SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: FEI-5384-46049

TVORBA UŽÍVATEĽSKÉHO ROZHRANIA K ROZVRHOVÉMU SYSTÉMU PRE FEI

DIPLOMOVÁ PRÁCA

2018 **Bc. Dávid Bednár**

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: FEI-5384-46049

TVORBA UŽÍVATEĽSKÉHO ROZHRANIA K ROZVRHOVÉMU SYSTÉMU 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 2018 **Bc. Dávid Bednár**

SÚHRN

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Študijný program:	Aplikovaná informatika
Autor:	Bc. Dávid Bednár
Diplomová práca:	Tvorba užívateľského rozhrania k rozvrhovému systému pre FEI
Vedúci záverečnej práce:	Mgr. Ing. Matúš Jókay, PhD.
Miesto a rok predloženia práce:	Bratislava 2018

Tvorenie rozvrhov je rozsiahly a časovo náročný proces, ktorým sa každoročne zaoberá väčšina vzdelávacích inštitúcií vrátane FEI. Kvôli tejto motivácii vznikol v roku 2014 v rámci diplomovej práce (Emília Knapereková, 2014) prototyp rozvrhového systému pre vysoké školy. Na základe tohto prototypu sa neskôr v ďalšej diplomovej práci vytvorila základná aplikácia rozvrhu (Martin Račák, 2017). Cieľom tejto práce je aktualizácia projektu a pokračovanie v jeho vývoji, s dôrazom kladeným na používateľské rozhranie systému. Oboznámili sme sa s predošlými prácami na tomto systéme a stav implementácie rozhrania sme zmenili z funkcionálnej v jazyku Elm na objektovú v jazyku Angular 5 s obohatením o prvky funkcionálneho programovania pomocou Redux manažmentu stavov. Hlavným prínosom systému má byť jednoduchosť a intuitívnosť pri jeho používaní, a teda aj jednotlivé funkcie a možnosti sú orientované na tieto požiadavky. V závere našej práce sa nachádza zhodnotenie jej prínosu a dosiahnuté výsledky v porovnaní so stanovenými cieľmi na začiatku projektu.

Kľúčové slová: rozvrhový systém, používateľské rozhranie, angular 5

ABSTRACT

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

Study Programme:	Applied Informatics
Author:	Bc. Dávid Bednár
Master's thesis:	GUI implementation for FEI schedule IS
Supervisor:	Mgr. Ing. Matúš Jókay, PhD.
Place and year of submission:	Bratislava 2018

Schedule creation is an extensive and time consuming process that is being undertaken annually by most educational institutions including FEI. Due to this motivation, first prototype of a university schedule system (Emília Knapereková, 2014) was created in the master thesis in 2014. Based on this prototype, the basic application of the schedule was created later in the next master thesis (Martin Račák, 2017). We have become familiar with previous work on this system, and the state of implementation of the interface has changed from functional in Elm to object oriented in Angular 5 with enhancement of Functional Programming elements using Redux State Management. The main benefits of the system are the simplicity and intuitiveness of its use, and therefore the various functions and options are oriented to these requirements. At the end of our work, we evaluated its benefits and achieved results compared to the goals set at the beginning of the project.

Keywords: scheduling system, user interface, angular 5

Poďakovanie

Chcel by som sa v prvom rade poďakovať vedúcemu projektu Mgr. Ing. Matúš Jókay, PhD za nasmerovanie pri riešení tejto diplomovej práce a za informácie a rady poskytnuté pri konzultáciach. Ďalšie moje poďakovania smerujú Ing. Martinovi Račákovi za nezištné a nápomocné oboznámenie so stavom predchádzajúceho riešenia projektu. Rodine a priateľom ďakujem za trpezlivosť a za poskytnutie vhodného pracovného prostredia na úspešné dokončenie diplomovej práce.

Obsah

Zoznam obrázkov a tabuliek

Úvod

V minulosti, pred etablovaním počítačov do života ľudí, sa väčšina činností, ako aj zostavovanie rozvrhov, vykonávala manuálne. Rozvrhár musel ručne spracovávať zhromaždené údaje, odsledovať kolízie a nastaviť celý proces tak, aby správne fungoval a dalo sa podľa rozvrhu spoľahlivo vyučovať. Pri vysokých školách je tento problém potrebné riešiť aj viac ako dvakrát do roka kvôli rôznorodosti výučbového obdobia (zimný / letný semester, semester výučby / skúškové obdobie). Komplexnosť (počet vyučujúcich, študentov, predmetov a miestností), jednotlivé pravidlá (všeobecné aj osobitné) a rozmanitosť výnimiek celý proces ešte viac komplikujú. V tejto sfére sa počítače stali vhodným nástrojom pri zozbieraní dát, riešení problémov a kolízií, generovaní rozvrhov a za pomoci internetu aj v jednoduchom distribuovaní online pre koncových používateľov, ako sú samotní vyučujúci či študenti. Tak, ako sa časom vyvíjali takéto systémy, sa vyvíjali aj nové štandardy, technológie a trendy v oblasti vývoja informačných systémov. Existuje veľký počet komerčných softvérových produktov, ktoré pomáhajú s problémom tvorenia rozvrhov, avšak kvôli rôznorodosti, špecifickým požiadavkám, postupom a rôznej zložitosti každej inštitúcie sa stále vedú zaujímavé štúdie v oblasti plánovania procesov pri tvorbe rozvrhov ako aj v oblasti dizajnu a návrhu rozhrania pre koncových používateľov.

V prvej kapitole našej práce sa venujeme analýze problému. Popisujeme všeobecný proces tvorby rozvrhov a zameriavame sa na vystupujúce entity a fakty, ktoré sú zohľadňované pri výstupoch výsledných rozvrhov. Analyzujeme stav existujúcich riešení, ktoré sa v súčasnosti využívajú na rozvrhovanie s dôrazom na ich funkcionalitu, možnosti a náhľad používateľského rozhrania. V druhej kapitole sme sa zamerali na zosumarizovanie cieľov práce na základe špecifikácie požiadaviek kladených na systém. Podrobnejšie popisujeme spôsob, ako v súčasnosti prebieha proces tvorby rozvrhu na našej fakulte FEI. Tretiu časť diplomovej práce venujeme samotnému návrhu riešenia. Od popisu vysokoúrovňovej architektúry sa jednotlivými podkapitolami približujeme ku konkrétnym architektonickým prvkom používaných v aplikácii. V štvrtej kapitole predstavujeme použité technológie, pre ktoré sme sa rozhodli pri implementácii navrhovaného systému. Poslednú piatu kapitolu venujeme samotnej implementácii riešenia a popisu jednotlivých modulov spolu s ukážkami a vysvetleniami základných najdôležitejších prvkov aplikácie.

V práci budeme na označenie cudzojazyčných výrazov, prípadne na označenie názvov modulov, ktoré je kvôli porozumeniu lepšie neprekladať, používať písmo štýlu *italic*. Príklady príkazov budeme pre zvýraznenie od bežného textu odlišovať písmom `technického štýlu`.

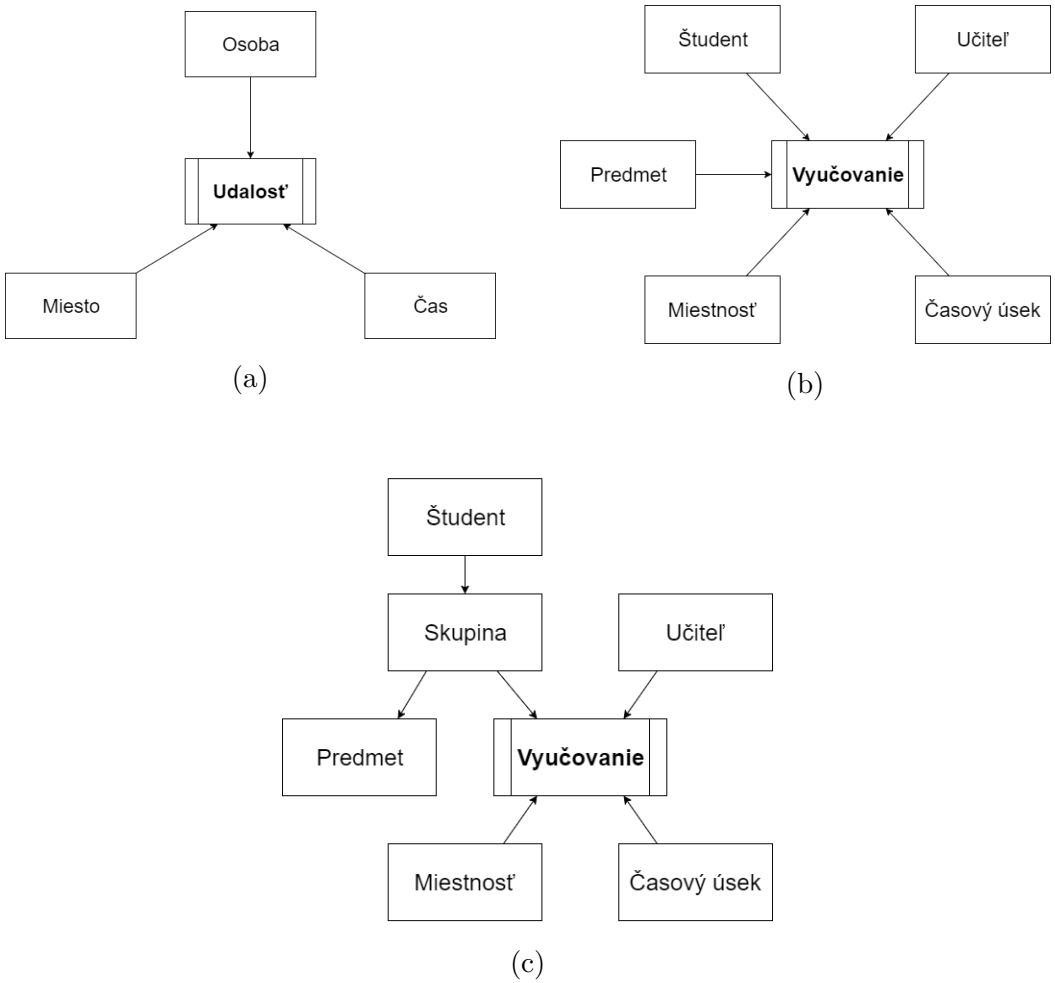
1 Analýza problému

Vo všeobecnosti sa za rozvrhový systém považuje softvér, pomocou ktorého sa vytvárajú rozvrhy. Pri tvorení rozvrhu sa súbor udalostí priraďuje do určitého počtu časových okamihov, ktoré podliehajú istým súborom obmedzení a pravidiel. Zvyšovaním počtu udalostí a ohraničení sa zvyšuje aj množstvo kolízií, ktoré je potrebné minimalizovať či úplne vylúčiť, no zároveň zachovať požadované obmedzenia a výnimky. Plánovanie a tvorba akademického rozvrhu je výpočtovo náročný a obvykle zložitý proces. Je to NP úplný problém optimalizácie, ktorý potrebuje heuristický prístup k nájdeniu riešenia. Hlavnou motiváciou výskumov v tejto oblasti je minimalizovať celkovú cenu použitých zdrojov pri celom procese vytvárania rozvrhov.

Samostatnou dôležitou súčasťou riešenia, je nastavenie vhodného zobrazovania informácií používateľom v systéme a prispôsobiť ho tak, aby bolo používanie čo najviac intuitívne a prehľadné ako pre rozvrhára, ktorý bude kontrolovať vygenerovaný rozvrh a ručne upravovať kolízie a udalosti podľa špecifík, tak aj pre vyučujúcich a študentov, ktorý si budú prezerať svoje rozvrhy. Veľmi dôležitou funkcionalitou je okrem samotného generovania aj manuálna úprava rozvrhu, ktorá je z hľadiska časovej náročnosti pre rozvrhára najväčším zdržaním pri tvorbe rozvrhov (nakoľko automaticky vygenerované rozvrhy nedokážu spoľahlivo nájsť riešenia na všetky kolízie, osobité požiadavky, výnimky atď).

