export dát z, resp. do AIS,
- tvorba rozvrhu na semester a skúškové obdobie,
- detekciu kolízií rozvrhových akcií.

2.1 Špecifikácia požiadaviek

Práca Rozvrhový systém pre vysoké školy [2] obsahuje analýzu a špecifikáciu požiadaviek na rozvrhový systém. Pri jej tvorbe autorka vychádzala z poznatkov nadobudnutých počas tímového projektu, analýzy existujúcich riešení a konzultácií s rozvrhármi FEI STU. Táto špecifikácia zostáva naďalej platná a vychádzali sme z nej pri návrhu zmien, resp. ďalšej implementácie systému. Keďže cieľom našej práce bol hlavne ďalší vývoj aplikácie, v tejto podkapitole sme ju preto iba zosumarizovali vo forme vysoko-úrovňového pohľadu na funkcionalitu systému s cieľom vytvoriť kontext pre čitateľa. Systém má umožňovať:

- Tvorbu rozvrhu na semester a skúškové obdobie.
- Import a export dát z, resp. do AIS čoho súčasťou je aj ich korekcia pred ich ďalším spracovaním.
- Konfiguráciu nastavení systému ako sú napr. parametre generátora rozvrhov, počet týždňov jednotlivých období, farby pre zobrazovanie rozvrhu atď.
- Vkladanie požiadaviek na rozvrh, vrátane osobných požiadaviek študentov a vyučujúcich.
- Generovanie rozvrhu.

- Manuálnu úpravu rozvrhu pomocou príjemného drag-and-drop (ťahaj a pusť) používateľského rozhrania.
- Kontrolu a zobrazovanie kolízií v rozvrhu.
- Rezerváciu miestností na výnimočnú udalosť.
- Navrhovanie zmien rozvrhu ostatnými používateľmi pre posúdenie rozvrhárom.
- Správu oprávnení používateľov.
- Prezeranie výsledného rozvrhu.
- Zálohovanie dát a verziovanie jednotlivých rozvrhov.

2.2 Tvorba rozvrhov na FEI

Proces tvorby rozvrhu na Fakulte elektrotechniky a informatiky STU sa v čase písania tejto práce významne neodlišoval od toho ako bol popísaný v práci mojej predchodkyne [2], čo sme si overili na osobnom stretnutí s rozvrhárom Mgr. Dávidom Panczom, PhD. Na FEI sa rozvrhy na semester a skúškové obdobie tvoria oddelene. Každé z období má na starosti iný rozvrhár a každý z nich používa na ich tvorbu iný systém.

Na tvorbu rozvrhov na semester sa používa starý program WinRozvrhy, ktorý vznikol dávnejšie priamo pre potreby FEI, aj napriek tomu že rozvrhár má k dispozícií novší program Roger, keďže rozvrh ním generovaný je nepoužiteľný a možnosti jeho manuálnej úpravy sú nedostatočné. Rozvrh na skúškové obdobie sa tvorí s pomocou MS Excel a súboru skriptov na import dát a čiastočnú kontrolu kolízií.

Obom rozvrhárom je pred samotnou tvorbou rozvrhov doručený aktuálny súbor dát z AIS, ktorý obsahuje zoznamy miestností, učiteľov, predmetov atď. Zároveň im pred, ako aj počas obdobia tvorby rozvrhov prichádzajú požiadavky na rozvrh od vyučujúcich, ktoré musia pri jeho tvorbe zohľadňovať. Niektoré z nich je možné zadať do rozvrhového systému (napr. hodinové dotácie predmetov), ale veľkú časť z nich musí rozvrhár iba zobrať do úvahy počas manuálnej úpravy rozvrhu (napr. preferencie miestností alebo času). Pri tvorbe rozvrhov sa vychádza z predošlých rozvrhov na jednotlivé obdobia. Výsledný rozvrh na semester sa na záver importuje do AIS, kde si ho potom môžu používatelia prezerať a v prípade študentov sa aj prihlasovať na jednotlivé termíny cvičení. Rozvrh na semester aj skúškové obdobie sa tiež zverejní na stránke fakulty.

Takýto proces tvorby rozvrhov na FEI je v oboch prípadoch zdĺhavý a neefektívny a potreba nového rozvrhového systému, ktorý by ho zlepšil je stále aktuálna.

3 Návrh

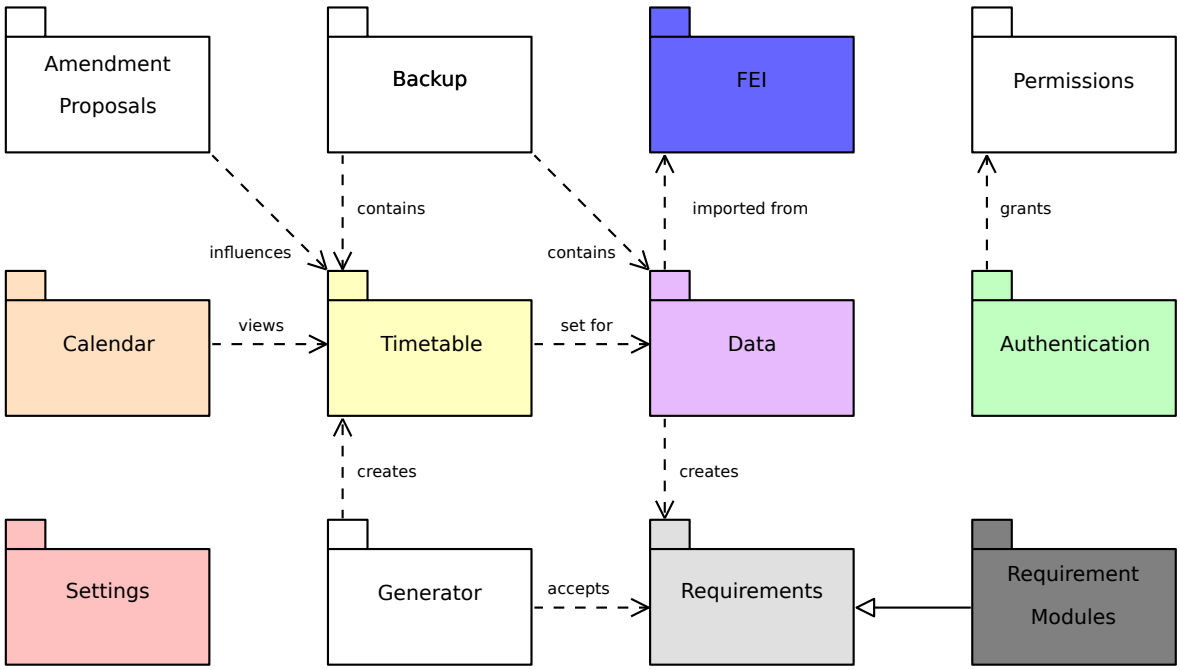
V tejto kapitole popisujeme návrh ďalšieho vývoja rozvrhového systému. V krátkosti približujeme pôvodný návrh rozvrhového systému našej predchodkyne, ktorý sme porovnali s našim návrhom, pričom vysvetľujeme motiváciu jednotlivých zmien a zvolených prístupov.

3.1 Architektúra systému

Základným predpokladom je, že by systém mal byť platformovo nezávislý a umožňovať jednoduché zdieľanie rozvrhov. Logickou voľbou preto bolo implementovať ho formou webovej aplikácie.

3.1.1 Pôvodná architektúra

Na obrázku 5 môžme vidieť zjednodušený model pôvodnej architektúry systému reprezentovaný ako UML diagram balíčkov. Balíčky, ktorých funkcionalitu moja predchodkyňa aspoň čiastočne implementovala, sú farebne zvýraznené. Nasleduje krátky popis jednotlivých balíčkov.



Obrázok 5: Pôvodná architektúra – diagram balíčkov (zoskupení tried) [2]

Backup Zálohovanie jednotlivých verzií rozvrhu. K takto vytvorenej zálohe sa bude môcť rozvrhár neskôr vrátiť.

Calendar Zobrazovanie rozvrhových akcií v tabuľke. Podporuje rôzne typy časových mriežok, medzi ktorými je možné prepínať pri zobrazení rozvrhu na týždeň alebo deň. Ponúka tiež mesačný prehľad rozvrhových akcií (bez mriežky).

FEI Import a export dát z, resp. do AIS. Pre každú inštitúciu môže existovať podobný balíček, v závislosti od požadovaného formátu vstupných alebo výstupných dát.

Authentication Overenie totožnosti používateľa, na základe ktorej mu systém pridelí práva na prístup k vybraným funkciám systému.

Permissions Nastavenie oprávnení používateľov.

AmendmentProposals Navrhovanie zmien v umiestnení rozvrhových akcií.

Timetable Vytváranie rozvrhových akcií pre rozvrh na semester aj skúškový rozvrh.

Data Úprava dát, nevyhnutných pre tvorbu rozvrhu (zoznamy predmetov, študentov, miestností, atď.). Import týchto údajov môže zabezpečiť osobitný balíček (v našom prípade balíček *FEI*).

Settings Správa nastavení celého rozvrhového systému. Tento balíček je využívaný všetkými ostatnými balíčkamikmi.

Generator Generovanie rozvrhu na základe vstupných požiadaviek (z balíčka *Requirements*).

Requirements Správa požiadaviek vyučujúcich, študentov, miestností. Bude obsahovať rozhranie pre pripojenie modulov z balíčka *RequirementModules*.

RequirementModules Implementácia jednotlivých typov požiadaviek na rozvrh. Každý typ požiadavky bude musieť dodržať rozhranie na pripojenie k modulu *Requirements*.

Tieto balíčky korešpondujú s časťami webovej aplikácie, ktorá na serveri ukladá dáta do databázy a poskytuje ich cez protokol HTTP formou HTML dokumentu tenkému klientovi, ktorý beží vo webovom prehliadači. Dáta sa upravujú prevažne cez formuláre, po odoslaní ktorých server ako odpoveď odošle opäť celý dokument, s menšími výnimkami, keď sa na komunikáciu používa AJAX, ktorý umožňuje so serverom komunikovať asynchrónne. Hlavná interakcia prebieha v balíčku *Calendar* a *Timetable*, ktoré slúžia na manipuláciu rozvrhu, pričom balíček *Calendar* obsahuje aj drag-and-drop funkcionalitu, implementovanú v JS.