1.1 Proces tvorby rozvrhu

Pri procese tvorby rozvrhu sa jedná o minimalizačný problém, ktorého cieľom je prehľadať priestor všetkých dostupných možností a nájsť to najlepšie riešenie. Výstup však musí spĺňať čo najviac vstupných požiadaviek a zároveň zaručiť splnenie všetkých ohraničení daného prehľadávaného priestoru. Tieto ohraničenia sa týkajú jednotlivých vstupných elementov ako sú čas (kedy), priestor (kde), osoba (kto) a iné. [1]



Obrázok 1: Pojmy a vzťahy medzi nimi používané pri rozvrhoch vo všeobecnosti (a), pri rozvrhoch na školách (b) a pri rozvrhoch so zaradením študentov do skupín (c) [1]

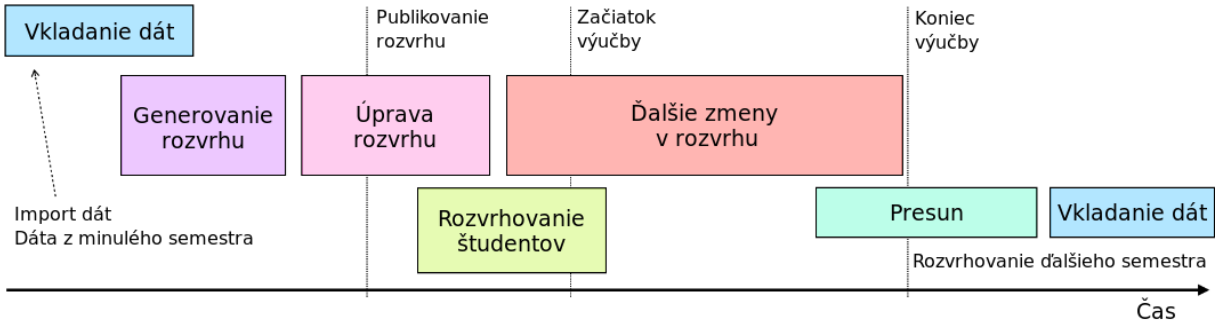
Ohraničenia sú rozdelené na tvrdé ohraničenia (z angl. Hard Constraints) a mäkké ohraničenia (z angl. Soft Constraints). Tvrdé ohraničenia nesmú byť porušované (napr. študent/vyučujúci môže byť v jednom časovom okamihu prítomný iba na jednom mieste), zatiaľ čo tie mäkké nie je potrebné dodržať, avšak ovplyvňujú bonitu celého navrhovaného rozvrhu (napr. preferencia časových úsekov, individuálne potreby). [2]

Taktiež sa zohľadňujú dáta z predchádzajúcich cyklov a období. Požiadavky sú okrem doby inicializácie kladené aj počas obdobia semestra, čiže sú dynamické a časom sa môžu meniť. Jedná sa hlavne o prípady jednorázových udalostí (napr. nahrádzané prednášky a cvičenia, konferencie, mimovýučbové prednášky). Rozvrh s vyššou bonitou sa všeobecne považuje za lepší, pretože splnil viac mäkkých ohraničení a je vhodnejší na použitie. To, či je obmedzenie tvrdé alebo mäkké, sa definuje pri vývoji matematického modelu, ktorý stojí za problémom plánovania. Na základe toho sa od seba líšia mnohé navrhnuté

implementácie.

Za rozvrh univerzity je spravidla zodpovedný jeden prípadne viacero rozvrhárov (napr. každý má na starosti rozvrh inej fázy počas roka - výučba / skúškové obdobie). V dnešnej dobe prevláda zväčša kombinovaná tvorba rozvrhu, kedy sa automaticky generovaný rozvrh manuálne upravuje rozvrhárom na základe zozbieraných požiadaviek a určitých obmedzení. Obmedzenia do systému zadávajú samotní vyučujúci a študenti, prípadne je na túto činnosť pre dané oddelenie priradený zástupca oddelenia, ktorý má túto činnosť na starosti.

Na nasledujúcom obrázku je znázornený harmonogram akcií pri tvorbe rozvrhu počas semestra:



Obrázok 2: Harmonogram akcií pri tvorbe rozvrhu [3]

1.2 Existujúce rozvrhové systémy

Rozmanitosť existujúcich rozvrhových systémov zahŕňa aj pestrosť v ich ponúkaných funcionalitách a možnostiach. Ich rozdielnosť najčastejšie spočíva v:

- podporovanom operačnom systéme,
- licencii a cene,
- dizajne používateľského rozhrania,
- oblasti zamerania (základné školy, stredné školy, univerzity a ďalšie),
- spôsobe tvorenia rozvrhu (manuálny, automatický, kombinovaný),
- použitom algoritme generovania rozvrhu,
- možnostiach úprav zdrojov, udalostí a iných vstupných dát,
- možnostiach importu a exportu,

- možnostiach administratívy,
- zabezpečení.

V nasledujúcich podkapitolách si priblížime niektoré z nich z hľadiska poskytovaných funkcionalít a možností a aj z hľadiska dizajnu používateľského rozhrania.

1.2.1 CelCat

CELCAT Timetabler[1] poskytuje integrovaný prístup k časovému rozvrhovaniu a rezervácii miestností, pričom sa snaží o čo najlepšie využitie zdrojov, priestoru a času. Jednotný prístup k správe dát umožňuje časomeračom a priestorovým rezervátorom v rámci celej univerzity s cieľom získať prístup k jedinému, jasnému a koherentnému zdroju informácií o rozvrhovom poriadku.

V CELCAT sú rozvrhy zostrojené ako mriežky, kde sú dni rozdelené na časové bloky, ktoré sú označované ako obdobia. Udalosti sa vytvárajú v časovej mriežke rozvrhu, ak je jeden alebo viac zdrojov (napr. zamestnanci, miestnosti, študenti atď.), ktoré sa majú uskutočniť počas jedného alebo viacerých období.

Prístup k CELCAT je zabezpečený prostredníctvom úloh špecifických pre jednotlivé oddelenia. Používatelia majú zvyčajne prístup k zdrojom a rozvrhom, ktoré sú priamo spojené s ich oddelením. Za výnimočných okolností môžu používatelia mať prístup k zdrojom, ktoré nie sú okamžite spojené s ich oddelením, ak to predtým schválili všetky zainteresované oddelenia a udelili mu povolenie [4].

Existujú tri typy používateľov CELCAT:

- Hlavný rozvrhár celého oddelenia
- Osoba rezervujúca miestnosti
- Prezerajúci používateľ (z angl. Read-only user)

Všetci používatelia majú prístup len na čítanie ku všetkým zdrojom a rozvrhom, ktoré sú v rámci CELCAT-u. S cieľom zaistiť bezpečnosť údajov sa používateľské účty vytvárajú len pre pracovníkov, pre ktorých je prístup k systému CELCAT žiadaný a nevyhnutný.

Dáta sa do systému dajú buď importovať alebo vytvárať ručne. Medzi najhlavnejšie možnosti systému patria:

- Vytvorenie nového / Otvorenie existujúceho rozvrhu
- Tlač / zdieľanie rozvrhu / automatické zverejnenie rozvrhu

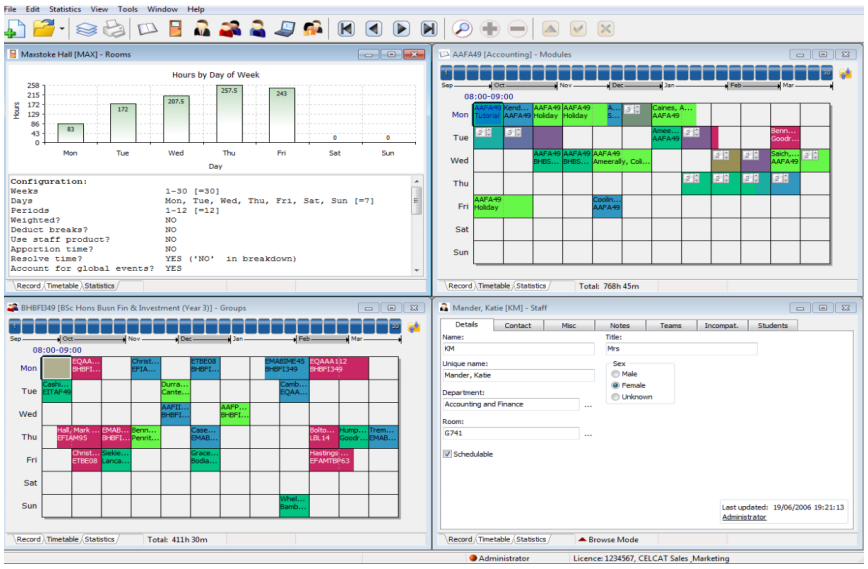
[1] `https://www.celcat.com/`

- Možnosť spravovania zdrojov
- Možnosť spravovania udalostí
- Klasifikácia zdrojov a udalostí
- Pokročilé funkcie konfigurácie (dĺžka vyučovacích hodín, individuálne podujatia a ďalšie)
- Konfigurácia miestností (počet sedadiel, vybavenie)
- Náhľad kalendára udalostí
- Detekcia kolízií a rôzne úrovne ich riešenia
- Poradca dostupných zdrojov pre udalosť
- Podpora štandardných a externých štúdií
- Štatistiky

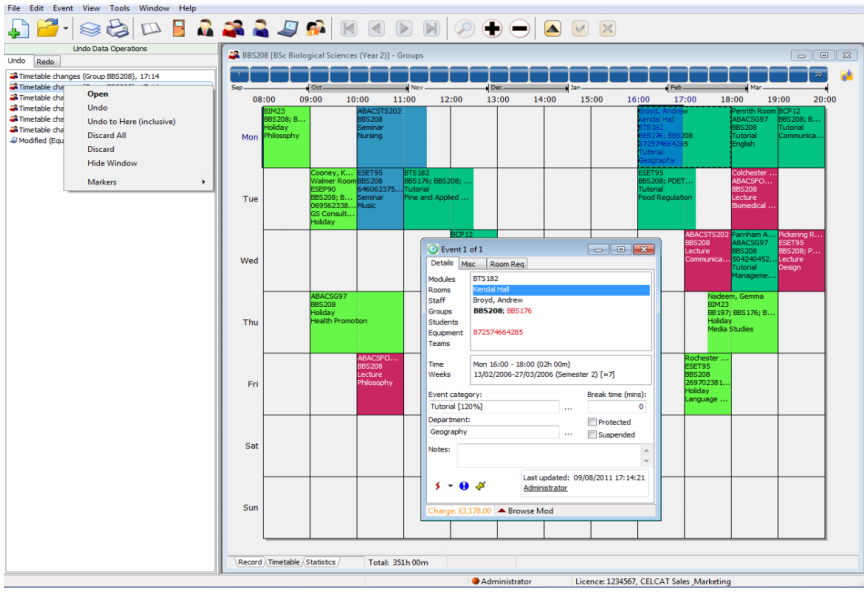
V prípade systému CELCAT treba spomenúť, že sa jedná o komerčný softvér. Rozhranie je už pomerne zastaralé, s čím úzko súvisí aj celý používateľský zážitok pri používaní systému. Niektoré bežné príkazy nie sú ľahko prístupné a vyžadujú si veľa klikov. Ďalším negatívom je fakt, že systém podporuje iba prácu na 4 oknách súčasne.

Podrobnejšie informácie o možnostiach celého systému sú dostupné v dokumente používateľskej príručky CELCAT[2].

[2] `http://www3.imperial.ac.uk/pls/portallive/docs/1/10225698.PDF`



Obrázok 3: Náhľad používateľského rozhrania systému CELCAT [4]



Obrázok 4: Náhľad kalendára udalostí systému CELCAT [4]

1.2.2 aSc TimeTables

Asc TimeTables[3] je rozvrhový systém, ktorý je primárne zameraný na stredné školy, pretože je založený na systéme s 8 obdobiami. Používateľ zadá dni, predmety, učiteľov a učebne, ktoré sa majú zahrnúť do rozvrhu. Po zadaní všetkých informácií Asc TimeTables vytvorí výsledný rozvrh, ktorý je založený na nasledujúcich kritériách [5]:

- Špecifikácia typu predmetu

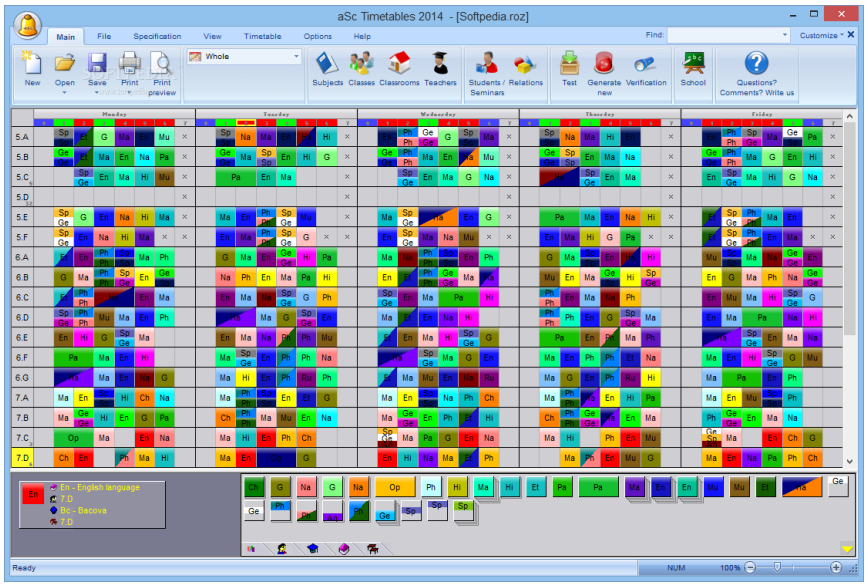
[3] `https://www.asctimetables.com/`

- Použitie pravidla proporcionálneho rozloženia hodín počas celého týždňa
- Okná učiteľov
- Špecifikácia maximálneho počtu okien učiteľov
- Špecifikácia maximálneho počtu okien učiteľov za deň
- Možnosť generovania nultých hodín
- Možnosť príchodu učiteľa na druhú lekciu alebo dokonca špecifikovať vyučovací blok manuálne
- Použitie pravidla pre celé a rozdelené formy vyučovacích hodín
- Pridelenie hodín do jednotlivých tried
- Zamknutie hodín v určených pozíciách
- Stanovenie úrovne komplexnosti generovania rozvrhu
- Počet kariet na otáznych pozíciach v rozvrhu
- Vzťahy všetkých kariet