3.1.2 Navrhnuté zmeny

Architektúra pôvodného systému je vhodná skôr pre menej interaktívne aplikácie, v ktorých si klient vystačí s formulármi a minimálnou manipuláciou s Document Object Model (DOM). Pokiaľ je potrebné, aby aplikácia poskytovala bohaté používateľské rozhranie, vrátane drag-and-drop s možnosťou komplexnej manipulácie dát, je vhodnejšie presunúť viac kompetencií na stranu klienta. To umožní predovšetkým lepšie riadiť zložitosť systému.

Rozhodli sme sa preto zo systému klientskú časť osamostatniť do podoby Single-page Application (SPA), teda jednostránkovej aplikácie, ktorá bude so serverom komunikovať pomocou webového API, z ktorého sa tým pádom stane webová služba. To zo sebou prináša mnohé výhody, z ktorých niektoré sú:

- posielanie iba nevyhnutných dát znižuje odozvu klienta,
- uľahčenie manipulácie dát, keďže ich klient nemusí získavať z DOM,
- lepšie oddelenie zodpovedností klienta a servera zvyšuje možnosti ich oddeleného vývoja a sprehľadňuje zdrojový kód,
- možnosť interoperability s inými službami, napr. automatizáciu importu a exportu dát z, resp. do AIS, prístup k dátam o aktuálne voľných miestnostiach pre aplikácie tretích strán atď.,
- možnosť poskytnúť offline funkcionalitu, napr. prehliadanie poslednej načítanej verzie rozvrhov,
- možnosť implementácie viacerých klientov, napr. v podobe natívnych mobilných aplikácií.

Nevýhodou takejto zmeny je, že sa tým zvýši náročnosť a zložitosť implementácie. Avšak tento nárast je v porovnaní s rýchlo rastúcou komplexnosťou systému, spôsobenou pridávaním novej funkcionality, konštantný, keďže sa po prvotnej úprave vývojového prostredia, použitých technológií a refaktoringu zdrojového kódu zastaví.

Kvôli zmene tohto typu v podstate nie je potrebné meniť dátový model na strane servera, ktorý je bližšie popísaný v práci mojej predchodkyne [2]. Z pohľadu z diaľky by sa tak len zmenil formát a štruktúra URL zdrojov, ktoré server poskytuje. Avšak rozhodli sme sa ho upraviť z iných dôvodov. Hlavné zmeny sú nasledovné:

- Odstránenie balíčka *Calendar*, ktorý slúžil na prezentáciu a úpravu rozvrhu. Po zmene architektúry preberie na seba túto úlohu klient.
- Premenovanie balíčka *Data* na *School*, čo lepšie popisuje jeho zameranie.
- Nastavenia rozvrhov (trvanie periód, počet týždňov atď.) sa budú ukladať pre každý rozvrh osobitne do databázy, namiesto pre celý systém všeobecne v konfiguračnom súbore. Zároveň sa tým stratí potreba explicitne rozdeľovať rozvrhy na semester a skúškové obdobie do osobitných tried. Umožníme tým prácu s rozvrhmi, ktoré majú rôzne parametre (napr. dĺžku vyučovacej hodiny).
- S tým tiež súvisí zmena reprezentácie času konania udalosti v rozvrhu. Neukladáme jej absolútny čas a dátum, ale iba relatívny. Rozvrh bude mať nastavený počet týždňov obdobia, trvanie jednej časovej periódy (vyučovacej hodiny) atď. Udalosť v rozvrhu tak bude obsahovať zoznam týždňov, v ktoré sa uskutoční, konkrétny deň v týždni a interval časových periód, ktoré sú pre ňu vyhradené. Napr. cvičenie z matematiky sa uskutoční prvých desať týždňov semestra, v pondelok, počas tretej až štvrtej periódy. Týmto sa zjednoduší reprezentácia udalostí a uľahčí výpočet ich kolízií.

3.2 Webová služba

Webová služba je softvérový systém, navrhnutý na podporu interoperabilných interakcií medzi počítačmi cez sieť. [9] Vo webovej službe sa webové technológie ako napr. HTTP, pôvodne navrhnuté na komunikáciu medzi človekom a počítačom, používajú na komunikáciu medzi zariadeniami, resp. programami. Môžeme identifikovať dve hlavné triedy webových služieb:

- REST webové služby, ktorých primárnym cieľom je manipulácia reprezentácií webových zdrojov s použitím uniformnej množiny bezstavových operácií,
- svojvoľné webové služby, ktoré môžu exponovať ľubovoľnú množinu operácií. [10]

Rozhodli sme sa, že serverová časť rozvrhového systému bude REST webová služba, pričom komunikácia bude prebiehať cez HTTP. Jej bezstavovosť zlepšuje prehľadnosť, objaviteľnosť a škálovateľnosť webového API a znižuje previazanosť medzi serverom a klientom. REST architektúra je v súčastnosti pomerne rozšírená, čoho dôsledkom je dostupnosť viacerých knižníc a frameworkov pre jej podporu v rôznych programovacích jazykoch, čo uľahčuje jej implementáciu.

Serverová časť aplikácie tak bude mať na starosti všetko to čo predtým, okrem používateľského rozhrania. To znamená, že dáta:

- sa ukladajú do databázy,
- interne sú reprezentované formou entitných tried,
- externe sú reprezentované formou webových zdrojov,
- je možné s nimi manipulovať cez webové API,
- je možné ich importovať a exportovať z, resp. do preddefinovaných formátov,
- je možné ich zálohovať,
- prístup k nim je autentifikovaný a autorizovaný.

Z biznis logiky ma na starosti:

- generovanie rozvrhov,
- výpočet kolízií rozvrhových akcií.

3.2.1 REST

Representational State Transfer (REST) alebo RESTful webové služby sú jedným so spôsobov ako zabezpečiť interoperabilitu medzi počítačovými systémami na internete. Webové služby, ktoré dodržujú tento architektonický štýl, dávajú dotazujúcim systémom možnosť pristupovať a manipulovať s textovými reprezentáciami webových zdrojov s použitím preddefinovanej množiny bezstavových operácií. Webové zdroje boli pôvodne definované ako dokumenty alebo súbory identifikované pomocou ich URL, ale dnes ich definícia zahŕňa každú entitu, ktorú je možné identifikovať, pomenovať, adresovať alebo s ňou akýmkoľvek spôsobom narábať na webe.

Dotazy na URI zdrojov v rámci REST webovej služby vyvolajú odpoveď, ktorá býva najčastejšie vo formáte JSON, XML alebo HTML. Odpoveď môže potvrdiť, že došlo k nejakej modifikácii uloženého zdroja a môže poskytnúť hypertextové odkazy na iné súvisiace zdroje alebo kolekcie zdrojov. Operácie, ktoré sú k dispozícii pri komunikácií cez HTTP, ktorý býva použitý najčastejšie, zahŕňajú tie, ktoré sú preddefinované HTTP slovesami (metódami), teda napr. `GET`, `POST`, `PUT`, `DELETE` atď. Tým, že REST systémy používajú bezstavový protokol, štandardné operácie a opätovne používajú komponenty, ktoré môžu byť spravované a aktualizované bez toho, aby to ovplyvnilo celý systém, aj keď práve beží, sa snažia dosiahnuť vysoký výkon, spoľahlivosť a schopnosť rásť. [11]

Na reprezentáciu zdrojov sme si vybrali formát JavaScript Object Notation (JSON). Je to ľahký, textový, jazykovo nezávislý formát výmeny údajov. Je odvodený z jazyka JavaScript (JS), ale v súčastnosti veľa programovacích jazykov podporuje jeho parsovanie a generovanie a zároveň je ľahko čitateľný aj pre ľudí. Zakladá sa na pároch atribútov a hodnôt. [12]

3.2.2 Autentifikácia

V kontexte webovej služby je autentifikácia mechanizmus asociácie prichádzajúceho dotazu s množinou identifikačných údajov, čo môže byť napr. používateľ, od ktorého dotaz prišiel alebo token, ktorým bol podpísaný. [13] Uvádzame niekoľko bežných schém, ktoré je možné použiť na implementáciu autentifikácie pre REST webovú službu:

HTTP basic authentication V preklade jednoduché overenie prístupu, je metóda, v ktorej klient odošle používateľské meno a heslo ako súčasť dotazu. Tieto údaje sú zakódované v kódovaní Base64 a nie sú pri prenose nijako chránené. [14] Keďže toto pole musí byť odoslané v hlavičke každého dotazu, ktorý má byť autentifikovaný, klient (webový prehliadač) by mal prístupové údaje po určitú dobu udržiavať vo svojej vyrovnávacej pamäti, ak sa chce vyhnúť tomu, aby ich používateľ musel zakaždým zadávať. HTTP neposkytuje webovému serveru spôsob ako klienta inštruovať, aby používateľa „odhlásil“, teda uvoľnil prihlasovacie údaje z vyrovnávacej pamäte. Existuje niekoľko možností ako v niektorých prehliadačoch tieto údaje vymazať, avšak toto správanie je nekonzistentné medzi jednotlivými prehliadačmi a ich verziami. Príklad výsledného poľa v hlavičke dotazu:

```
Authorization: Basic dGVzdDoxMjPCow==
```

Session manažment Keďže HTTP protokol je z definície bezstavový, tak na to, aby bolo možné asociovať dotaz s akýmkoľvek iným dotazom, je potrebné si medzi jednotlivými dotazmi zapamätať nejaké údaje. Jedno z možných riešení je vytvoriť na strane servera tzv. session (reláciu), do ktorej si môže server ukladať ľubovoľné informácie o klientovi. Na strane klienta je potrebné uložiť si iba session ID, ktoré sa pri každom dotaze odošle na server najčastejšie vo forme HTTP cookie, prípadne ako URL parameter. V prípade, že sa klient úspešne autentifikuje sa táto informácia pridá do jeho session, čiže zostáva autentifikovaný po dobu platnosti session, resp. pokiaľ neurobí dotaz na odhlásenie. Takýto prístup je rozšírený medzi klasickými webovými aplikáciami.