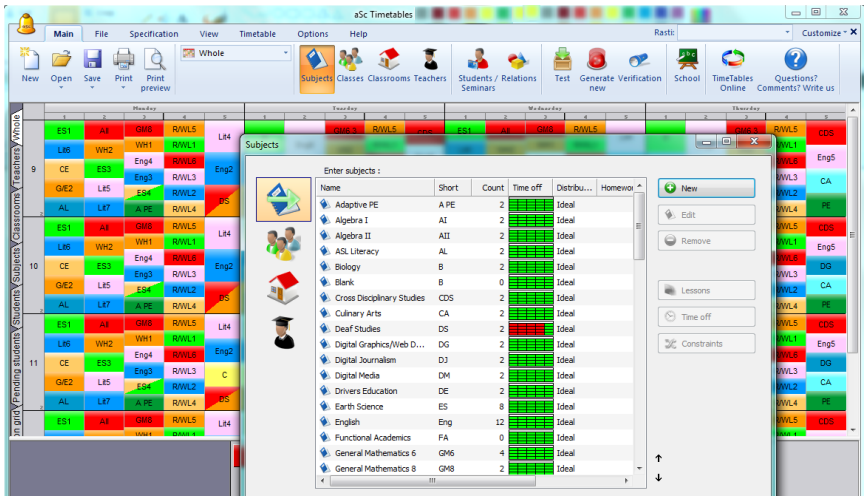
Medzi kladné stránky systému patrí to, že má jednoduché ovládanie, nakoľko program používa štandardné MS WindowsTM rozhranie. Asc TimeTables je navrhnutý pre efektívne zadávanie a kontrolu dát. Okrem svojej jednoduchosti poskytuje Asc TimeTables aj skvelú dokumentáciu, ktorá obsahuje sekciu pre testovanie. Systému však chýba lepšie zabezpečenie, modul rozsiahlejšej administrácie a možnosť zadávania vstupov a preferencií od študentov.

Bližšie špecifiká možností tohto systému sú dostupné v oficiálnej dokumentácii[4] Asc TimeTables systému.

[4] `help.asctimetables.com/pdf/asc_timetables_en_P1.pdf/`



Obrázok 5: Náhľad používateľského rozhrania systému aSc TimeTables [5]



Obrázok 6: Zobrazenie zoznamu predmetov v systéme aSc TimeTables [5]

1.2.3 Mimosa

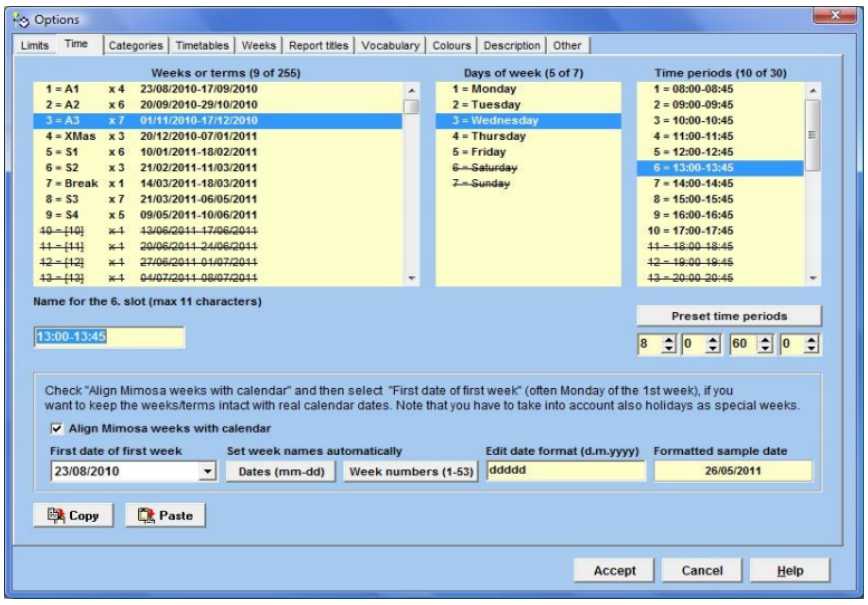
Rozvrhový systém Mimosa je desktopová aplikácia pre OS Windows, napísaná v jazyku Borland Delphi, ktorá sa používa na plánovanie udalostí na stredných školách, univerzitách, ale aj vo firmách. Program používa tabuľky na zadanie informácií od používateľa. Softvér umožňuje používateľovi zadať kurz, inštruktora a čas, aby zistil, ako vytvoriť rozvrhový plán. Tvorbu rozvrhu je možné vykonať manuálne, ale v prípade potreby aj automaticky prípadne kombinovaným spôsobom. Na prácu s dátami v podobe importu a exportu bola implementovaná podpora pre formáty CSV, iCalendar, vCalen-

dar. Mimosa má aj vlastný formát dát obdobného názvu [6]. Systém Mimosa používa svoju vlastnú databázu.

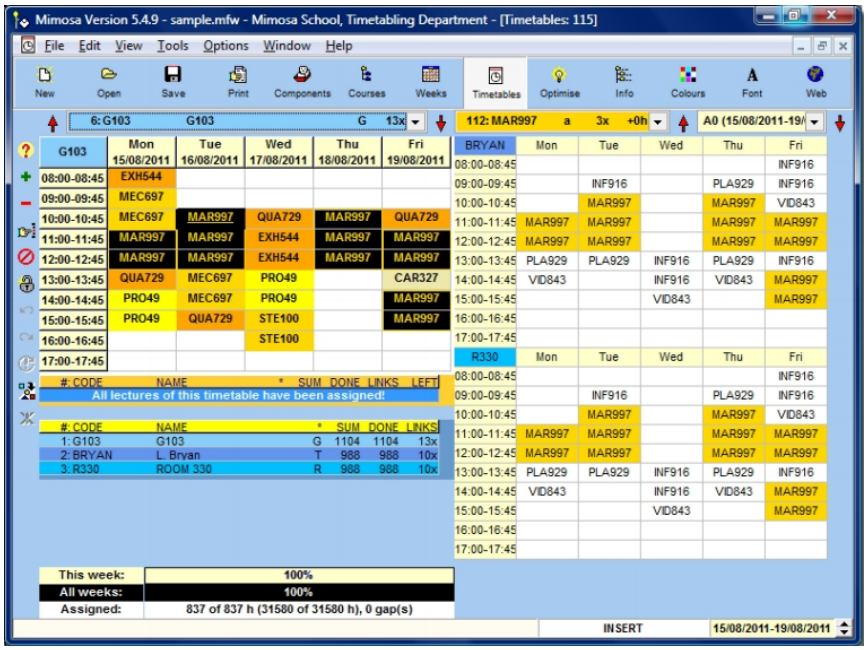
Medzi hlavné výhody systému patria:

- Rozsiahla funkcionalita
- Automatická kontrola kolízií
- Napodobňovanie predchádzajúcich riešení a výber miestností
- Stromové zobrazenie prehliadania udalostí
- Funkcia Drag&Drop
- Možnosť manuálnej úpravy rozvrhu
- Kontextová pomoc
- Podrobná dokumentácia

Nevýhodou je podpora iba pre jednu platformu a platená komerčná licencia potrebná na používanie plnej verzie systému.



Obrázok 7: Zobrazenie nastavení v systéme Mimosa [6]



Obrázok 8: Zobrazenie rozvrhu v systéme Mimosa [6]

2 Cieľ práce

Cieľom našej práce je oboznámiť sa s predchádzajúcimi prácami [7] a [8] a na základe existujúcej implementácie rozšíriť systém o moderné a prehľadné používateľské rozhranie. V predchádzajúcej práci [8] bol primárnym cieľom skôr vývoj serverovej časti systému, pričom používateľské rozhranie bolo iba prototypom napísaným vo funkcionálnom jazyku Elm[5]. My sme sa rozhodli pre vývoj používateľského rozhrania v novej technológií Angular 5, ktorá v spojení s ďalšími nástrojmi umožňuje tvoriť moderné aplikácie ako z hľadiska funkcionality, tak aj z hľadiska dizajnu.

2.1 Špecifikácia požiadaviek

Autori oboch predchádzajúcich prác pomerne jasne špecifikovali požiadavky kladené na celý rozvrhový systém. Vo všeobecnosti má systém umožňovať:

- Spravovanie a upravovanie používateľských oprávnení.
- Pracovanie s dátami, ich import a export dát v spolupráci so systémom AIS
- Zálohovanie dát a možnosť vytvárania verzií rozvrhov

[5] `http://elm-lang.org/`

- Vkladanie jednotlivých požiadaviek a obmedzení vplývajúcich na tvorbu výsledného rozvrhu
- Konfigurovanie systémových parametrov ovplyvňujúcich výsledky generátora rozvrhov, ale aj vzhľad a správanie používateľského rozhrania
- Automatizované generovanie rozvrhov
- Manuálne tvorenie rozvrhov a úpravy v nich pomocou intuitívneho prostredia a nástrojov Drag&Drop s funkciou sledovania kolízií
- Rezervovanie miestností pre nerozvrhové udalosti
- Navrhovanie vylepšení a iných zmien v rozvrhu jeho vlastnými používateľmi

Požiadavky aj naďalej zostávajú rovnaké, avšak naša práca sa zameriava skôr konkrétne na oblasť používateľského rozhrania a požiadaviek kladených na túto časť systému. V tejto oblasti sú kladené ako hlavné tieto požiadavky:

- Moderný dizajn aplikácie s použitím najnovších technológií
- Prehľadné a intuitívne používateľské rozhranie
- Responzívne správanie aplikácie zohľadňujúce rozlíšenie zariadenia
- Jednoduchá navigácia medzi modulmi systému bez zbytočných preklikov do hĺbky

2.2 Tvorba rozvrhov na FEI

Na fakulte FEI sa proces tvorenia rozvrhov rozdeľuje do dvoch fáz osobitne. Jednou fázou je tvorba rozvrhu na semester a tou druhou je fáza tvorby rozvrhu na skúškové obdobie. Každú z týchto fáz má na starosti iný rozvrhár. Aktuálny stav procesu tvorenia rozvrhov sme konzultovali s rozvrhárom Mgr. Dávidom Panczom, PhD. Existujú dva programy na tvorbu rozvrhov, ktoré majú rozvrhári FEI k dispozícii. Novším z nich je program Roger, ktorý však generuje horšie výsledky a disponuje obmedzenejšou funkcionalitou manuálnych úprav vygenerovaných rozvrhov než starší program WinRozvrhy. Preto sa na tvorbu rozvrhov FEI používa tento starší program. Vo fáze skúškového obdobia sa rozvrhy vytvárajú pomocou súboru skriptov na import dát a v MS Excel sa manuálne vykonáva čiastočná kontrola kolízií. Samotný proces začína súborom dát z AIS, ktorý je podkladom pre vstupné parametre pri generovaní. Tieto dáta obsahujú informácie o zozname miestností, predmetov, učiteľov a zoznamy daľších elementov a vzťahov medzi nimi. Ďalšími

dátami pre rozvrhárov, ktoré vstupujú do celého procesu, sú jednotlivé požiadavky vyučujúcich. Väčšinu týchto obmedzení treba zohľadňovať individuálne a zaznamenať ich do systému manuálnymi úpravami. Výsledný rozvrh zohľadňuje aj predchádzajúce rozvrhy z daných období. Posledným krokom procesu je importovanie výsledného rozvrhu do AIS, kde je sprístupnený koncovým používateľom či už na prezeranie alebo prihlasovanie na rozvrhové akcie v prípade študentov a taktiež následné zvrejnenie na stránke fakulty.

Zo samotného popisu celého procesu ako aj z konzultácií s rozvrhárom vyplýva, že aktuálny stav tvorenia rozvrhov je skutočne časovo náročný proces a potreba manuálnych úprav v neefektívnom rozhraní rozvrhárom komplikuje prácu. Manuálnym zásahom sa však kvôli individuálnym požiadavkám a obmedzeniam nedá úplne predísť, a preto sa budeme snažiť celé rozhranie systému orientovať hlavne na jednoduchosť a intuitívnosť používania a jednotlivé operácie orientovať na časovú úsporu a efektívnosť práce v ňom.

3 Návrh

V nasledujúcej kapitole popisujeme návrh nových častí systému ako aj potrebné použité existujúce časti z predošlých prác riešiacich tento rozvrhový systém. Priblížime si nami navrhovanú architektúru systému a prejdeme ku samotnému náčrtu dizajnu používateľského rozhrania.

3.1 Architektúra systému

Medzi požiadavkami kladenými na systém boli jednoduchosť a prehľadnosť používania, moderná architektúra a dizajn a v neposlednom rade aj platformová nezávislosť. Najvhodnejšou voľbou pre takto definovaný systém je navrhnúť ho vo forme webovej jednostránkovej aplikácie (z angl. Single-page Application). Z hľadiska architektúry sa jedná o klient-server aplikačný model.

Klient komunikuje vo formáte JSON prostredníctvom REST API, ktoré mu poskytuje server cez HTTP (resp. HTTPS). Model klientskej časti pokryjeme modernou Redux architektúrou, ktorá je momentálne na vzostupe a je vhodná pre systémy so zložitejšími a rozsiahlejšími úpravami globálneho stavu aplikácie.

Serverovú časť rozvrhového systému zabezpečuje webová služba poskytujúca REST API a podľa potreby dáta vyberá z alebo ukladá do PostgreSQL databázy. Architektúru serverovej časti sme sa rozhodli použiť z predchádzajúcej práce [8].

3.1.1 Jednostránková aplikácia

Vývoj webových aplikácií bol vždy významnou súčasťou vývoja softvéru. V interakcii s komplexnými webovými aplikáciami však bolo veľa obmedzení, čo spôsobilo, že nie sú tak responzívne ako desktopové aplikácie.

Prvé webové aplikácie boli jednoduché statické stránky vytvorené iba s HTML a CSS. Tieto statické stránky boli pomalé, pretože každá stránka, teda všetky HTML a CSS, museli byť stiahnuté zo servera vždy, keď používateľ klikol na odkaz. Všetko sa zmenilo s príchodom AJAX, a teda asynchrónneho JavaScript-u a XML. AJAX je súbor techník, ktoré umožnili web aplikácii rýchle inkrementálne aktualizácie používateľského rozhrania bez opätovného načítania celej stránky prehliadača. V porovnaní s čisto statickými webovými stránkami, ktoré si používateľ musel stiahnuť pri každom kliknutí na odkaz, boli webové stránky používajúce AJAX mnohokrát rýchlejšie.

Medzičasom sa vyvíjali a vylepšovali nové verzie tradičných technológií používaných na vývoj webových aplikácií ako HTML, CSS a JS, čo umožňovalo vytvoriť novú generáciu flexibilných webových jednostránkových aplikácií. Tento nový typ webovej aplikácie je

z hľadiska interaktivity porovnateľný s desktopovými aplikáciami, a zároveň nevyžaduje žiadne ďalšie doplnky na fungovanie. Jediná požiadavka, ktorá by mala byť ideálne splnená, je práca s najnovšiou verziou webového prehliadača. Toto môže byť považované za nevýhodu týchto aplikácií, pretože podpora JavaScript-u je neustále rozsiahlym problémom. Stále existuje veľa používateľov, ktorí nie vždy aktualizujú svoje webové prehliadače na najnovšiu verziu alebo dokonca úmyselne vypnú JS z bezpečnostných dôvodov alebo z dôvodu ochrany osobných údajov.

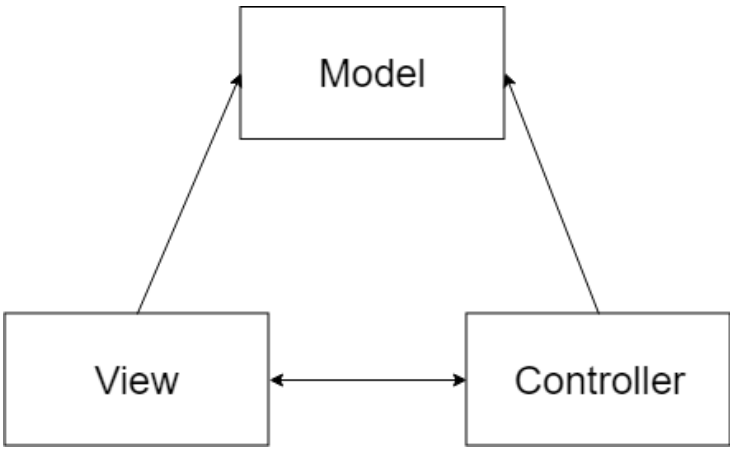
Jednostránkové aplikácie pokračovali v raste a objavovala sa potreba lepšej štruktúry týchto aplikácií. Vývojári začali vytvárať komponenty na lepšiu organizáciu kódu aplikácií. Jedna z prvých knižníc na báze komponentov bola React[6], ktorú vytvoril Facebook. Keďže React bol len knižnicou na vykresľovanie dát, vznikala bočná potreba spravovania údajov vo veľkých aplikáciách. Pre tento problém s riadením údajov bol predstavený architektonický vzor nazývaný Flux. Po jeho uvoľnení začali mnohé vývojárske tímy realizovať svoje vlastné implementácie Flux-u.

3.1.2 Redux

Existuje veľa rôznych architektonických vzorov, ktoré sa dajú použiť pri vývoji webových aplikácií. MVC (Model-View-Controller) je veľmi populárny, široko podporovaný vzor, používaný už niekoľko desiatok rokov. Model uchováva údaje potrebné pre aplikáciu. Zobrazenie je nevyhnutné na zobrazenie dát používateľovi. Riadiaci systém funguje ako lepidlo medzi modelmi a zobrazeniami. Vyžaduje údaje z modelu a počúva na udalosť v zobrazení [9]. Treba poznamenať, že existuje veľa MVC podobných architektúr, MVP (Model-ViewPresenter), MVVM (Model-View-ViewModel) alebo HMVC (Hierarchický Model-View-Controller), ktoré sú buď variáciou alebo odvodením vzoru MVC.

[6] `https://reactjs.org/`

Najjednoduchšia verzia MVC je znázornená na obrázku nižšie:



Obrázok 9: Architektúra MVC

Manažment stavu aplikácie bol vo frontendových frameworkoch vždy neustálym problémom. Základná myšlienka mutácie stavov na viacerých úrovniach pri frontendových frameworkoch robí MVC prístup neefektívny. Jeden z hlavných rozdielov medzi Redux[7] a MVC je tok dát. MVC umožňuje obojsmerný dátový tok, zatiaľ čo Redux používa jednosmerný dátový tok. Napríklad v MVC môžu dáta prúdiť z *View* do *Controller* a naspäť, zatiaľ čo pri Redux-e musí byť zmena vykonaná prostredníctvom vyvolania akcie.

Stavová mutácia bola zrejmá už v Angular 1 (AngularJS[8]), kde bola logika riadenia stavu aplikácie rozdelená medzi direktívy, kontrolery a službu, pričom každá úroveň mala svoju vlastnú logiku pre mutáciu stavu. Tým sa segmentovaný stav celej aplikácie stal náchylný na nekonzistentnosť a takmer nemožný na otestovanie. Tieto problémy sa stávali čoraz zjavnejšími a ťažšie sa im predchádzalo, keďže vývoj aplikácií napredoval a ich zložitosť časom narastala. Frontendové aplikácie sa začali stávať čoraz komplexnejšími a reaktívnejšími voči používateľským vstupom. Avšak čoraz populárnejšie znaky vývoja frontendu, ako napr. dôraz na funkcionálne programovanie, podporili vznik nového prístupu manažovania stavu aplikácie.

Hlavnou motiváciou pre Redux bola predvídateľnosť stavových mutácií a zníženie zložitosti spravovania údajov v aplikáciách pomocou jednosmerného toku údajov. Pretože v systéme Redux toky prechádzajú iba jedným smerom a stav aplikácie sa nadhádza na jednom mieste, celý systém sa stáva viac predvídateľným. Redux tiež používa nemeniteľné spravovanie údajov, ktoré spôsobuje menej vedľajších účinkov v toku dát, čo vedie k jednoduchšiemu programovaniu a ladeniu [10].

[7] `https://redux.js.org/`
[8] `https://angularjs.org/`

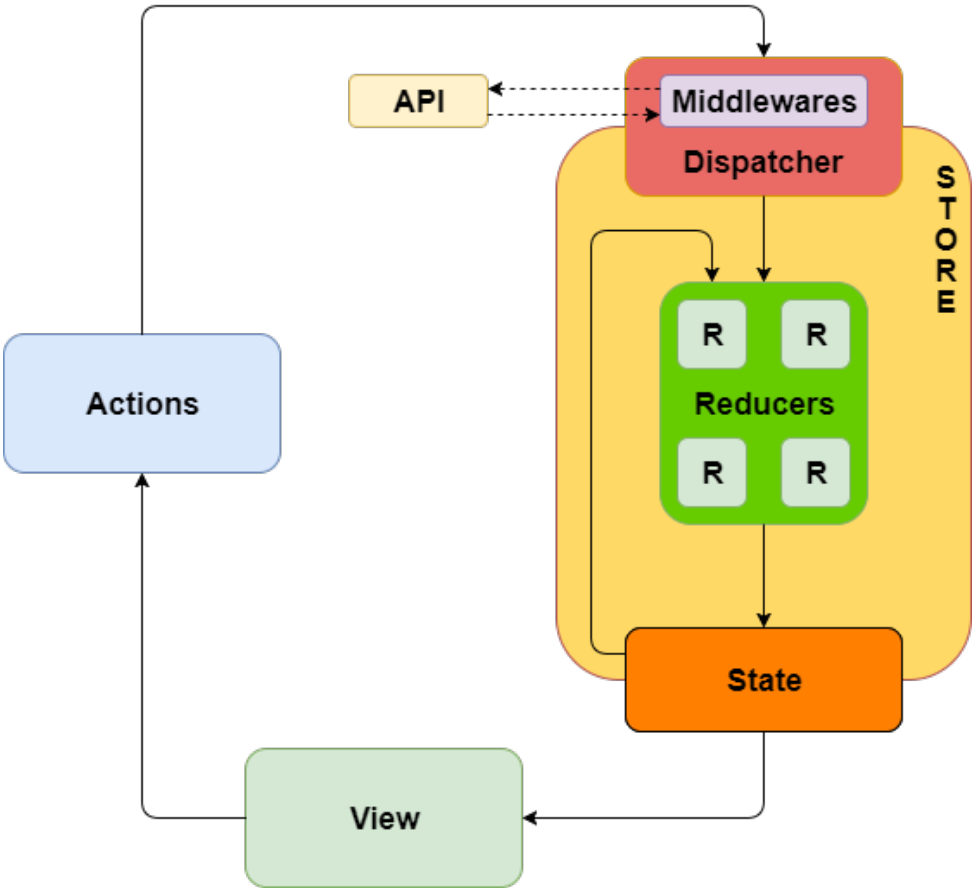
Redux sa skladá z troch hlavných častí. Prvou časťou sa nazýva úložisko (z angl. Store), v ktorom je uložený stav a každá časť údajov aplikácie. Je to čistý JS objekt, ktorý môže byť vytvorený viacerými spôsobmi. Úložisko v Redux-e je nemeniteľné. Ďalšou časťou sú akcie (z angl. Actions). Akcie sú taktiež jednoduché JS objekty, ktoré zabezpečujú informáciu o tom, čo sa má stať, ale neuvádzajú spôsob ani samotnú logiku, ako by sa to malo robiť. Akcie obsahujú údaje, ktoré sa odosielajú do úložiska. Poslednou časťou sú reduktory (z angl. Reducers), ktoré spájajú úložisko a akcie. Reduktory obsahujú logiku aplikácie a sú to jednoduché čisté funkcie (z angl. Pure functions), ktoré robia zmeny v úložisku. Reduktory nemenia existujúce údaje, ale vždy vrátia úplne nové údaje.

Redux teda sleduje tri základné princípy [11]:

- Jednosmerný tok údajov.
- Stav celej aplikácie je uložený v jedinom nemeniteľnom stavovom strome. Jediný spôsob, ako zmeniť stav, je vyvolať akciu.
- Zmeny v úložisku sa vykonávajú pomocou čistých funkcií (z angl. Pure functions). To znamená, že nemenia pôvodný stav, ale vracajú nové stavové objekty.

Pri dodržaní týchto zásad môžeme dosiahnuť predvídateľné a reprodukovateľné správanie v aplikácii.

Nasledujúci obrázok popisuje tok dát v architektúre Redux:



Obrázok 10: Redux - tok dát v procese [12]

Napriek tomu, že Redux vznikol prostredníctvom komunity React[9], knižnice tretích strán ako napríklad ngrx/store[10] a rozšírenia ako RxJS[11], urobili z Redux-u rovnako vhodný koncept na použitie v aplikáciách Angular. Redux je jednou z mnohých implementácií Flux-u[12]. Populárny sa stal najmä pre širokú komunitu a dobrú dokumentáciu.

Redux pomáha udržať kód ľahšie čitateľný, udržiavateľný a rozrastajúci. Redux je tiež škálovateľný. Existuje jednoduchý a jasný spôsob, ako rozšíriť aplikáciu, ktorá používa Redux. Napríklad akcie (z angl. Actions) môžu byť uložené v samostatnom súbore, takže nie sú roztrúsené naprieč celým kódom aplikácie. Tiež reduktory (z angl. Reducers) môžu byť uložené na jednom mieste pre jasnú štruktúru. Keď sa v Redux-e pridáva nová funkcia, musí byť vytvorená akcia. Potom musí byť do reduktora pridaná funkcia, ktorá vykoná skutočnú zmenu v úložisku (z angl. Store). Redux má aj svoje nevýhody, pretože

[9] `https://reactjs.org/`
[10] `https://github.com/ngrx/store`
[11] `http://reactivex.io/rxjs/`
[12] `https://facebook.github.io/flux/`

je spojený s písaním väčšieho počtu opakovaného kódu ako pri štandardných architektúrach. Taktiež je relatívne ťažké migrovať existujúci projekt na používanie systému Redux, pretože používa radikálne odlišný prístup na riadenie stavu aplikácie a toku údajov.