Autentifikácia na základe tokenu Token je reťazec podpísaný serverom, ktorý klient

získa ako odpoveď na požiadavku o jeho vytvorenie, v ktorej sa preukáže svojimi prihlasovacími údajmi. Následne ho pridá do hlavičky každého dotazu, ktorý má byť autentifikovaný. Na rozdiel od session manažmentu si server nemusí nič pamätať, všetky potrebné údaje sú zakódované v tokene. Jeho dôveryhodnosť a integrita by mala byť zabezpečená kombináciou správne zvolených kryptografických primitív. Token máva spravidla časovo obmedzenú platnosť. Je na klientovi ako si token uloží. Príklad výsledného poľa v hlavičke dotazu:

```
Authorization: Bearer 9944b09199c62bcf9418ad846dd0e4bbdfc6ee4b
```

Autentifikácia na základe dotazu Požiadavky je možné autentifikovať tým, že sa identifikačné údaje zaradia do parametrov URL reťazca, namiesto využitia HTTP hlavičky. Je potrebné, aby klient a server zdieľali spoločné tajomstvo, tzv. API kľúč, a zároveň jeho ID. Klient pri konštrukcii dotazu pridá medzi parametre ID kľúča, spravidla aj časovú pečiatku (kvôli časovému obmedzeniu platnosti dotazu), následne dotaz s použitím kľúča podpíše a výsledný podpis tiež pridá medzi parametre dotazu. Výhodou tohto prístupu je, že umožňuje priamy prístup tretích strán k privátnym dátam, bez nutnosti ich k tomu „splnomocniť“, napr. pomocou proxy servera. Výsledný dotaz môže vyzerať napr. takto:

```
GET /object?timestamp=1490099398&apiKey=7d129f12fa322280&signature=
f213669fb72a47ae622a9b71f39a6c6d
```

Na zabezpečenie dôveryhodnosti sa zvyčajne každá z týchto schéme používa v kombinácii s HTTPS.

Autentifikáciu REST API sme sa rozhodli riešiť pomocou tokenu, pretože zachováva bezstavovosť servera, je vhodná na použitie aj v klientoch, implementovaných ako natívne aplikácie (teda nielen v prehliadači) a zároveň existuje niekoľko štandardov, ktoré ju popisujú a pre ktoré sú k dispozícii voľne dostupné implementácie. Konkrétne sme si vybrali štandard JSON Web Token (JWT).

3.2.3 JSON Web Token

JSON Web Token je otvorený štandard, ktorý definuje kompaktný a sebestačný spôsob bezpečného prenosu informácií medzi zúčastnenými stranami ako JSON objekt, ktorý obsahuje isté tvrdenia. Tokeny môžu byť podpísané s použitím tajomstva (s HMAC algoritmom) alebo privátneho kľúča (s RSA algoritmom). Napríklad, server môže vygenerovať token, ktorý obsahuje tvrdenie „prihlásený ako admin“ a poskytnúť ho klientovi. Klient následne môže tento token použiť na dokázanie, že je prihlásený ako admin. JWT je reprezentovaný ako sekvencia Base64URL zakódovaných častí oddelených bodkou, typicky:

Hlavička Obsahuje typ tokenu, čo je JWT a hašovací algoritmus, ktorý sa používa, napr. HMAC SHA256. Príklad hlavičky:

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Užitočné dáta Obsahuje tvrdenia, čo sú výroky o entite (typicky používateľovi) a ďalšie metadáta. Existujú tri typy tvrdení:

Rezervované tvrdenia Množina preddefinovaných tvrdení, ktoré nie sú povinné, ale odporúčané. Niektoré z nich sú: **iss** (vydavateľ), **exp** (čas exspirácie), **sub** (predmet), **aud** (publikum) a iné.

Verejné tvrdenia Tieto môžu byť definované ľubovoľne používateľmi JWT. Kvôli vyhnutiu sa kolízií by mali byť definované v IANA JSON Web Token registri alebo definované ako URI, ktoré obsahuje kolízne rezistentný menný priestor.

Privátne tvrdenia Toto sú vlastné tvrdenia, vytvorené na zdieľanie informácií medzi stranami, ktoré sa dohodli na ich použití.

Príklad užitočných dát:

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

Podpis Na vytvorenie podpisu je potrebné aplikovať algoritmus deklarovaný v hlavičke na zreťazenie hlavičky a užitočných dát. Podpis sa používa na overenie toho, že odosielateľ JWT je ten, za koho sa vydáva a na zaručenie, že správa nebola počas prenosu upravená. [15, 16] Príklad konštrukcie podpisu:

```
HMACSHA256(
  base64UrlEncode(hlavička) + "." + base64UrlEncode(užitočné dáta),
  tajomstvo)
```

Príklad výsledného tokenu (zalomenie riadkov len pre účely zobrazenia):