3.1.3 REST

Reprezentatívny stavový prevod (REST) je architektonický štýl, ktorý definuje súbor obmedzení a vlastností založených na protokole HTTP. Webové služby, ktoré zodpovedajú architektonickému štýlu REST alebo RESTful webové služby, zabezpečujú interoperabilitu medzi počítačovými systémami na internete. Webové služby kompatibilné s REST umožňujú žiadajúcim systémom prístup a manipuláciu s textovými reprezentáciami webových zdrojov pomocou jednotnej a vopred definovanej množiny bezstavových operácií. Iné druhy webových služieb, ako sú webové služby SOAP (Simple Object Access Protocol), vystavujú svoje vlastné sady operácií, čim sa stávajú robustnejšie. REST využíva menšiu šírku pásma, čo ho robí vhodnejším na internetové použitie v porovnaní s technológiou SOAP.

Vo webovej službe RESTful, môžu byť odpovede na dotazy kladené na URI zdrojov vo formáte XML, HTML, JSON alebo v inom formáte. Odpoveď môže potvrdiť, že došlo k nejakej zmene uloženého prostriedku a odpoveď môže poskytnúť hypertextové odkazy na iné súvisiace zdroje alebo zbierky zdrojov. Pri komunikácii najčastejšie prostredníctvom HTTP protokolu sú k dispozícii operácie GET, POST, PUT, DELETE a ďalšie preddefinované metódy CRUD HTTP.

Pomocou bezstavových protokolov a štandardných operácií sa systémy REST zameriavajú na rýchly výkon, spoľahlivosť a schopnosť rozširovania tým, že opätovne používajú komponenty, ktoré je možné spravovať a aktualizovať bez toho, aby to ovplyvnilo systém ako celok, a to aj počas jeho prevádzky [13].

3.1.4 Autentifikácia

Autentifikácia je proces rozpoznávania identity používateľa. Je to mechanizmus, ktorý spája prichádzajúcu požiadavku so súborom identifikačných poverení. Poskytnuté poverenia sa porovnávajú s údajmi o používateľských oprávneniach uložených v databáze v lokálnom operačnom systéme alebo v autentizačnom serveri. Proces overovania vždy beží na začiatku spustenia aplikácie ešte predtým, ako dôjde ku kontrole a povereniu oprávnení a skôr, než budú povolené ďalšie operácie v aplikácii. Rôzne systémy môžu vyžadovať rôzne typy poverení na zistenie totožnosti používateľa. Identifikačné poverenie často nadobúda formu hesla, ktoré je tajné a známe iba jednotlivcovi a systému.

Proces pozostáva z jednoduchých krokov:

- Používateľ sa pokúša pripojiť k webovým službám.
- Webové služby požiadali používateľa o poverenie (informácie o totožnosti).
- Používateľ poskytuje poverenia.
- Webové služby overujú totožnosť používateľa overovaním zadaných poverení a zodpovedajúcim spôsobom odpovedajú.

Proces overovania je možné opísať v dvoch odlišných fázach - identifikácia a skutočná autentifikácia. Identifikačná fáza poskytuje používateľovi identitu s bezpečnostným systémom. Táto identita je poskytovaná vo forme ID používateľa. Bezpečnostný systém vyhľadá všetky abstraktné objekty, ktoré pozná, a nájde konkrétnu, ktorú aktuálny používateľ momentálne používa. Akonáhle sa to stane, používateľ bol identifikovaný. Skutočnosť, ktorú používateľ tvrdí, však nevyhnutne nemusí byť pravdivá. Skutočný používateľ môže byť priradený k inému abstraktnému používateľskému objektu v systéme, a preto mu musia byť udelené práva a povolenia a používateľ musí systému poskytnúť dôkazy preukazujúce jeho totožnosť. Proces určenia nároku na totožnosť používateľa prostredníctvom kontroly dôkazov poskytnutých používateľmi sa nazýva autentifikácia a dôkaz, ktorý poskytuje používateľ počas procesu autentifikácie, sa nazýva identifikačné poverenie.

REST framework poskytuje niekoľko schém autentifikácie a tiež umožňuje implementovať vlastné schémy. Schémy overovania sú vždy definované ako zoznam tried. REST sa pokúsi autentifikovať s každou triedou v zozname a nastaví *request.user* a *request.auth* pomocou návratovej hodnoty prvej úspešne overenej triedy. Ak sa žiadna trieda neoverí, *request.user* bude nastavené na inštanciu *django.contrib.auth.models.AnonymousUser* a *request.auth* bude nastavené na *None*.

Hodnotu *request.user* a *request.auth* pre neoverené požiadavky je možné upraviť pomocou nastavení *UNAUTHENTICATED_USER* a *UNAUTHENTICATED_TOKEN*.

Nižšie si popíšeme niekoľko bežných autentifikačných schém používaných pri REST webových službách.

Basic Authentication schéma používa základnú autentifikáciu HTTP, podpísanú používateľským menom a heslom v dotaze. Je to najjednoduchší spôsob implementácie overovania. V tejto schéme sa informácie o používateľskej totožnosti, tj poverenia, odosielajú v zakódovanom formáte *base64*. Získaná zakódovaná hodnota *base64* sa odošle pomocou HTTP hlavičky *Authorization* [14]. Ak sa úspešne overí, *Basic Authentication* poskytuje nasledujúce poverenia.

- *request.user* bude Django inštancia typu *User*.
- *request.auth* bude *None*.

Neoverené odpovede, ktorým sa odmietne povolenie, spôsobia neoprávnenú odpoveď *HTTP 401 Unauthorized* s príslušnou hlavičkou *WWW-Authenticate*.

Problém s touto schémou overovania je, že používateľské meno a heslo sú zakódované a nie šifrované, čiže sa dajú ľahko dekódovať. V dôsledku tohto problému by nemala byť základná schéma overenia implementovaná tam, kde komunikácia prebieha cez HTTP a nie cez HTTPS. Základné overenie je vo všeobecnosti vhodné iba na testovanie.

Odosielané poverenia pri schéme *Basic Authentication* môžu napríklad vyzerať nasledovne:

```
GET /api/v1/gotham/ HTTP/1.1
Host: payatu.com
Authorization: Basic YmF0bWFuOmJhdG1hbkAxMjM=
```

Obrázok 11: Príklad poverenia pri schéme Basic Authentication [15]

HMAC Authentication schéma, pri ktorej namiesto odosielania hesla v zakódovanej podobe posiela klient hash hodnotu hesla s inými informáciami. Iné informácie vo všeobecnosti pozostávajú zo slovies (operácií) HTTP, z URL, časovej pečiatky, hashu v tele správy alebo náhodného čísla. Vhodným riešením je použiť hodnotu hash správy tela pri konštrukcii HMAC hashu, pretože zabezpečí integritu odosielaných údajov.

Ak napríklad používateľ *batman* pristupuje k prostriedku *gotham*, potom výpočet HMAC môže vyzerať nasledovne:

```
hash_value = base64encode(hmac('sha256', 'password', 'GET+/api/v1/gotham'))
```

```
GET /api/v1/gotham/ HTTP/1.1
Host: payatu.com
Authorization: hmac batman:hashvalue
```

Obrázok 12: Príklad poverenia pri schéme HMAC [15]

Token Authentication schéma používa jednoduchú schému *Authentication HTTP* založenú na tokenoch. Autentifikácia prostredníctvom tokenu je vhodná pre nastavenia klient-server, ako sú natívni desktopoví a mobilní klienti. Reťazec je podpísaný serverom a klient ho získa po požiadaní o jeho vytvorenie ako odpoveď. Sever si však nemusí nič

pamätať a všetky potrebné informácie sú uložené a kryptograficky zakódované v tokene. Jednou z vlastností tokenu je aj jeho ohraničená platnosť.

Ak chceme použiť schému *Token Authentication*, musíme nakonfigurovať triedy autentifikácie tak, aby zahŕňali *TokenAuthentication* a navyše obsahovali *rest_framework.authtoken* v nastavení *INSTALLED_APPS*.

Odosielané poverenia pei schéme Token Authentication môžu napríklad vyzerať nasledovne:

```
GET /api/v1/gotham/ HTTP/1.1
Host: payatu.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:56.0) Gecko/20100101 Firefox/56.0
Accept: application/json
Accept-Language: null
Accept-Encoding: gzip, deflate
Authorization: Bearer GjQcs9OiCb7tsuAVBbiYfP3SuypGKZ
Content-Type: application/json
Connection: close
```

Obrázok 13: Príklad poverenia pri schéme Token Authentication [15]

V našom systéme je na REST API autentifikáciu používaná schéma *Token Authentication*. Tá udržiava bezstavovost servera a je vhodná pre nastavenia klient-server pre desktopových aj mobilných klienov. Autor predchádzajúcej práce [8] sa rozhodol konkrétne pre štandard JWT (JSON Web Token).

3.1.5 JSON Web Token

JSON Web Token je otvorený štandard založený na formáte JSON pre vytváranie prístupových tokenov, ktoré obsahujú istý počet tvrdení. Napríklad server by mohol vygenerovať token, ktorý obsahuje tvrdenie *"prihlásený ako admin"* a poskytne ho klientovi. Klient by potom mohol použiť token, aby dokázal, že je prihlásený ako admin. Tokeny sú podpísané súkromným kľúčom jednej strany (zvyčajne serverom), takže obidve strany (ostatné, ktoré už sú vhodným a dôveryhodným spôsobom v držbe príslušného verejného kľúča) dokážu overiť, že token je legitímny. Tokeny sú navrhnuté tak, aby boli kompaktné, bezpečné pre adresy URL a použiteľné najmä v kontexte jednotného prihlásenia webového prehliadača (SSO). Tvrdenia JWT sa môžu zvyčajne používať na odovzdávanie totožnosti overených používateľov medzi poskytovateľom totožnosti a poskytovateľom služieb alebo akýchkoľvek iných typov tvrdení, vyžadovaných obchodnými procesmi. JWT majú zvyčajne tri časti a to hlavičku, užitočné dáta a podpis [16].

Hlavička identifikuje, ktorý algoritmus sa používa na generovanie podpisu, a môže vyzerať takto [17]:

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

HS256 označuje, že tento token je podpísaný pomocou *HMAC-SHA256*.

Užitočné dáta obsahujú tvrdenia a metadáta. Existujú tri typy tvrdení a to registrované, verejné a súkromné tvrdenia.

- Registrované tvrdenia: Ide o súbor odporúčaných preddefinovaných tvrdení, ktoré nie sú povinné. Sem patria napríklad tvrdenia iss (vydavateľ), exp (čas expirácie), sub (predmet), aud (publikum) a ďalšie.
- Verejné tvrdenia: Môžu byť ľubovolne definované používateľmi JWT. Aby sa však zabránilo kolíziám, mali by byť definované v registri IANA JSON alebo definované ako URI, ktoré obsahuje menný priestor odolný voči kolíziam.
- Súkromné tvrdenia: Ide o vlastné tvrdenia vytvorené na zdieľanie informácií medzi stranami, ktoré súhlasia s ich použitím a nepatria medzi registrované ani verejné tvrdenia.

Objekt užitočných dát môže vyzerať nasledovne:

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

Podpis je vypočítaný pomocou algoritmu deklarovaného v hlavičke, ktorý spojí tajomstvo so zakódovanou hlavičkou a užitočnými dátami, ktoré sú oddelené bodkou. Ak napríklad chceme použiť algoritmus *HMAC SHA256*, podpis sa vytvorí nasledujúcim spôsobom:

```
HMACSHA256(
```

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

Podpis sa používa na overenie toho, že správa nebola počas prenosu zmenená a v prípade tokenov podpísaných so súkromným kľúčom môže tiež overiť, že odosielateľ JWT je ten, kým tvrdí že je.