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.
TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ
```

3.2.4 Návrh API

V tejto časti popisujeme dizajn API, ktoré bude webová služba poskytovať. Jedným z kľúčových princípov REST je rozdelenie API na logické zdroje. Sú manipulované cez HTTP dotazy, kde každá z možných metód má špecifický význam. Jednotlivé zdroje sú v URI reprezentované podstatnými menami, ktoré dávajú zmysel z pohľadu konzumenta API. Výstup, teda textová reprezentácia zdrojov, je vo formáte JSON. Rovnako aj vstupné dáta, ktoré sú súčasťou tela dotazu. API tiež vracia zmysluplné HTTP status kódy, napr. `201 Created` ako odpoveď na `POST` dotaz, ktorého výsledkom je vytvorenie zdroja alebo `204 No Content` ako odpoveď na úspešný dotaz, ktorej telo je prázdne (napr. `DELETE` dotaz). Zdroje môžu, ale nemusia, korešpondovať s entitami interného dátového modelu servera.

Nasleduje popis významu akcií, ktoré je možné použiť na manipuláciu zdrojov, avšak jednotlivé zdroje ich nemusia podporovať všetky. Ich použitie sme ilustrovali na príklade jedného zdroja:

- `GET /rooms/` - načíta zoznam miestností,
- `GET /rooms/42/` - načíta konkrétnu miestnosť,
- `POST /rooms/` - vytvorí novú miestnosť,
- `PUT /rooms/42/` - aktualizuje miestnosť s ID 42,
- `PATCH /rooms/42/` - parciálne aktualizuje miestnosť s ID 42,
- `DELETE /rooms/42/` - vymaže miestnosť s ID 42.

Medzi metódami `PUT` a `PATCH` existuje významná odlišnosť v tom, že `PUT` je na rozdiel od `PATCH` idempotentná. To znamená, že ak metódu `PUT` aplikujeme viac krát s tými istými dátami na rovnaký zdroj, výsledkom bude stále ten istý zdroj, čo pri `PATCH` platiť nemusí. Metódy, ktoré zdroje vytvárajú alebo aktualizujú (`POST`, `PUT` a `PATCH`), vracajú reprezentáciu zdroja, ktorý vytvoria alebo upravia, aby klient nemusel vytvárať nový dotaz, kvôli získaniu aktuálnej reprezentácie zdroja.

Ak môže byť zdroj v relácii s viacerými zdrojmi, ID zdroja s ktorým je v relácii je obsiahnuté v jeho reprezentácii. Fakt, že zdroj môže byť v relácii iba s jedným iným zdrojom je vyjadrený tým, že jeho URI sa skladá z URI nadradeného zdroja, za ktorou nasledujú segmenty prislúchajúce jemu ako podradenému zdroju. Napr. udalosť musí prislúchať nejakému konkrétnemu rozvrhu. Jej cesta môže vyzerať nasledovne: `/timetables/7/events/14/` - udalosť s ID 14 prislúchajúca k rozvrhu s ID 7.

Zdroje, pri ktorých to má zmysel, musí byť možné filtrovať na základe vybraných parametrov. Tieto parametre budú súčasťou URL dotazu na zdroj. Napr. dotaz `GET /rooms/?min_capacity=100` vyfiltruje miestnosti, ktoré majú kapacitu minimálne 100 ľudí. Koncové body, ktoré vracajú zoznam zdrojov, tak budú robiť po stranách, napr. `GET /rooms/?page=2` vráti druhú stranu. Súčasťou odpovede bude celkový počet zdrojov v zozname a počet zdrojov na stranu bude konštantný. Pokiaľ parameter `page` nebude súčasťou dotazu, vráti sa prvá strana zoznamu. Takisto musí byť možné v zdrojoch vyhľadávať na základe ich vybraných atribútov, ktoré sú reťazce. Napr. dotaz `GET /rooms/?search=ab` vyfiltruje miestnosti, ktorých názov obsahuje podreťazec *ab*. Zdroje tiež bude možné zoradiť, napr. dotaz `GET /rooms/?ordering=name` zoradí miestnosti podľa názvu. Filtrovanie pomocou viacerých parametrov bude možné kombinovať. Napr. dotaz `GET /rooms/?page=2&min_capacity=100&ordering=name&search=ab` vyfiltruje miestnosti podľa kritérií, ktoré sú zjednotením predošlých príkladov.

Autentifikácia API je bezstavová. Používame JWT, ktorý sa prenáša v hlavičke dotazu. Zdroje, ktoré neobsahujú citlivé údaje, sú na čítanie prístupné aj bez autentifikácie. Na zaistenie bezpečnosti API využívame protokol TLS. Nasleduje krátky popis jednotlivých zdrojov:

activities Kombinácia kategórie aktivity a voliteľne predmetu (napr. mimoškolské aktivity nemusia súvisieť s predmetmi).

activity-categories Kategórie aktivít, napr. prednáška, cvičenie, seminár atď.

api-token-auth Slúži na vytvorenie autentifikačného tokenu. Podporuje iba metódu `POST`, ktorej vstupom je používateľské meno a heslo a výstupom token.

api-token-refresh Slúži na aktualizáciu autentifikačného tokenu. Podporuje iba metódu `POST`, ktorej vstupom je aktuálny token a výstupom obnovený token (s novým časom exspirácie).

courses Vyučujúce predmety, obsahujú ID ich príslušného oddelenia.

departments Oddelenia školy členené hierarchicky. Obsahujú ID rodiča, pokiaľ nie sú koreňom stromu. Napr. FEI je rodičom ÚIM a pod.

equipment Vybavenie, napr. tabuľa, projektor atď.

groups Skupiny zamestnancov a študentov členené hierarchicky. Obsahujú ID rodiča, pokiaľ nie sú koreňom stromu. Napr. Aplikovaná informatika je rodičom skupiny Bezpečnosť informačných systémov.

room-categories Kategórie miestností, napr. poslucháreň, posilovňa, laboratórium atď.

rooms Miestnosti v škole, obsahujú kategóriu, vybavenie a ID ich príslušného oddelenia.

timetables Rozvrhy na jednotlivé obdobia a ich metadáta.

events Udalosti v rozvrhu, sú kombináciou aktivity, miestnosti, času konania a voliteľne skupín. Každá udalosť patrí k práve jednému rozvrhu, preto k nim nie je možné pristupovať osobitne, iba cez URL rozvrhu, napr. `/timetables/7/events/`.

3.3 Webový klient

Webový klient je aplikácia, ktorá beží v internetovom prehliadači. S REST API, ktoré poskytuje server, komunikuje prostredníctvom HTTP (resp. HTTPS) vo formáte JSON. Obsahuje používateľské rozhranie (UI), ktoré slúži na interakciu s rozvrhovým systémom, pričom používateľom poskytuje funkcionalitu definovanú v špecifikácii požiadaviek v kapitole 2.1 a podrobnejšie v práci mojej predchodkyne [2].

3.3.1 Nefunkcionálne požiadavky

Klient by mal spĺňať aj tieto nefunkcionálne požiadavky:

- podporovať minimálne posledné dve verzie najrozšírenejších moderných webových prehliadačov,
- byť responzívny, čiže prispôsobovať svoje UI veľkosti zariadenia (resp. zobrazenia),
- mať príjemné a efektívne UI,
- v prípade, že používateľ nie je autentifikovaný, poskytnúť funkcionalitu, na ktorú autentifikácia nie je potrebná,
- v prípade, že sa mu nepodarí vytvoriť spojenie so serverom, poskytnúť funkcionalitu, na ktorú spojenie so serverom nie je potrebné,

- byť výkonný a minimalizovať množstvo prenášaných dát,
- byť bezpečný, teda obsahovať ochranu proti bežným hrozbám ako napr. Cross-site scripting (XSS),
- byť prístupný pre používateľov s postihnutím.

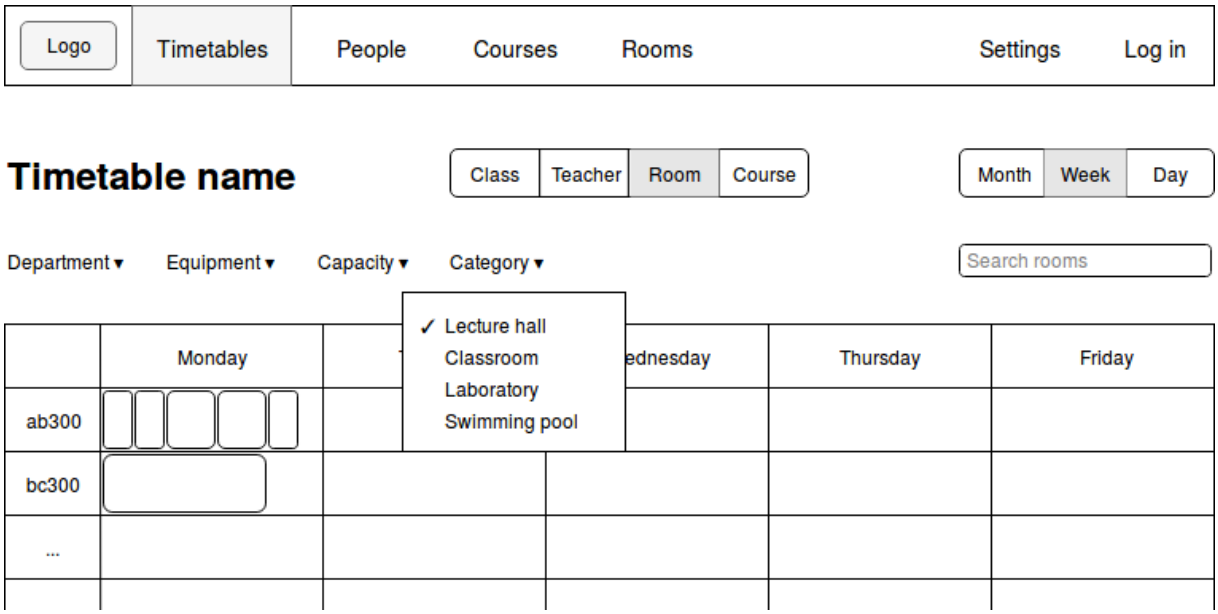
3.3.2 Používateľské rozhranie

Klient podporuje editovanie údajov o zdrojoch, alokovaných v rámci tvorby rozvrhu (predmety, miestnosti, študenti atď.), ako aj obmedzenia a požiadavky na rozvrh a parametre generátora pomocou prehľadných formulárov. Avšak najdôležitejšou časťou UI je editácia jednotlivých rozvrhových akcií (udalostí) v rozvrhu. Klient musí pre zvolený rozvrh podporovať jeho zobrazenie v rôznych časových mriežkach (dennej, týždennej a pod.) a filtrovanie podľa rôznych zdrojov (miestností, predmetov, vyučujúcich a pod.), pričom jeho editácia prebieha pomocou drag-and-drop presúvania, resp. vytvárania rozvrhových akcií. Pri tejto časti UI nám bol inšpiráciou Kalendár Google[1], FullCalendar[2] a Prime Timetable (podkapitola 1.2.1). Tiež by mal prehľadne a v reálnom čase zobrazovať ich prípadné kolízie. Klient tiež musí poskytovať manažment osobitných rozvrhov a ich nastavení (počet týždňov, dni výuky, dĺžky časových období v dni atď.).