Nasledujúci text zobrazuje JWT, ktorý má zakódovanú predchádzajúcu hlavičku a užitočné dáta a je podpísaný s tajomstvom.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.
TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ
```

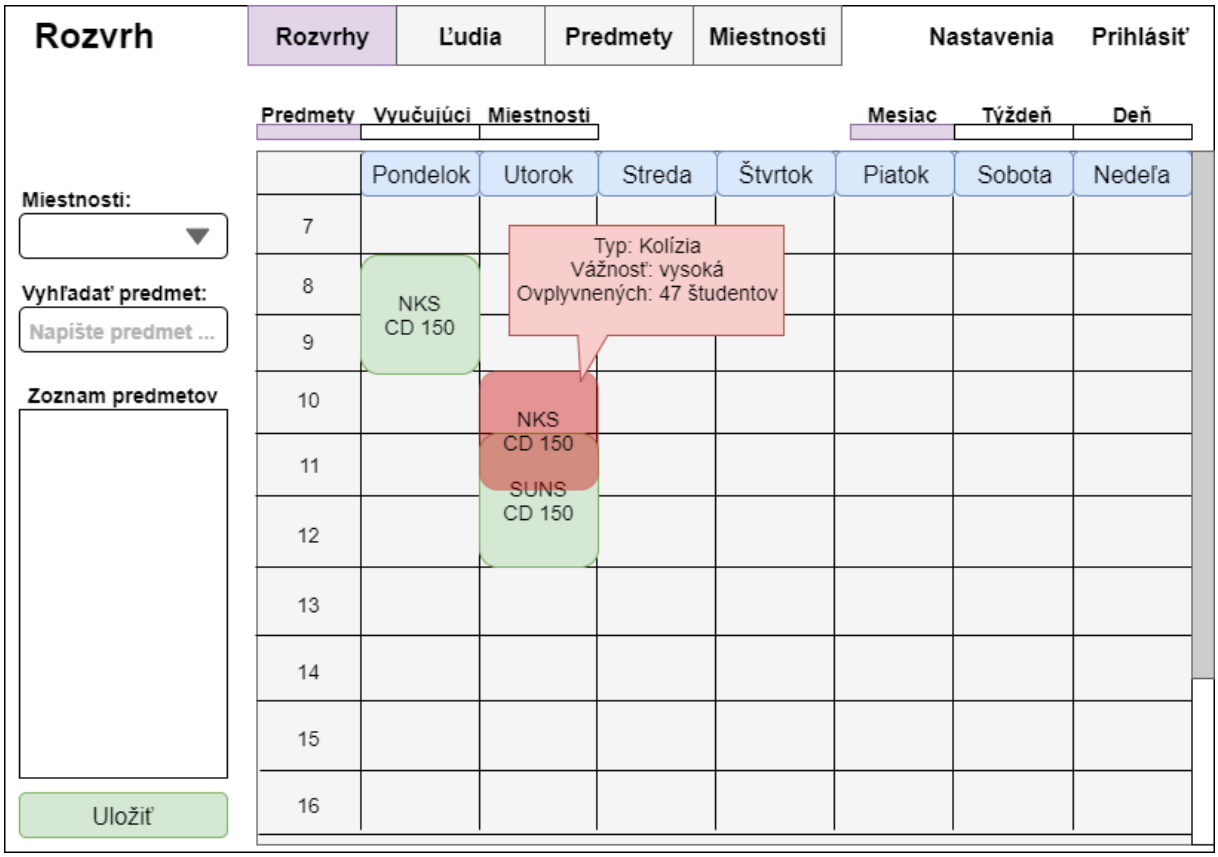
Bližšie informácie o konkrétnej implementácii JWT v našom systéme sú popísané v predchádzajúcej diplomovej práci [8].

3.2 Návrh používateľského rozhrania

Najdôležitejšou časťou používateľského rozhrania je kalendár udalostí. Do kalendára sa dajú vkladať nové udalosti, upravovať a prípadne mazať už existujúce. Kalendár podporuje Drag&Drop funkcionalitu, pomocou ktorej sa dajú jednotlivé udalosti v kalendári presúvať na iné miesta časovej mriežky a takisto podporuje funkcionalitu rozširovania časového rozsahu existujúcej udalosti ťahaním jej okrajov. Klient umožňuje zobrazenie časovej mriežky z viacerých náhľadov, a to na dňovej, týždňovej a mesačnej úrovni. Inšpirovali sme sa riešeniami z vyššie popísaných existujúcich rozvrhových systémov 1.2 a z hľadiska dizajnu podľa Kalendáru Google[13] a aplikácie Vonigo[14]. Návrh rozhrania pre kalendár udalostí je načrtnutý na obrázku 14 nižšie.

[13] https://calendar.google.com

[14] https://dribbble.com/shots/1458422-Vonigo-Schedule/attachments/216307



Obrázok 14: Návrh rozhrania pre kalenár udalostí

Na základe špecifikovaných požiadaviek 2.1 je aplikácia prehľadná a práca s ňou nevyžaduje zbytočné prekliky na uskutočnenie jednoduchých operácií. Používateľské rozhranie je schopné prispôsobiť sa viacerým druhom zariadení a jeho správanie je plne responzívne zohľadňujúc rozlíšenie zariadenia a možnosti voľného priestoru na obrazovke. Menu celej aplikácie a navigácia medzi modulmi je vedená v duchu jednoduchosti na báze dizajnu najmodernejších webových aplikácií. Rozhranie je možné kofigurovať podľa preferencií používateľa, ktorý si môže napríklad zmeniť usporiadanie navigačných panelov menu alebo upraviť ich zafarbenie napríklad kvôli horším svetelným podmienkam okolia.

4 Použité technológie

Daná kapitola pojednáva o technológiach, ktoré sme použili pri implementácii návrhu nášho rozvrhového systému.

Program z predchádzajúcej diplomovej práce [8] je na strane klienta implementovaný vo funkcionálnom jazyku Elm. Momentálne je však tento jazyk v oblasti vývoja frontendu stále v experimentálnych fázach a nie je istota dlhodobej podpory v tejto oblasti. Našu aplikáciu má používať široké spektrum ľudí na univerzite a má ňou byť v budúcnosti využívaná, preto je dlhodobá podpora technológií veľmi dôležitým faktorom. Naša práca je primárne zameraná na používateľské rozhranie a hlavnou zmenou bude prechod na modernú technológiu Angular 5, ktorá ma širokú komunitu v oblasti vývoja frontendu a hlavne je v danej oblasti dlhodobo podporovaná. Hlavné komponenty rozhrania boli použité z webového frameworku Bootstrap[15]. My použijeme komponenty, ktoré sú natívne pre Angular 5 (Angular Material[16]). Jednotlivým technológiam použitých na strane klienta sa podrobnejšie venujeme nižšie v kapitolách 4.2.1 až 4.2.4.

Serverovú časť projektu sa budeme snažiť využiť v čo najväčšej miere z predchádzajúcich rokov a do jej implementácie neplánujeme nijak významnejšie zasahovať. Jednotlivé použité technológie na strane servera opisujeme nižšie v kapitolách 4.1.1 až 4.1.3.

Tabuľka 1 prehľadne popisuje navrhnuté zmeny použitých technológií:

Technológia	Pôvodná	Navrhnutá
Databáza	PostgreSQL	PostgreSQL
Server	Python 3.5 Django 1.11 (LTS)	Python 3.5 Django 1.11 (LTS)
Klient	Elm 0.18	**Angular 5 + TypeScript** **RxJS Observable**

Tabuľka 1: Navrhnuté zmeny použitých technológií

4.1 Serverová časť

Serverová časť systému je zabezpečovaná REST API dopytmi a dáta sa ukladajú do, prípadne vyberajú z PostgreSQL databázy.

[15] `https://getbootstrap.com/`

[16] `https://material.angular.io/`

4.1.1 Python

Python[17] je interpretovaný, objektovo orientovaný programovací jazyk na vysokej úrovni s dynamickou sémantikou. Jeho vysoko postavené dátové štruktúry v kombinácii s dynamickým písaním a dynamickým viazaním robia tento jazyk veľmi atraktívnym pre rýchly vývoj aplikácií ako aj pre použitie ako skriptovací jazyk alebo ako akýsi medzičlánok na prepojenie existujúcich komponentov dohromady. Jednoduchá a ľahko naučiteľná syntax jazyka Python zdôrazňuje čitateľnosť, a tým znižuje náklady na údržbu programu. Python podporuje moduly a balíky, čo zlepšuje modularitu programu a opätovné použitie kódu. Python prekladač a rozsiahla štandardná knižnica sú k dispozícii v zdrojovej alebo binárnej forme pre všetky hlavné platformy a môžu byť voľne distribuované [18].

Python 3 je nová verzia jazyka, ktorá je spätne nekompatibilná s verziou 2. Jazyk je väčšinou rovnaký, ale mnohé detaily, najmä to, ako fungujú vstavané objekty (slovníky a reťazce), sa značne zmenili a veľa zastaralých funkcií bolo finálne odstránených.

Oficiálna podpora Python 2 končí rokom 2020, a kvôli tomuto faktu sa v projekte využíva jeho novšia verzia Python 3 s dlhodobou podporou (z angl. LTS - Long Term Support).

4.1.2 Django

Django[18] je vysokoúrovňový a open-source Python webový framework, ktorý podporuje rýchly vývoj a čistý pragmatický dizajn. Hlavným cieľom Django je uľahčiť vytváranie komplexných databázových webových stránok. Django zdôrazňuje opätovnú použiteľnosť a zapájateľnosť komponentov, menej kódu, voľnejšie väzby, rýchly vývoj a princíp neopakovať sa. Python sa používa v celom rozsahu, a to aj pre súbory nastavení a dátové modely. Django tiež poskytuje voliteľné rozhranie pre CRUD operácie, ktoré je generované dynamicky prostredníctvom introspekcie a nakonfigurované prostredníctvom administrátorských modelov [19].

4.1.3 PostgreSQL

PostgreSQL[19] je open-source, výkonný objektovo-relačný databázový systém, ktorý používa a rozširuje jazyk SQL v kombinácii s mnohými funkciami, ktoré bezpečne ukladajú a škálujú najzložitejšie dátové úlohy. PostgreSQL získal silnú reputáciu pre svoju overenú architektúru, vzájomné prepojenie, integritu údajov, robustnú sadu funkcií a rozšíriteľnosť. PostgreSQL beží na všetkých hlavných operačných systémoch, je kompatibilný s ACID (Atomicity, Consistency, Isolation, Durability) od roku 2001 a má robustné do-

[17] `https://www.python.org/`
[18] `https://www.djangoproject.com/`
[19] `https://www.postgresql.org/`

plnky, ako napríklad populárne rozšírenie geografickej databázy PostGIS[20] [20].

4.2 Klientská časť

Klient je tvorený ako jednostránková aplikácia (z angl. Single-page Application), ktorá sa so serverom spája prostredníctvom REST API. Implementačným základom je jazyk TypeScript a jeho implementačný framework Angular 5. Jedná sa o najmodernejšiu dostupnú verziu (v čase písania tejto práce). Štýlovanie celého používateľského rozhrania je definované prostredníctvom SCSS štýlov.

4.2.1 TypeScript

TypeScript[21] je silne typový, objektovo orientovaný, kompilovateľný jazyk spolu so sadou nástrojov. Je nadstavbou jazyka JavaScript[22], môže opätovne použiť všetky jeho existujúce nástroje a knižnice a sám poskytuje širšiu radu možností. TypeScript prijíma základné stavebné prvky programu z jazyka JavaScript. Všetok kód napísaný v jazyku TypeScript sa pre účely vykonania prevedie na ekvivalentný kód v jazyku JavaScript. TS je kompatibilný naprieč prehliadačmi, zariadeniami a operačnými systémami. Môže byť spustený v ľubovoľnom prostredí, na ktorom beží JavaScript a nepotrebuje vyhradený virtuálny stroj alebo špecifické runtime prostredie na vykonanie [21].

Medzi prínosy jazyka TypeScript patria:

- Kompilácia - JavaScript je interpretovaný jazyk. Preto je pre zistenie jeho validnosti potrebné ho spustiť. Prekladač jazyka TS poskytuje funkciu kontroly chýb. TS kompiluje kód a generuje chyby pri kompilácii, čo pomáha nájsť chyby ešte pred samotným spustením skriptu.

- Silné statické typovanie - JavaScript nie je silne typový. TypeScript je dodávaný s voliteľným systémom statického typovania a zadávania typov pomocou TLS (Transport Layer Security). Typ premennej, deklarovanej bez svojho typu, dokáže TLS vyvodiť na základe jej hodnoty.

- TypeScript podporuje definície typov existujúcich knižníc jazyka JavaScript poskytovaných v súbore definícií (s príponou .d.ts).

- TypeScript podporuje objektovo orientované programovacie koncepty ako triedy, rozhrania, dedenie, atď.

[20] `https://postgis.net/`
[21] `https://www.typescriptlang.org/`
[22] `https://www.javascript.com/`

TypeScript má tieto tri komponenty:

- Jazyk - obsahuje syntax, kľúčové slová a anotácie typov.
- Kompilátor- TypeScript kompilátor (tsc) konvertuje inštrukcie napísané v jazyku TypeScript na ekvivalentný kód v jazyku JavaScript.
- Jazyková služba - odhaľuje ďalšiu vrstvu okolo hlavného potrubia kompilátora. Jazyková služba podporuje bežný súbor štandardných operácií editora, ako napríklad vyplnenie vyhlásení, pomoc pri podpisovaní, formátovanie kódu a označovanie, farbenie, atď.

4.2.2 Angular 5

Angular[23] je voľne širiteľná platforma na tvorbu webových aplikácií vyvinutá pod vedením vývojového tímu Angular v Google a prispievaním od komunity jednotlivcov a korporácií. Aplikácie sú napísané vyskladaním HTML šablón angularovským značením, písaním tried komponentov, pridávaním aplikačnej logiky v službách a zapuzdrovaním komponentov a služieb do modulov. Angular preberá a predstavuje obsah aplikácie v prehliadači a reaguje na interakcie používateľa podľa zadaných pokynov. Novšie verzie Angular-u vznikli úplným prepisom staršej verzie AngularJS do jazyka TypeScript (kvôli štandardom EcmaScript 6 [24]). Angular 5 je v porovnaní s predchodcami Angular 2 a Angular 4 doplnený o viaceré vylepšenia a je oveľa rýchlejší ako jeho predchádzajúce verzie.

Angular 5 bol spustený 1. novembra 2017 a oproti predchodcom sa líši nasledovnými vylepšeniami:

- Optimalizátor buildovania: napomáha odstraňovať nepotrebný kód z aplikácie.
- Angular Universal State Transfer API a DOM podpora: použitím ktorých vieme veľmi jednoducho zdieľať stav aplikácie medzi serverom a klientom.
- Vylepšenia kompilátora: jedno z najlepších vylepšení Angular 5, spočíva v zlepšení podpory inkrementálneho kompilovania aplikácie.
- Zachovanie medzier: v skorších verziách Angular-u boli počas buildovania aplikácie vytvárané nepotrebné nové riadky, tabulátory a medzery. V Angular 5 vznikla možnosť voľby, či sú pre nás potrebné alebo nie. Angular 5 podporuje ich zakázanie ako

[23] `https://angular.io/`
[24] `http://es6-features.org/`

na aplikačnej vrstve, tak aj na vrstve jednotlivých komponentov. Napríklad, ak by sme sa rozhodli zakázať ich iba pre *TestComponent*, kód by vyzeral nasledovne:

```
01. @Component({
02.             templateUrl: 'test.component.html',
03.             preserveWhitespaces: false
04.         }
05.         export class TestComponent {}
```

Obrázok 15: Príklad zákazu medzier na vrstve komponentu

Ak by sme potrebovali zakázať biele znaky na úrovni celej aplikácie, voľbu by sme aplikovali v *tsconfig.json* súbore nasledovne:

```
01. "angularCompilerOptions": {
02.     "preserveWhitespaces": false
03. }
```

Obrázok 16: Príklad zákazu medzier na aplikačnej vrstve

- Zvýšenie štandardizácie v prehliadačoch: v predchádzajúcich verziách Angular-u sme boli závislí na *i18n* vždy, keď sme chceli podporiť internacionalizáciu v našej aplikácii. V aplikácii Angular 5 teraz nemusíme závisieť od *i18n*. Poskytuje nové dátumové, číselné a menové potrubia, ktoré zvyšujú internacionalizáciu vo všetkých prehliadačoch a eliminujú potrebu *i18n* polyfillov.
- exportAs: v Angular 5 sa podporujú viacnásobné názvy pre direktívy aj komponenty.
- HttpClient: pred verziou Angular 4.3 sa používal *@angular/HTTP* modul pre všetky druhy požiadaviek HTTP. V Angular 5 bol tento modul nahradený novým modulom s názvom *HttpClientModule*, ktorý sa nachádza pod balíkom *@angular/common/http*.
- Vylepšená podpora dekorátorov: v Angular 5 vieme použiť lambda výrazy namiesto menovania funkcií, ako tomu bolo predtým. V kóde vyzerá tento rozdiel napríklad nasledovne:

```
Component({
    provider: [{
        provide: 'my-service',
        useValue: testMethod()
    }]
})
export class CustomClass {}
```

Obrázok 17: Menovanie funkcie v skorších verziách Angularu

```
Component({
            provider: [{
                provide: ''
                my - service ', useFactory: () => null}]
            })
            export class CustomClass {}
```

Obrázok 18: Použitie lambda výrazu v Angular 5

- Nové udalosti životného cyklu smerovača: v Angular 5 sa do smerovača pridalo niekoľko nových udalostí životného cyklu, ako sú *ActivationStart, ActivationEnd, ChildActivationStart, ChildActivationEnd, GuardsCheckStart, GuardsCheckEnd, ResolveStart and ResolveEnd.*

- Angular 5 podporuje TypeScript vo verzii 2.3.

- Vylepšenie rýchlosti kompilátora: zlepšenie v kompilátore umožnilo rýchlejšie vývojové buildovanie. Teraz môžeme jednoducho spustiť príkaz *ng serve/s –aot* vo vývojovom príkazovom riadku a urýchliť buildovanie.

4.2.3 RxJS - Observable

Reactive Extensions (Rx)[25] je knižnica pre vytváranie asynchrónnych programov a programov založených na udalostiach pomocou sledovateľných sekvencií a operátorov dopytu v štýle LINQ.

Dátové sekvencie môžu mať mnoho foriem, ako napríklad tok dát zo súboru alebo webovej služby, požiadavky na webové služby, systémové upozornenia alebo sériu udalostí, ako je napríklad vstup pre používateľa.

Kód, ktorý sa zaoberá viacerými udalosťami alebo asynchrónnym výpočtom, sa rýchlo komplikuje, pretože potrebuje vybudovať stavový stroj na riešenie problémov s objednávaním požiadaviek. Okrem toho sa kód musí zaoberať úspešným a neúspešným ukončením každého samostatného výpočtu. To vedie ku kódu, ktorý nedodržuje normálny tok riadenia, ktorý je mohutný a je náročné ho pochopiť a ťažko sa udržiava.

[25] `http://reactivex.io/rxjs/`

Reaktívne rozšírenia reprezentujú všetky tieto dátové sekvencie ako pozorovateľné sekvencie. Aplikácia sa môže prihlásiť k týmto pozorovateľným sekvenciám a prijímať asynchrónne upozornenia pri príchode nových údajov.

RxJS sa používa na rozvrhovanie asynchrónnych výpočtov a výpočtov založených na udalostiach. V nástroji Rx sa *observer* prihlási k *Observable*. Potom *observer* reaguje na akúkoľvek položku alebo sekvenciu položiek, ktoré *Observable* vysiela. Tento vzor uľahčuje súbežné operácie, pretože sa nemusia blokovať počas čakania, kým *Observable* nevyšle objekty. Namiesto toho vytvoria strážcu vo forme *observer-a*, ktorý je pripravený primerane reagovať na čokoľvek v budúcnosti, čo *Observable* urobí.

RxJS nemá žiadne závislosti a hladko spolupracuje s oboma synchrónnymi dátovými tokmi, ako sú iteračné objekty v jazyku JavaScript aj asynchrónnymi tokmi s jedinou hodnotou, ako napríklad *Promises*. [22]

Tabuľka 2 referuje prehľad prístupov spracovania toku dát:

	Jedna návratová hodnota	Viac návratových hodnôt
Synchronous/Interactive	Object	Iterables (Array|Set|Map|Obj)
Asynchronous/Reactive	Promise	Observable

Tabuľka 2: Prehľad prístupov spracovania toku dát [22]

4.2.4 SCSS

Sass[26] je preprocesor skriptovací jazyk, ktorý je interpretovaný alebo kompilovaný do kaskádových štýlov (CSS). SassScript je samotný skriptovací jazyk a pozostáva z dvoch syntaxí. Pôvodná syntax, nazývaná "odsadená syntax", používa odsadenie na oddelenie kódových blokov a nové riadky na oddelenie pravidiel. Používa zložené zátvorky na označenie blokov kódu a bodkočiarky na oddelenie riadkov v rámci bloku. Nezáleží na úrovniach odsadenia alebo na vynechanom priestore [23]. V skutočnosti je SCSS syntax nadradenou CSS čo znamená, že SCSS obsahuje všetky funkcie CSS, ale navyše bola rozšírená o funkcie Sass-u. Súborom s odsadenou syntaxou a SCSS sa tradične dávajú koncovky *.sass*, respektíve *.scss*.

4.3 Vývojové prostredie

Vyvíjali sme pod OS Ubuntu 16.04 LTS a ako integrované vývojové prostredie sme použili editor Visual Studio Code. Tak ako pri predchádzajúcom projekte sme ako databázu použili PostgreSQL, no aktualizovali sme ju na najnovšiu verziu 10.1. Na vytvorenie

[26] `https://sass-lang.com/`

virtuálneho prostredia sme použili pyenv[27] a na inštaláciu potrebných balíčkov pre Python nástroj pip[28].

[27] https://github.com/pyenv/pyenv
[28] https://pypi.org/project/pip/

5 Implementácia

V danej kapitole popisujeme spôsob a postup pri vývoji používateľského rozhrania nášho rozvrhového systému. Detailnejšie sa venujeme jednotlivým modulom aplikácie a ich implementácii ako samostatných oddielov systému, z ktorých každý plní inú funkcionalitu a slúži na iné účely. Systému sme z predchádzajúcej verzie ponechali rovnaký názov Elisa.

5.1 Architektúra klienta

Systém sme na strane webového klienta implementovali ako SPA (z angl. Single page application) jednostránkovú aplikáciu . Pre tento typ aplikácií je príznačné, že sa jej zdrojové kódy (HTML, JS a CSS) načítavajú už pri prvotnom načítaní aplikácie. Stránkovanie je vykonávané prostredníctvom *routing navigation*, kedy sa síce URL mení, ale stránka nemá potrebu sa počas behu obnovovať. Na pohľad to teda vyzerá, že klikom na iný modul sa zobrazuje iná stránka, avšak v skutočnosti sa na danej stránke mení iba jej obsah a nie stránka celá.

5.2 Štruktúra klienta

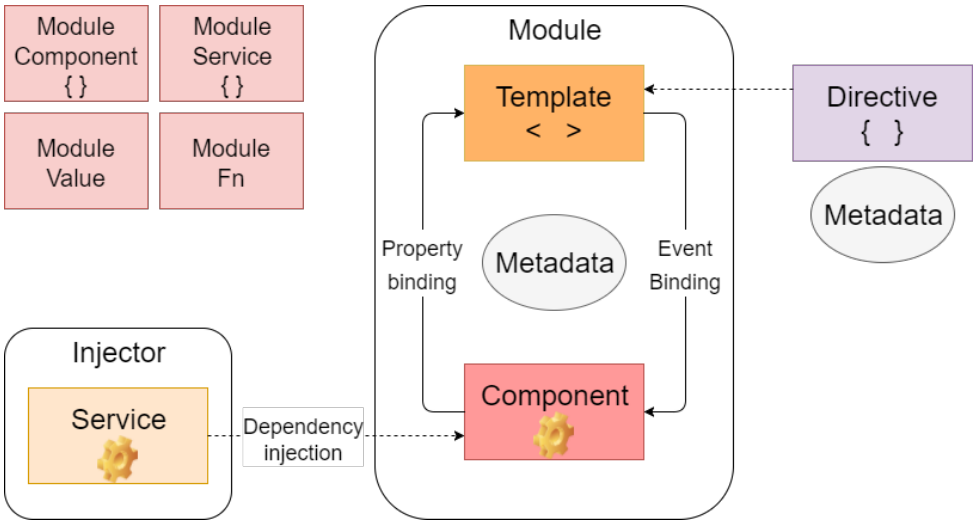
Klientskú časť našej aplikácie sme vyvíjali hierarchicky ako moduly, podmoduly a komponenty. Najskôr sme ako prvé implementovali menšie časti - komponenty, ktoré sme neskôr spájali do väčších podmodulov a nakoniec tie do ucelených modulových častí. Každý komponent má zadefinované svoje vlastné správanie a zobrazovaciu šablónu. Jednotlivé moduly sú rozdelené ako podstránky aplikácie a majú svoju URL, napríklad zobrazenie prehľadu má cestu `/dashboard`

Existuje osem hlavných blokov aplikácie Angular:

- Modul: Angular aplikácie sú modulárne a Angular má vlastný modulárny systém nazvaný *NgModules*. Modul je navrhnutý tak, aby vykonával jediný úkon.
- Komponent: trieda so šablónou, ktorá sa zaoberá zobrazením (z angl. View) aplikácie a obsahuje základnú logiku stránky.
- Šablóny: forma HTML, ktorá Angular-u hovorí, ako má vykresliť komponent. Zobrazenie je definované pomocou šablóny.
- Metadáta: hovoria o tom, ako spracovať triedu (z angl. Class).
- Väzba údajov: mechanizmus na koordináciu medzi časťami šablóny a časťami komponentu. Existujú štyri formy syntaxe väzby údajov.

- Direktívy: trieda s metadátami. Keď Angular vykresľuje šablóny, transformuje DOM podľa pokynov uvedených v direktívach.
- Služby: široká kategória zahŕňajúca akúkoľvek hodnotu, funkciu alebo funkcionalitu, ktorú aplikácia potrebuje. Je to trieda s úzkym, ale dobre definovaným účelom.
- Vkladanie závislostí: spôsob, ako dodať novú inštanciu triedy s plne vytvorenými závislosťami, ktoré vyžaduje. Väčšina závislostí sú služby. Angular toto vkladanie používa na poskytovanie nových komponentov s potrebnými službami.

Naledujúci obrázok popisuje vnútornú architektúru použitú v Angular-e:



Obrázok 19: Architektúra použitá v Angular-e [24]

Pri implementácii sme postupovali najmä vo vývoji do šírky, čiže sme sa snažili o poskytnutie minimálnej funkcionality s dôrazom na celkový dizajn a náhľad možných budúcich potrieb takejto rozvrhovej aplikácie. Nedbali sme pritom len na potreby samotného rozvrhára, ale implementovali sme aj moduly, ktoré môžu byť využívané inými používateľmi, ktorí sa priamo nezúčastňujú rozvrhovania, ale systém má byť určený aj im. Jednotlivé komponenty sme sa snažili implementovať tak, aby ich bolo možné kedykoľvek a kdekoľvek v aplikácii jednoducho opakovane použiť.

Medzi takéto komponenty patria:

- komponent registrácie a prihlasovania,
- formulárové prvky a ich validácia,
- prehľadové karty s informáciami o najdôležitejších metrikách,

- prehľad blížiacich sa stretnutí a udalostí,
- kalendár udalostí z viacerých časových náhľadov s funkcionalitou Drag&Drop a rozširovania ťahaním okrajov,
- mailbox na mailovú komunikáciu,
- chat na rýchlu konverzáciu,
- zoznam kontaktov,
- osobný profil používateľa,
- konfiguračný panel,
- horné menu a bočná navigácia,
- smerovanie medzi stránkami aplikácie,
- dialógové okná,
- dizajnové prvky aplikácie.

Komponenty vytvárame pomocou Angular CLI. Príkazom `ng generate` vytvoríme komponent, pridáme ho do deklarácií v našom *ElisaModule*, ktorý sa následne importuje ako celok do *AppModule* na najvyššej vrstve aplikácie. *ElisaModule* vlastne takýmto spôsobom importuje všetky vytvorené hlavné komponenty a združuje ich na jednom mieste. Vo všeobecnosti sa jednoduchý komponent vytvára nasledovne:

`ng generate component` *`component_name`*

Každý komponent má životný cyklus vytvorenia, aktualizácie a zničenia. Spolu so súborom *.ts* komponentu sa vytvorí trieda, ktorá implementuje *OnInit* rozhranie z *Core* knižnice. Niektoré komponenty potrebujú vykonať volanie API na načítanie údajov, ale komponent by to sám o sebe nikdy nemal robiť. Na tieto účely sú v Angular používané súbory *service*, takže sa komponent môže sústrediť iba na zobrazovanie údajov a prenos údajov do iných komponentov, ktoré ich potrebujú. V našom prípade pre komponent kalendár vyzerá úvodné načítanie dát nasledovne:

Obrázok 20: Ukážka kódu inicializačného načítania dát

V *ngOnInit* časti uskutočníme požiadavku API volania na aktualizovanie údajov počas *OnInit* fázy komponentu pri obnovení stránky, avšak prevedenie tohto API volania sa implementuje v *service* súbore.

Obrázok 21: Ukážka kódu požiadavky v *ngOnInit*

Takýto *service* súbor môžme takisto vytvoriť pomocou Angular CLI nasledovne:

`ng generate service` *`service_name`* `-module=`*`module_name`*

To vytvorí triedu *service_name* a pridá túto triedu služby medzi *providers* do *module_name* modulu.

Angular používa triedu *HttpClient* na komunikáciu so vzdialeným serverom. Na použitie v našej službe (a kdekoľvek inde v aplikácii), musíme importovať *HttpClientModule* do *AppModule*. Potom môžme vložiť *HttpClient* do konštruktora nášho *service* súboru. Následne vytvoríme metódy na CRUD operácie s dátami, ktoré volajú HTTP volania na API. Prihlásili sme sa k vrátenej *Observable* a priradili vrátene dáta k deklarovanej premennej.

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { ActivatedRouteSnapshot, Resolve, RouterStateSnapshot } from '@angular/router';

import { Observable } from 'rxjs/Observable';
import { Subject } from 'rxjs/Subject';

@Injectable()
export class CalendarService implements Resolve<any>
{
    events: any;
    onEventsUpdated = new Subject<any>();

    constructor(private http: HttpClient)
    {}

    resolve(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): Observable<any> | Promise<any> | any
    {
        return new Promise((resolve, reject) => {
            Promise.all([
                this.getEvents()
            ]).then(
                ([events]: [any]) => {
                    resolve();
                },
                reject
            );
        });
    }

    getEvents()
    {
        return new Promise((resolve, reject) => {

            this.http.get('api/calendar/events')
                .subscribe((response: any) => {
                    this.events = response.data;
                    this.onEventsUpdated.next(this.events);
                    resolve(this.events);
                }, reject);
        });
    }

    updateEvents(events)
    {
        return new Promise((resolve, reject) => {
            this.http.post('api/calendar/events', {
                id  : 'events',
                data: [...events]
            })
```