Podrobnejšie si popíšeme rozhranie, ktorá slúži na prezeranie a editovanie rozvrhov. Jeho návrh môžme vidieť na obrázku 6. V hornej časti sa pod hlavným menu nachádza názov rozvrhu a vedľa neho lišty na filtrovanie zobrazených udalostí podľa typu zdroja a časovej granularity. Pod nimi sa nachádza lišta na filtrovanie zobrazených zdrojov (riadkov v mriežke) podľa ich špecifických parametrov. V prípade miestností ide napr. o ich kategóriu či vybavenie. Napravo od nej je vstupné pole pre vyhľadávací reťazec, ktorý slúži na filtrovanie zobrazených zdrojov na základe ich textových parametrov, v prípade miestností napr. podľa ich názvu alebo kategórie. Tieto spôsoby filtrovania je možné kombinovať, napr. vyhľadať miestnosti, ktorých názov obsahuje podreťazec ab s minimálnou kapacitou 100 ľudí a súčasťou ich vybavenia je projektor.

Nasleduje samotný rozvrh, prezentovaný ako tabuľka, ktorej stĺpce (okrem prvého) sú časové jednotky a riadky tvoria zdroje, spĺňajúce filtrovacie kritériá, predvolene všetky. Ak je riadkov priveľa, sú stránkované. Na obrázku je ako príklad rozvrh miestností na jeden týždeň. Bunka, ktorá je priesečníkom miestnosti a dňa je ešte vertikálne rozdelená na časové periódy (vyučovacie hodiny). Udalosť je zobrazená ako dlaždica, okupujúca takú časť bunky, ktorá korešponduje s časom jej konania. Udalosti je možné v mriežke

[1] `https://calendar.google.com`
[2] `https://fullcalendar.io/`



Obrázok 6: Návrh používateľského rozhrania klienta

presúvať myšou spôsobom drag-and-drop. Rovnako aj meniť ich veľkosť (trvanie), resp. vytvárať nové udalosti potiahnutým kurzoru po neobsadenej časti mriežky. Po kliku na názov zdroja (miestnosti) v prvom stĺpci sa zobrazí detailný rozvrh pre daný zdroj tak, že riadkami v tabuľke budú časové periódy. Teda obsah jej riadku v zozname zdrojov sa v podstate otočí o 90° a udalosti sa namiesto zoradenia v jednom riadku zľava doprava, zobrazia po riadkoch zhora nadol.

Dlaždice, reprezentujúce udalosti, sú farebne odlíšené a ich štítok a farba označujú zdroj, ktorý má v danom zobrazení najväčší zmysel. V našom príklade by to bola skratka a farba predmetu (resp. aktivity), pokiaľ by sme prezerali predmety, bola by to naopak názov a farba miestnosti. Svoju farbu môže mať priradený predmet, skupina (krúžok), miestnosť, resp. ich kategória, ktorá sa použije ako predvolená, ak nie je určená farba konkrétneho zdroja. Po vytvorení novej udalosti alebo kliknutím na už existujúcu udalosť sa otvorí dialógové okno, ktoré bude obsahovať formulár na editovanie jej atribútov.

4 Použité technológie

Prototyp, z ktorého vychádzame, bol implementovaný v jazyku Python 2 s použitím webového frameworku Django a databázy PostgreSQL na strane servera a na tvorbu grafického používateľského rozhrania využíva CSS framework Boostrap a jazyk JavaScript s knižnicou jQuery. V tejto kapitole popisujeme technológie, ktoré sme si vybrali na implementáciu nášho návrhu a zdôvodníme ich výber.

4.1 Serverová časť

Serverová časť systému pozostáva z webovej služby, ktorá poskytuje REST API a dáta ukladá do SQL databázy. Vznikla adaptáciou pôvodného prototypu systému.

4.1.1 Python

Python je vysoko-úrovňový, interpretovaný, interaktívny, programovací jazyk, vytvorený Guidom van Rossumom. Zahŕňa moduly, výnimky, dynamické typovanie a triedy. Kombinuje značnú expresivitu s veľmi čistou syntaxou a dobrou čitateľnosťou. [17] Zabezpečuje tiež automatickú správu pamäte a podporuje rozmanité programovacie paradigmy, vrátane objektovo-orientovanej, imperatívnej, procedurálnej a funkcionálnej. Má obsiahlu štandardnú knižnicu a jeho interpretre sú dostupné na širokej škále platforiem.

Aj keď Python 3 vyšiel v roku 2008, Python 2 je stále široko používaný a podporovaný, keďže Python 3 nie je plne spätne kompatibilný. Pôvodný systém bol napísaný vo verzii Python 2.7 aj kvôli podpore knižníc tretích strán. Keďže všetky knižnice, ktoré sme potrebovali využiť v súčasnosti podporujú Python 3 a oficiálna podpora Python 2 končí v roku 2020 [18] rozhodli sme sa prejsť na Python 3.5.

4.1.2 Django

Django je slobodný a open-source, vysoko úrovňový webový framework, napísaný v jazyku Python. Kladie dôraz na znovupoužiteľnosť a modulárnosť komponentov, rýchly vývoj a princíp DRY (neopakuj sa). Súčasťou Djanga je ORM modul, ktorý slúži na popísanie databázovej štruktúry a prístup k databáze v Python kóde. Tiež obsahuje modul na návrh URL schémy, ktorá obsahuje jednoduché mapovanie medzi URL vzormi a spätnými volaniami Python funkcií. [19]

Django sme aktualizovali na verziu 1.11, čo je posledná verzia, ktorá podporuje súčasne Python 2 aj 3 a preto má tiež predĺženú podporu. Použili sme aj niekoľko modulov tretích strán, medzi inými:

Django MPTT Nástroje na implementáciu ukladania hierarchických dát v databáze s

použitím štruktúry Modified Pre-order Traversal Tree (MPTT). Cieľom je, aby boli operácie pre načítanie dát efektívne.[3]

Django REST framework Flexibilná sada nástrojov na vytváranie webových API. [13]

Django REST framework JWT Podpora autentifikácie pomocou JWT pre Django REST framework.[4]

Django REST Swagger Generátor Swagger/OpenAPI dokumentácie pre Django REST framework.[5]

Django import/export Poskytuje rozhranie na import a export dát v rôznych formátoch.[6]

4.1.3 PostgreSQL

PostgreSQL je objektovo-relačný databázový systém s dôrazom na rozšíriteľnosť a dodržiavanie štandardov. Jeho transakcie sú plne ACID kompatibilné. Je to slobodný a open-source softvér. Beží na všetkých hlavných operačných systémoch a má natívne programovacie rozhranie pre veľké množstvo programovacích jazykov a dobrú podporu vo frameworku Django. [20]

4.2 Klientská časť

Klienta tvorí SPA, ktorá so serverom komunikuje prostredníctvom REST API. Na implementáciu sme si vybrali jazyk Elm, minimálne doplneným JavaScriptom, s použitím UI frameworku Bootstrap. Na uľahčenie písania CSS sme použili preprocesor Sass. Pre jazyk Elm neexistujú samostatné implementačné frameworky, ktoré by uľahčovali tvorbu aplikácií, akými sú napr. Angular alebo React pre JS. Túto rolu zohráva architektonický vzor The Elm Architecture (TEA), ktorý má podporu v štandardnej knižnici, je popísaný v oficiálnej dokumentácii a je de facto štandardom pre tvorbu aplikácii v Elme. Viac sa mu venujeme v podkapitole 5.2.1.

4.2.1 JavaScript

JavaScript (JS) je vysoko-úrovňový, dynamický, netypovaný, interpretovaný programovací jazyk. Bol štandardizovaný v jazykovej špecifikácii ECMAScript. Je jednou zo základných webových technológií a podporujú ho všetky moderné prehliadače. [21] Je tiež možné ho použiť na strane servera s runtime prostredím, ako je napr. Node.js.

[3] `https://django-mptt.github.io/django-mptt/`
[4] `https://getblimp.github.io/django-rest-framework-jwt/`
[5] `https://django-rest-swagger.readthedocs.io`
[6] `https://django-import-export.readthedocs.io/en/latest/`

4.2.2 Elm

Elm je čisto funkcionálny programovací jazyk, ktorý sa kompiluje do JavaScriptu. Kladie silný dôraz na jednoduchosť, ľahkosť použitia a kvalitné nástroje. Má silné statické typovanie s odvodzovaním, aj vďaka čomu je v ňom takmer nemožné vyvolať chybu vykonávania programu (runtime error). Obsahuje vlastnú výkonnú implementáciu virtuálneho DOM. Vďaka svojmu typovému systému dokáže automaticky detegovať zmeny v API. Túto informáciu využíva vo svojom oficiálnom katalógu balíčkov na vynútenie sémantického verziovania, pomocou čoho sa jeho používatelia vyhnú nepríjemným prekvapeniam pri aktualizácií balíčkov. [22]

Tieto vlastnosti jazyka Elm prispievajú k dobrej udržovateľnosti a ľahkému refaktorovaniu kódu. Aj napriek tomu, že sa jedná o pomerne mladý jazyk (vznikol v roku 2012 [23]) s obmedzenou ponukou rozširujúcich balíčkov, usúdili sme, že jeho prednosti prevažujú nad týmto dočasným nedostatkom a rozhodli sme sa, že ho použijeme na implementáciu webového klienta. V prípade, že by sme potrebovali využiť JS knižnicu alebo funkcionalitu prehliadača, ktorú Elm natívne nepodporuje, je tak možné urobiť, keďže poskytuje dobrú interoperabilitu s JS. Pužili sme Elm vo verzii 0.18.

4.2.3 Sass

Sass je preprocesor a skriptovací jazyk, ktorý sa interpretuje do CSS a je s ním plne kompatibilný. Rozširuje CSS o niekoľko mechanizmov, ktoré sú prítomné vo viac tradičných programovacích jazykoch ako sú napr. premenné, logické vrstvenie blokov, cykly, mixiny a dedenie selektorov. [24] Hlavnou prednosťou Sass je, že urýchľuje a sprehľadňuje písanie CSS.

4.2.4 Bootstrap

Bootstrap je jedným s najpopulárnejších HTML, CSS a JS front-end webový framework. Obsahuje šablóny pre typografiu, formuláre, tlačidlá, navigáciu a iné komponenty rozhrania, ako aj voliteľné JS rozšírenia. Takisto pomáha efektívne vytvárať responzívne stránky a aplikácie, ktorých vzhľad sa škáluje podľa veľkosti zariadenia, na ktorom sú zobrazené. Okrem CSS je jeho zdrojový kód prístupný aj vo forme zdrojových kódov pre preprocesory Less a Sass. [25]

4.3 Vývojové prostredie

Na vývoj sme použili OS Ubuntu 16.04 a editor GNU Emacs. Využili sme databázu PostgreSQL vo verzii 9.5 zo systémového repozitára balíčkov. Na správu verzií Pythonu a jeho virtuálnych prostredí (virtualenv) sme použili nástroj pyenv a na inštaláciu rozširu-

júcich Python modulov nástroj pip. Na zostavovanie a opätovné živé načítanie klientského kódu sme použili nástroj webpack. Na inštaláciu Elm platformy, webpacku a ostatných balíčkov, potrebných na vývoj klienta, sme použili správcu balíčkov npm s Node.js 6.10. Webovú službu aj klienta sme spúšťali lokálne na vývojovom HTTP serveri, dodávanom spolu s Djangom, resp. webpackom. Klienta sme testovali v prehliadačoch Firefox a Chromium.

5 Implementácia

V tejto kapitole opisujeme náš postup pri aktualizácii pôvodného systému a realizácii navrhnutých zmien, pričom niektorým dôležitým častiam sa venujeme detailnejšie. Zameriame sa predovšetkým na webového klienta, ktorý tvorí používateľské rozhranie systému. Na záver spomenieme odporúčané prostredie pre nasadenie aplikácie. Vyvíjaný rozvrhový systém dostal pracovný názov Elisa.

5.1 Webová služba

Serverová časť systému, teda webová služba, vznikla transformáciou pôvodného prototypu. Keďže sme sa rozhodli zmeniť jeho architektúru a zároveň prejsť na Python 3 a najnovšie Django, jednoduchšie, než meniť zdrojové kódy pôvodného projektu, bolo vytvoriť nový Django projekt a z pôvodného prevziať a upraviť iba časti, ktoré sa nám hodili. To boli predovšetkým modely (entitné triedy), ktoré v podstate predstavujú rozloženie (schému) databázy s dodatočnými metadátami. V Djangu je takýto model tzv. jediný, konečný zdroj pravdy (single source of truth), ktorého prepojenie s relačnou databázou sprostredkúva objektovo-relačné mapovanie ORM. Cieľom je zachovať princíp DRY: definovať dátový model na jednom mieste a ostatné veci z neho automaticky derivovať.

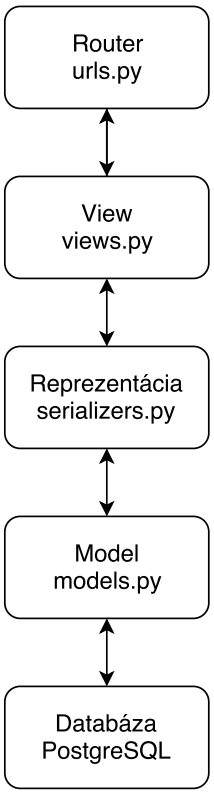
5.1.1 Štruktúra služby

Z hľadiska organizácie zdrojového kódu, ako aj biznis logiky je možné štruktúru služby popísať pomocou horizontálneho a vertikálneho pohľadu. Pri jej členení sme vychádzali z konvencií frameworku Django.

Horizontálne sa Django aplikácia skladá z hlavného projektu, čo je kolekcia nastavení (konfigurácia databázy, nastavenia Djanga samotného, nastavenia špecifické pre aplikáciu a pod.) a podaplikácií, tzv. apps, ktoré sú zároveň Python balíčkami. App je webová aplikácia, ktorá má vymedzenú funkcionalitu, napr. blog, zbierka obrázkov s mačičkami, systém komentárov atď. Jedna app môže byť vo viacerých projektoch. Apps v našej službe sú naše vlastné, ktoré do istej miery korešpondujú s balíčkami z návrhu v podkapitole 3.1.1 a tiež balíčky tretích strán, z ktorých niektoré sme popisujeme v podkapitole 4.1.2. Django projekt `elisa` v našom prípade korešponduje s balíčkom *Settings*. Kostru Django projektu alebo app je možné automaticky generovať pomocným programom, ktorý je súčasťou distribúcie Djanga.

Na obrázku 7 je znázornená vertikálna štruktúra služby. Funkcia jej častí je nasledovná:

Router Porovnáva cestu (z URL) zdroja, na ktorý sa klient dotazuje vo svojej požia-



Obrázok 7: Diagram vertikálnej štruktúry webovej služby

davke na server, so zoznamom regulárnych výrazov. Ak nájde zhodu, zavolá k nemu prislúchajúci view.

View Spracúva dotaz. Na základe jeho typu (`GET`, `POST`, `PUT` atď.) a parametrov rozhoduje, ktoré dáta budú prezentované (vrátené). Manipuluje s reprezentáciou zdrojov.[7]

Reprezentácia Slúži na serializáciu a deserializáciu inštancií modelov do, resp. z ich reprezentácií (JSON). Funguje podobne ako Django formuláre. Validuje dáta.

Model Jediný zdroj informácií o dátach a ich správaní v aplikácii. Manažuje stav databázy.

Databáza Perzistentné úložisko dát. Vo všeobecnosti je každý model namapovaný na jednu tabuľku v databáze. Na reprezentáciu vzťahov medzi modelmi môžu byť použité ďalšie tabuľky.

Na implementáciu API sme použili Django REST framework. Rozdelili sme ho na dve Django app, a to `school` a `timetables`. `school` má na starosti zdroje, týkajúce sa školy

[7]Autori Djanga používajú výraz view na pomenovanie komponentu, ktorý sa podobá controlleru v MVC, pozri `https://docs.djangoproject.com/en/dev/faq/general/`.

ako takej. Teda miestnosti, predmety, oddelenia atď. Dáta v app `timetables` sa týkajú rozvrhov a importujú sa v nej moduly zo `school`, opačne to neplatí. Autentifikáciu sme implementovali s využitím balíčka (app) Django REST framework JWT, s exspiráciou tokenu po dvoch dňoch a s možnosťou obnovy tokenu (bez nutnosti opätovného prihlásenia) po dobu maximálne dvoch týždňov. Implementovali sme všetky koncové body, popísané v návrhu v podkapitole 3.2.4, avšak API nemožno považovať za stabilné a jeho formát a funkcionalita podliehajú zmenám v závislosti od (budúcich) potrieb klienta. Výpis 1 obsahuje príklad tela odpovedi na dotaz `GET /rooms/376/`.

```
{
  "id": 376,
  "name": "cpu-a",
  "capacity": 20,
  "category": "učebňa počítačová",
  "department": 356,
  "equipment": [
    {
      "id": 41,
      "name": "tabuľa",
      "count": 1
    },
    {
      "id": 46,
      "name": "študentský počítač",
      "count": 20
    }
  ]
}
```

Výpis 1: Príklad JSON reprezentácie REST API zdroja – miestnosti

App `fei` slúži na import a export dát z FEI. Rozhodli sme sa ju implementovať nanovo s použitím balíčka Django import/export, ktorý tento proces robí robustnejším a uľahčuje prípadnú tvorbu analogickej app pre inú inštitúciu. Kvôli tomu sme potrebovali mierne upraviť formát vstupných CSV súborov, ktoré sme mali k dispozícii, avšak ich obsah a význam jednotlivých stĺpcov sa nezmenil a je popísaný v práci mojej predchodkyne, v prílohe A.4 [2].

5.1.2 Dokumentácia API

Na testovanie a dokumentáciu API sme použili balíček Django REST Swagger, ktorý slúži na generovanie dokumentácie vo formáte definovanom v špecifikácii OpenAPI (pôvodne Swagger). Tá určuje formát na popísanie REST API vo forme čitateľnej pre človeka aj počítač, s cieľom uľahčiť jeho tvorbu aj konzumáciu, vrátane generovania kódu a dokumentácie. [26]

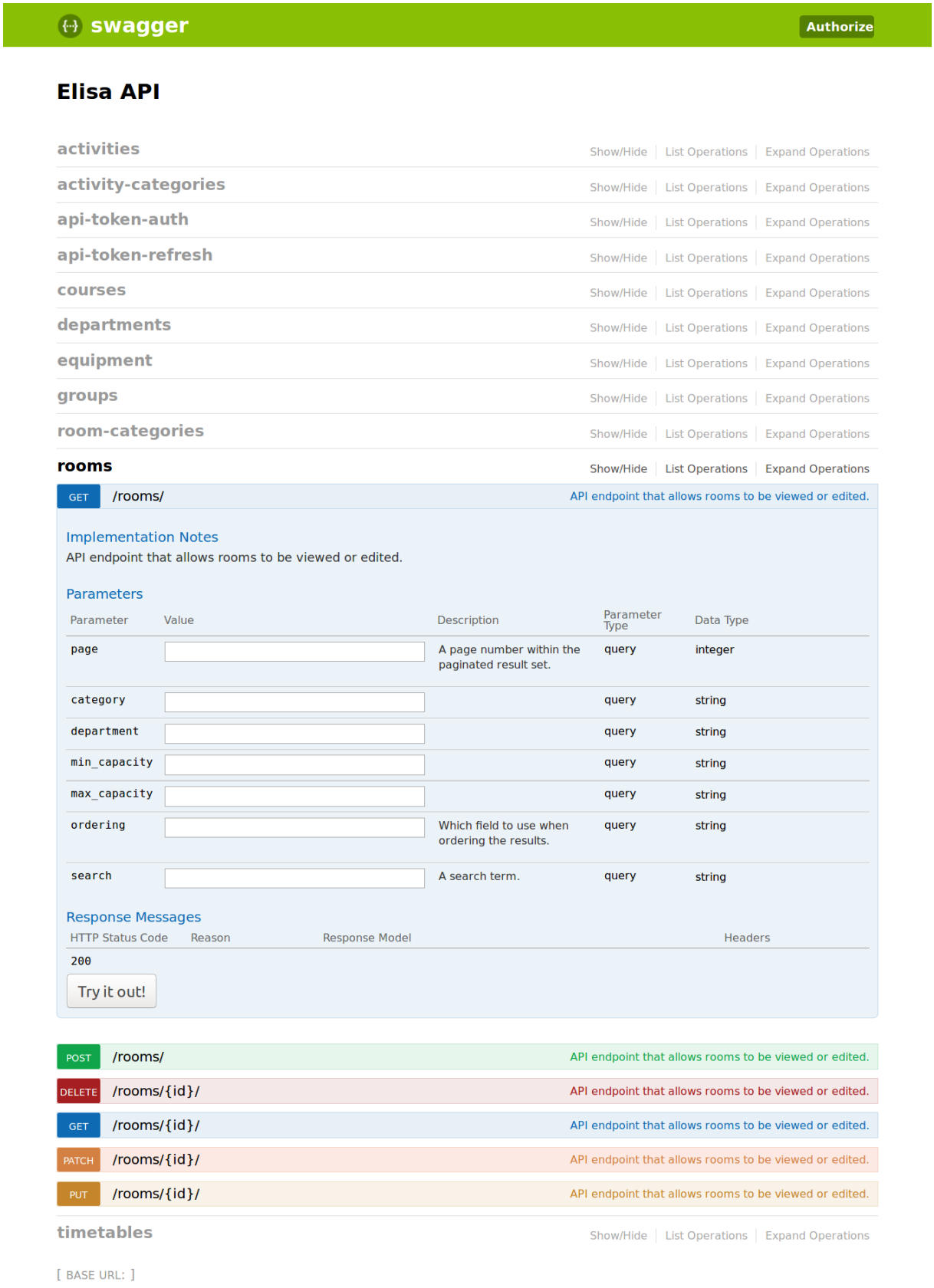
Django REST Swagger generuje z dátového modelu a tzv. Python docstringov, čo sú reťazce v zdrojovom kóde, slúžiace na jeho zdokumentovanie, najprv JSON schému a následne používateľské rozhranie (webovú stránku), na ktoré využíva Swagger UI. Ten umožňuje vizualizáciu a interakciu s API zdrojmi, vrátane posielania HTTP dotazov. Ukážku takto vytvorenej dokumentácie môžeme vidieť na obrázku 8.

5.2 Webový klient

Webového klienta sme implementovali ako SPA (jednostránkovú aplikáciu). Technicky ide o jednu webovú stránku, ktorá sa snaží používateľovi sprostredkovať podobnú skúsenosť ako desktopová aplikácia. Všetky zdrojové kódy klienta (HTML, JS a CSS) sa načítajú pri jedinom načítaní stránky. Stránka, na ktorej klient beží, sa počas normálneho behu neobnoví, ale jej URL sa mení v závislosti od aktuálneho stavu (zobrazenia) aplikácie. Každú takúto URL je možné použiť na otvorenie klienta (napr. ju zdieľať). Funkčná je aj navigácia v histórii prehliadača, aj keď v skutočnosti sa pri jej použití nemení samotná stránka, iba jej obsah. Pri interakcii s aplikáciou klient na pozadí dynamicky komunikuje so serverom.

Sústredili sme sa na implementáciu minimálnej podmnožiny požadovaných funkcií klienta, ktorá by umožnila použitie výsledného systému na FEI. Viaceré z implementovaných komponentov[8] sú vhodné na opätovné použitie, resp. poslúžia ako ukážka práce v jazyku Elm a využitia jeho prídavných balíčkov, čo pomôže pri ďalšom vývoji klienta. Medzi takéto komponenty patria:

[8]Slovo komponent používame vo význame „prvok aplikácie“, nie „objekt s enkapsulovaným stavom a metódami“. V Elme objekty neexistujú.



Obrázok 8: API dokumentácia vytvorená s pomocou Swagger UI

- smerovanie (routing) medzi stránkami (zobrazeniami) aplikácie,
- vytváranie dotazov na server,
- spracovanie odpovedí zo servera,
- autentifikácia a autorizácia,
- formulárové prvky a ich validácia,
- dialógové okná,
- drag-and-drop vytváranie elementov,
- interoperabilita s JS (tzv. porty).

5.2.1 The Elm Architecture

Elm poskytuje oficiálny spôsob ako organizovať prúd dát v aplikácii s názvom The Elm Architecture (TEA). TEA je jednoduchý vzor pre návrh webových aplikácií. Jeho výhodou je modularita, znovupoužiteľnosť kódu a efektívne testovanie. Zdá sa, že táto architektúra je dôsledkom dizajnu Elmu samotného, keďže sa prirodzene vyskytuje v Elm aplikáciách aj bez jej zamýšľaného použitia. TEA je ľahké použiť v Elme, ale je užitočná v ľubovoľnej interaktívnej front-end aplikácii. Dá sa rozdeliť na tieto základné, čisto oddelené časti [1]:

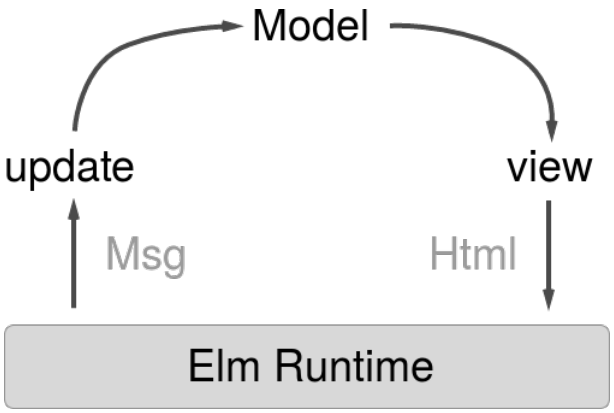
Model Stav aplikácie.

Udpate Slúži na aktualizáciu stavu.

View Slúži na zobrazenie stavu (ako HTML).

TEA na prvý pohľad pripomína tradičný vzor Model-View-Controller (MVC), avšak líši sa od neho v niekoľkých podstatných bodoch hlavne tým, že je viac obmedzený a deklaratívny:

- Model neobsahuje žiadnu biznis logiku, ani nijakým spôsobom nepopisuje správanie aplikácie, iba reprezentuje formu dát a aktuálny stav. Je jediným miestom, kde sa nachádzajú dáta aplikácie, pričom tieto dáta, tak ako všetky premenné v Elme, sú nemenné (immutable). Vždy, keď sú dáta transformované funkciou update, vytvorí sa nový model s aktualizovaným stavom.



Obrázok 9: Diagram The Elm Architecture [27]

- Update je jediným miestom, kde dochádza k transformácii modelu, čo uľahčuje uvažovanie o správaní programu. Je to funkcia, ktorej vstupom je správa a model a výstupom nový model, ktorý vznikne transformáciou predošlého modelu v závislosti od správy.

- View je bezstavová funkcia, ktorej vstupom je model a výstupom reprezentácia aktuálneho stavu (vo forme HTML). Tiež sa v nej registruje obsluha udalostí (množina event handlerov), ktorá emituje správy pre runtime, ktorý následne vyvolá spustenie `update` funkcie.

- Vstupným bodom Elm programu je funkcia `main`. Má na starosti prepojenie troch častí, popísaných vyššie. Jej výstupom je program, ktorý beži v runtime. Ten umožňuje obsluhu používateľských interakcií a manažment stavu. Runtime v nepretržitej slučke prijíma akcie od používateľa, mení stav a reprezentuje jeho zmeny. `update` a `view` sú iba funkcie na transformáciu dát, ich vykonávanie má na starosti runtime. [27] V štandardnej knižnici Elmu je implementovaných niekoľko rôznych funkcií, ktoré vytvoria program.

- Táto architektúra je do určitej miery vynútená aj na úrovni typového systému, ktorý obmedzuje možné typy vstupov a výstupov update, view a funkcií pre vytvorenie programu zo štandardnej knižnici. A teda nie je iba konvenciou.

- Dáta prúdia iba jedným smerom a na ich transformáciu sa používajú funkcie.

Bez toho, aby sme zachádzali do podrobností sytaxe Elmu, si priblížime jednoduchý príklad programu v TEA, ktorého zdrojový kód môžeme vidieť na výpise 2. Obsahuje počítadlo a dva tlačidlá, ktoré slúžia na jeho inkrementáciu a dekrementáciu. Ako `model`

nám postačuje jedno celé číslo. Vstupom do funkcie `update` je okrem modelu správa, ktorá môže nadobúdať iba hodnoty `Increment` alebo `Decrement` a jej výstupom je nový model, teda aktualizovaná hodnota počítadla. Funkcia `view` reprezentuje `model` v HTML, pričom pre jednotlivé tlačidlá registruje obsluhu udalosti kliknutia, ktorá bude produkovať správu `Increment`, resp. `Decrement`. `div` a `button` sú normálne Elm funkcie, ktorých vstupom je zoznam atribútov a zoznam detských uzlov. Celý view kód je kompletne deklaratívny. Nepotrebujeme upravovať DOM manuálne, túto prácu za nás robí Elm runtime. Počiatočný `model` a funkcie `update` a `view` prepája funkcia `beginnerProgram` z modulu `Html` štandardnej knižnice.

5.2.2 Štruktúra klienta

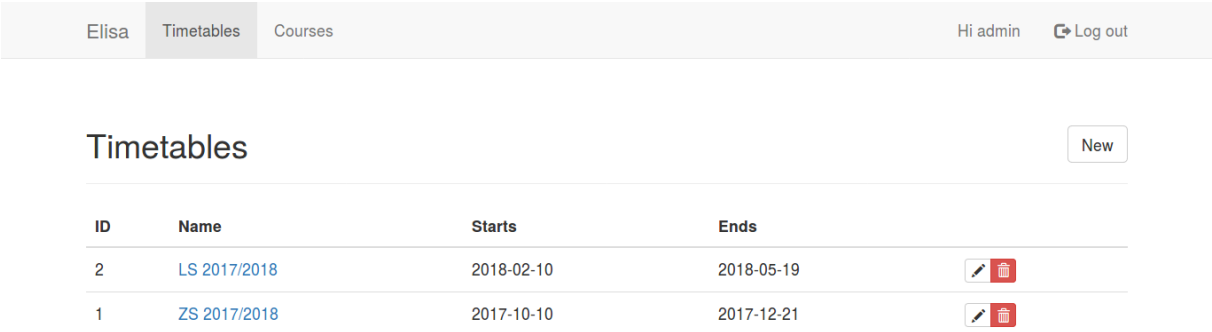
Nášho klienta sme implementovali pomocou TEA hierarchicky. Každá signifikantná, samostatná časť (podstránka) aplikácie má vlastný model, update a view, ktoré sú súčasťou globálneho (koreňového) modelu, update a view. Podobným spôsobom bývajú implementované aj rozširujúce balíčky, ktoré poskytujú nejaké enkapsulované UI komponenty ako napr. formuláre, dialógové okná, autocomplete a pod. Dáta, ktoré sú spoločné pre celú aplikáciu (všetky podstránky), v našom prípade informácie o prihlásenom používateľovi (dátový typ `User`), na vstup dostanú funkcie, ktoré ich potrebujú. Každá podstránka (zobrazenie) korešponduje s určitou URL, napr. zobrazenie rozvrhov má cestu `/timetables`.

Keďže Elm je silno staticky typovaný, dáta ktoré poskytuje server vo formáte JSON je potrebné previesť do Elm hodnôt, na čo sa používajú tzv. dekodéry. Štandardná knižnica poskytuje primitívne dekodéry pre základné dátové typy (`string`, `int`, `bool` a pod.), ktoré je možné kombinovať a vytvoriť tak dekodér pre zložený dátový typ (napr. zoznam predmetov). Takáto konverzia zároveň slúži aj na validáciu. Nielenže konvertujeme dáta z JSON, ale sa tiež uisťujeme, že JSON dodržuje určitú štruktúru. Možné chyby sa tak zachytia už pri prijatí dát a nie až pri ich použití. Obdobný proces sa používa na zakódovanie Elm hodnôt do JSON hodnôt.

Jediný stav aplikácie, ktorý sa zachováva aj po jej ukončení, je informácia o prihlásenom používateľovi. Po úspešnej autentifikácií sa autentifikačný token (JWT) uloží do `localStorage`, čo je perzistentné úložisko prehliadača, v ktorom sú dáta uložené ako páry kľúčov a hodnôt. `localStorage` je vymedzený pôvodom dokumentu (pravidlá same-origin), teda k tým istým dátam nie je možné pristupovať z dokumentov s dvomi rôznymi URL. [21] Po spustení klienta prebehne pokus o načítanie tokenu z `localStorage`. Ak sa tam nejaký nachádza a je valídny, tak sa používateľ pokladá za prihláseného. Pomocou `localStorage` tak vieme zachovať aktuálny stav (resp. jeho časť) klienta aj po vypnutí prehliadača. Úložisko `localStorage` je tiež možné využiť na implementáciu offline fun-

```elm
module Main exposing (..)

import Html exposing (Html, button, div, text)
import Html.Events exposing (onClick)

main =
    Html.beginnerProgram { model = model, view = view, update = update }

type alias Model = Int

model : Model
model = 0

type Msg = Increment | Decrement

update : Msg -> Model -> Model
update msg model =
    case msg of
        Increment ->
            model + 1

        Decrement ->
            model - 1

view : Model -> Html Msg
view model =
    div []
        [ button [ onClick Decrement ] [ text "-" ]
        , div [] [ text (toString model) ]
        , button [ onClick Increment ] [ text "+" ]
        ]
```

Výpis 2: Ukážka Elm programu v TEA [1]

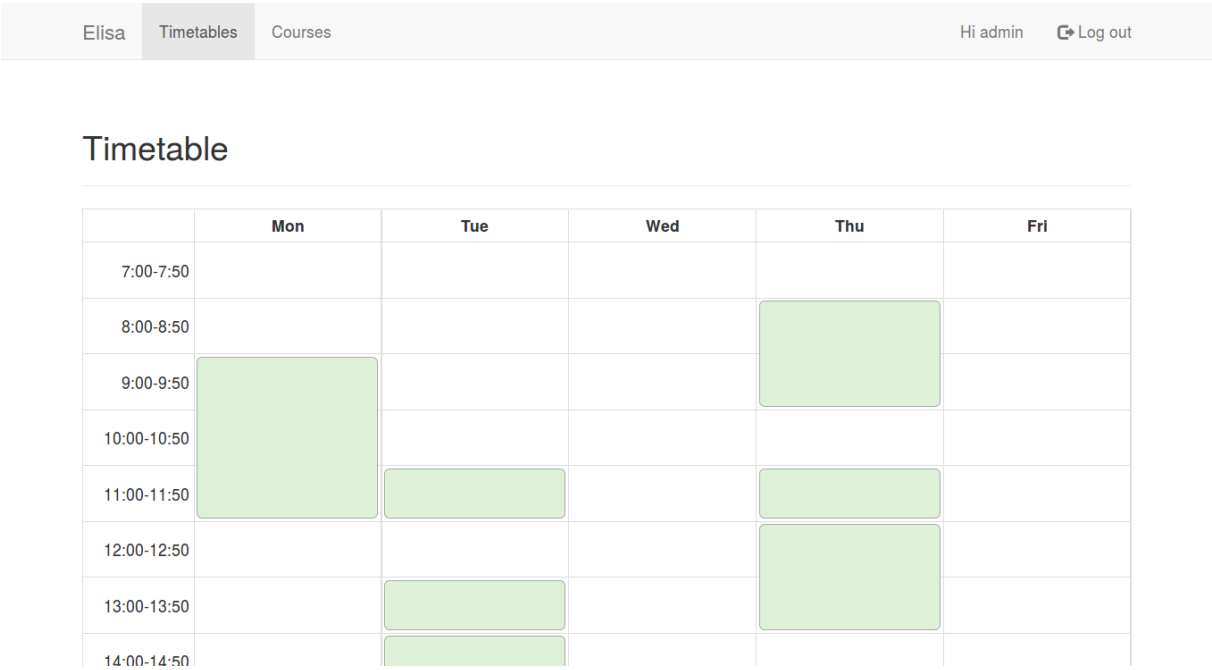


Obrázok 10: Používateľské rozhranie správy rozvrhov

kcionality. Tým, že sa autentifikačný token ukladá do `localStorage` a nie do cookie, sme sa tiež vyhli útokom typu Cross-site request forgery (CSRF), keďže sa súčasťou dotazu na server na rozdiel od cookie nestáva automaticky.

5.2.3 Používateľské rozhranie

Pri implementácii používateľského rozhrania sme vychádzali z návrhu v podkapitole 3.3.2. Je plne responzívne vďaka využitiu frameworku Bootstrap. Tiež sa mení podľa právomocí prihláseného používateľa. Pokiaľ používateľ nie je autorizovaný, má prístup len na čítanie. Na obrázkoch 10 a 11 sú jeho ukážky.



Obrázok 11: Používateľské rozhranie úpravy časovej mriežky

Záver

Autorka práce Rozvrhový systém pre vysoké školy navrhla a implementovala prototyp rozvrhového systému, ktorý by mal v konečnom dôsledku úspešne konkurovať iným systémom tohto typu. [2] Cieľom našej práce bolo aktualizovať v ňom použité technológie a pokračovať v jeho vývoji.

Najskôr sme sa informovali o probléme tvorby školských rozvrhov a niekoľkých jeho riešeniach v podobe už existujúcich systémov. Oboznámili sme sa so súčasným procesom tvorby rozvrhov na FEI a požiadavkami, ktoré sa na tieto rozvrhy kladú. Naštudovali sme si prácu Rozvrhový systém pre vysoké školy vrátane zdrojových kódov prototypu systému, ktorý v rámci nej vznikol a vyskúšali si jeho používanie.

Okrem aktualizácie v ňom použitých technológií sme navrhli niekoľko možností ako tento systém zlepšiť. Rozhodli sme sa pozmeniť architektúru systému a pretransformovať ho na webovú službu s REST API a webového klienta vo forme SPA s cieľom zvýšiť jeho interoperabilitu s inými systémami a umožniť vytvorenie viacerých klientov napr. vo forme natívnych (mobilných) aplikácií. Dôsledkom toho je ešte o čosi vyššia modularita celého systému, ktorá už pri jeho pôvodnom návrhu zohrávala dôležitú rolu. Takéto oddelenie tiež uľahčuje interaktívnu manipuláciu s dátami na strane klienta a znižuje odozvu používateľského rozhrania. Dátový model na strane servera sme z väčšej časti, ponechali pôvodný. Popísali sme technológie, ktoré sme použili na realizáciu nášho návrhu a odôvodnili ich výber. Priblížili sme si postup pri aktualizácii a ďalšom vývoji prototypu. Detailnejšie sme opísali zaujímavé aspekty implementácie systému a jeho fungovania.

Softvér zatiaľ stále nie je vhodný na nasadenie, čo je spôsobené jednak celkovou komplexnosťou problému a tiež tým, že s veľkou časťou použitých technológií sme sa stretli prvý krát. Existuje značný priestor na jeho ďalšie vylepšenie a jeho kompletizácia a prípadné nasadenie na FEI si zrejme bude vyžadovať dlhodobé úsilie a úzku spoluprácu s personálom fakulty. Zmena architektúry systému však otvára mnohé nové možnosti pre jeho ďalší vývoj, napr.:

- vývoj servera a klienta môže prebiehať oddelene,
- vývoj viacerých nezávislých klientov pre rôzne platformy,
- klient môže poskytovať offline funkcionalitu,
- integrácia servera s inými službami, napr. AIS alebo službami tretích strán.

Zoznam použitej literatúry

1. CZAPLICKI, Evan. *An Introduction to Elm* [online] [cit. 2017-04-21]. Dostupné z: `https://guide.elm-lang.org/`.
2. KNAPEREKOVÁ, Emília. *Rozvrhový systém pre vysoké školy*. 2014. Diplomová práca. Slovenská technická univerzita v Bratislave. EČ: FEI-5384-56111.
3. LUKÁČ, Matej. *Course timetabling at Masaryk University in the UniTime system*. 2013. Dostupné tiež z: `https://is.muni.cz/th/255726/fi_m/Master_Thesis.pdf`. Diplomová práca. Masaryk University.
4. *UniTime* [online] [cit. 2017-03-21]. Dostupné z: `http://www.unitime.org/present/unitime-highlights.pdf`.
5. *School scheduling software for Mac, PC, tablet and phone - Prime Timetable* [online] [cit. 2017-04-04]. Dostupné z: `http://www.primetimetable.com`.
6. *Mimosa - Scheduling Software for School and University Timetables* [online] [cit. 2017-04-04]. Dostupné z: `http://www.mimosasoftware.com/`.
7. *Mimosa Scheduling Software Freeware* [online] [cit. 2017-04-04]. Dostupné z: `http://mimosa-scheduling-software.soft112.com/`.
8. MÜLLER, Tomáš. *UniTime - University Timetabling* [online] [cit. 2017-03-16]. Dostupné z: `http://www.unitime.org/`.
9. W3C. *Web Services Glossary* [online] [cit. 2017-04-24]. Dostupné z: `https://www.w3.org/TR/2004/NOTE-ws-gloss-20040211`.
10. W3C. *Web Services Architecture* [online] [cit. 2017-04-24]. Dostupné z: `https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/`.
11. FIELDING, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures*. 2000. Dizertačná práca. University of California.
12. ECMA INTERNATIONAL. *The JSON Data Interchange Format* [online]. 2013 [cit. 2017-04-24]. No. 404. Dostupné z: `https://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf`.
13. CHRISTIE, Tom. *Django REST framework* [online] [cit. 2017-03-18]. Dostupné z: `http://www.django-rest-framework.org/`.

14. RESCHKE, J. *The 'Basic' HTTP Authentication Scheme* [online]. RFC Editor, 2015 [cit. 2017-03-20]. ISSN 2070-1721. Dostupné z: `https://tools.ietf.org/html/rfc7617`. RFC. RFC Editor.

15. JONES, M., BRADLEY, J. and SAKIMURA, N. *JSON Web Token (JWT)* [online]. RFC Editor, 2015 [cit. 2017-03-22]. ISSN 2070-1721. Dostupné z: `https://tools.ietf.org/html/rfc7519`. RFC. RFC Editor.

16. *JSON Web Token Introduction* [online] [cit. 2017-03-23]. Dostupné z: `https://jwt.io/introduction/`.

17. *General Python FAQ* [online] [cit. 2017-03-18]. Dostupné z: `https://docs.python.org/3/faq/general.html`.

18. PETERSON, Benjamin. *PEP 373 – Python 2.7 Release Schedule* [online]. 2008 [cit. 2017-03-18]. Dostupné z: `https://www.python.org/dev/peps/pep-0373/`.

19. *Django* [online] [cit. 2017-03-18]. Dostupné z: `https://djangoproject.com/`.

20. *PostgreSQL: About* [online] [cit. 2017-03-18]. Dostupné z: `https://www.postgresql.org/about/`.

21. FLANAGAN, David. *JavaScript: The Definitive Guide*. Sixth Edition. O'Reilly & Associates, 2011. ISBN 978-0-596-80552-4.

22. CZAPLICKI, Evan. *Elm* [online] [cit. 2017-03-19]. Dostupné z: `http://elm-lang.org/`.

23. CZAPLICKI, Evan. *Elm: Concurrent FRP for Functional GUIs* [online]. 2012 [cit. 2017-03-19]. Dostupné z: `http://elm-lang.org/assets/papers/concurrent-frp.pdf`.

24. CATLIN, Hampton, WEIZENBAUM, Natalie, EPPSTEIN, Chris, et al. *Sass: Syntactically Awesome Style Sheets* [online] [cit. 2017-04-25]. Dostupné z: `http://sass-lang.com/`.

25. OTTO, Mark, THORNTON, Jacob, et al. *Bootsrap* [online] [cit. 2017-03-19]. Dostupné z: `https://getbootstrap.com/`.

26. *Swagger Specification* [online] [cit. 2017-05-06]. Dostupné z: `http://swagger.io/specification/`.

27. REIMANN, Dennis. *The Elm Architecture – Simple, yet powerful – An overview by example* [online] [cit. 2017-04-21]. Dostupné z: `https://dennisreimann.de/articles/elm-architecture-overview.html`.

Prílohy

A Technická dokumentácia

Na vývoj systému by malo vyhovovať akékoľvek prostredie, ktoré podporuje Python 3, PostgreSQL a Node.js. Pre produkčné prostredie postačuje Python 3 a PostgreSQL. Postup ako nastaviť vývojové prostredie sme popísali v súboroch `README.md`, ktoré sa nachádzajú v priečinku so zdrojovými súbormi servera, resp. klienta na priloženom nosiči. Prostredie, ktoré sme používali na vývoj my, je popísane v podkapitole 4.3.

Webovú službu aj klienta je možné spúšťať lokálne s použitím vývojových webových serverov, ktoré sú súčasťou ich závislostí. Ale na nasadenie systému je potrebné použiť plnohodnotný webový server ako napr. Apache HTTP Server alebo Nginx. Kód klienta stačí zostaviť vo vývojovom prostredí a na server iba nahrať výsledné statické súbory (HTML, CSS, JS atď.), ktoré môže servírovať ten istý server ako webovú službu. Nasadenie služby odporúčame konzultovať s Django dokumentáciou[9]. Je potrebné zabezpečiť, aby všetka komunikácia so serverom prebiehala cez HTTPS, na čo navrhujeme využiť Let's Encrypt[10].

[9] `https://docs.djangoproject.com/en/1.11/howto/deployment/`
[10] `https://letsencrypt.org/`

B Obsah elektronického nosiča

Uvádzame prehľad iba najdôležitejších súborov a priečinkov.

```
/
|-- elisa                          # zdrojové súbory prototypu systému
|   |-- elisa-server               # serverová časť
|   |   |-- elisa                  # Django projekt, obsahuje nastavenia
|   |   |-- fei                    # balíček na import a export
|   |   |-- fei-data               # testovacie dáta určené na import
|   |   |-- manage.py              # skript na správu Django projektu
|   |   |-- README.md              # dokumentácia
|   |   |-- requirements.txt       # Python pip závislosti
|   |   |-- school                 # balíček na prácu so školskými dátami
|   |   `-- timetables             # balíček na prácu s rozvrhmi
|   `-- elisa-web                  # klientská časť
|       |-- elm-package.json       # metadáta Elm balíčku
|       |-- package.json           # metadáta npm balíčku
|       |-- README.md              # dokumentácia
|       |-- src                    # zdrojový kód klienta
|       |   |-- elm                # Elm súbory
|       |   `-- static             # HTML, JS, SCSS, obrázky
|       `-- webpack.config.js      # nastavenia webpack
`-- thesis                         # zdrojové súbory tejto práce
```