Obrázok 22: Ukážka kódu metód v *service* súbore

5.3 Prehľad modulov aplikácie

Tak ako sme už niekoľkokrát vyššie spomenuli, naša aplikácia je založená na moduloch, ktoré ako ucelené časti kódu vykonávajú každý svoju špecifickú úlohu. V danej kapitole si popíšeme jednotlivé moduly a priblížime si implementáciu niektorých dôležitejších komponentov a funkcionalít.

5.3.1 Modul autentifikácie

Tento modul aplikácie slúži na autentifikačné činnosti ako sú registrácia nového a prihlásenie existujúceho používateľa. Do tohto modulu môže takisto patriť aj stránka na obnovu zabudnutého hesla, stránka na zablokovanie konta v prípade opakovaného chybného pokusu o prihlásenie prípadne stránka na potvrdenie odoslania nového vygenerovaného hesla.

V prípade týchto komponentov na registráciu a prihlasovanie sme vytvorili formuláre a potrebné polia na zadávanie vstupu pomocou Angular Material komponentov. Na jednotlivé vyžadované polia sme aplikovali validátory (z angl. validators) a prostredníctvom *Observable* počúvame na zmeny (*valueChanges*) vykonávané na formuláry. Tieto polia sa teda validujú dynamicky a v prípadne nevalidnosti je používateľ upozornený na porušené pravidlá, ktoré nedodržal počas zadávania vstupu.

```
38
39      ngOnInit()
40      {
41          this.registerForm = this.formBuilder.group({
42              name           : ['', Validators.required],
43              email          : ['', [Validators.required, Validators.email]],
44              password       : ['', Validators.required],
45              passwordConfirm: ['', [Validators.required, confirmPassword]]
46          });
47
48          this.registerForm.valueChanges.subscribe(() => {
49              this.onRegisterFormValuesChanged();
50          });
51      }
52
53      onRegisterFormValuesChanged()
54      {
55          for ( const field in this.registerFormErrors )
56          {
57              if ( !this.registerFormErrors.hasOwnProperty(field) )
58              {
59                  continue;
60              }
61
62              // Clear previous errors
63              this.registerFormErrors[field] = {};
64
65              // Get the control
66              const control = this.registerForm.get(field);
67
68              if ( control && control.dirty && !control.valid )
69              {
70                  this.registerFormErrors[field] = control.errors;
71              }
72          }
73      }
74  }
```

Obrázok 23: Ukážka kódu na zavedenie validátorov a počúvania na zmeny

5.3.2 Modul prehľad

Modul prehľad slúži ako hlavná stránka aplikácie. V nej sa používateľovi zobrazujú základné informácie a metriky, ktoré sú pre neho podstatné, resp. by ho mohli zaujímať. Tieto metriky je v budúcnosti možné meniť v závislosti od typu prihláseného používateľa na základe používateľských práv, potrieb a preferencií.

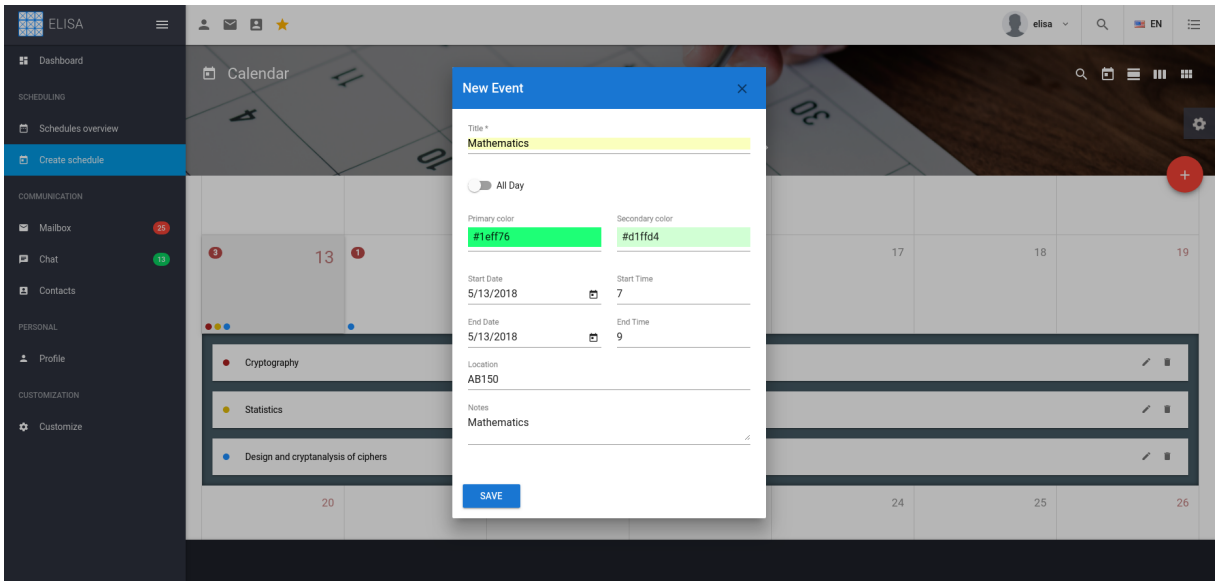
K daným zobrazovaným informáciam a metrikám sú samozrejme potrebné dáta. Na takéto účely sme si na klientovi vytvorili simulovanú databázu *elisa-fake-db.service.ts*, ktorá nám "vytvára" potrebné dáta. Následne tieto dáta doťahujeme na našich Angular Material *widgetoch*. Ako návratové hodnoty pri získavaní dát používame *Promise*, nakoľko sa jedná o jednu návratovú hodnotu a nie celý tok (z angl. stream).

```
/**
 * Resolve
 * @param {ActivatedRouteSnapshot} route
 * @param {RouterStateSnapshot} state
 * @returns {Observable<any> | Promise<any> | any}
 */
resolve(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): Observable<any> | Promise<any> | any
{

    return new Promise((resolve, reject) => {

        Promise.all([
            this.getProjects(),
            this.getWidgets()
        ]).then(
            () => {
                resolve();
            },
            reject
        );
    });
}

getProjects(): Promise<any>
{
    return new Promise((resolve, reject) => {
        this.http.get('api/project-dashboard-projects')
            .subscribe((response: any) => {
                this.projects = response;
                resolve(response);
            }, reject);
    });
}

```

Obrázok 24: Ukážka kódu na doťahovanie dát pomocou *Promise*

5.3.3 Modul rozvrhovanie

Modul rozvrhovanie slúži na prácu s rozvrhmi a kalendárom udalostí. Tento modul je základným stavebným kameňom pre používateľa typu rozvrhár. Medzi základné možnosti tohto modulu patrí predovšetkým vkladanie nových udalostí do kalendára, upravovanie a mazanie existujúcich udalostí, funkcionalita Drag&Drop na zmenu časového umiestnenia udalosti v rozvrhu a funkcionalita rozširovania časovej rezervácie udalosti ťahaním jej okrajov.



Obrázok 25: Komponent pridávania udalosti do kalendára

```
{
    id  : 'events',
    data: [
        {
            start    : addHours(startOfDay(new Date()), 9),
            end      : addHours(startOfDay(new Date()), 11),
            title    : 'Cryptography',
            allDay   : false,
            color    : {
                primary  : '#ad2121',
                secondary: '#FAE3E3'
            },
            resizable: {
                beforeStart: true,
                afterEnd   : true
            },
            draggable: true,
            meta     : {
                location: 'BC150',
                notes   : 'Cryptography in BC150'
            }
        },
```

Obrázok 26: Ukážka úryvku JSON objektu dát kalendára

```
import { isInside } from './util';
var CalendarDragHelper = /** @class */ (function () {
    function CalendarDragHelper(dragContainerElement, draggableElement) {
        this.dragContainerElement = dragContainerElement;
        this.startPosition = draggableElement.getBoundingClientRect();
    }
    CalendarDragHelper.prototype.validateDrag = function (_a) {
        var x = _a.x, y = _a.y;
        var newRect = Object.assign({}, this.startPosition, {
            left: this.startPosition.left + x,
            right: this.startPosition.right + x,
            top: this.startPosition.top + y,
            bottom: this.startPosition.bottom + y
        });
        return isInside(this.dragContainerElement.getBoundingClientRect(), newRect);
    };
    return CalendarDragHelper;
}());
export { CalendarDragHelper };
```

Obrázok 27: Ukážka kódu implementácie funkcionality Drag&Drop

```
import { isInside } from './util';
var CalendarResizeHelper = /** @class */ (function () {
    function CalendarResizeHelper(resizeContainerElement, minWidth) {
        this.resizeContainerElement = resizeContainerElement;
        this.minWidth = minWidth;
    }
    CalendarResizeHelper.prototype.validateResize = function (_a) {
        var rectangle = _a.rectangle;
        if (this.minWidth && rectangle.width < this.minWidth) {
            return false;
        }
        return isInside(this.resizeContainerElement.getBoundingClientRect(), rectangle);
    };
    return CalendarResizeHelper;
}());
export { CalendarResizeHelper };
```

Obrázok 28: Ukážka kódu implementácie funkcionality “ťahanie okrajov”

Funkcionality súvisiace s kalendárom sme implementovali za pomoci komponentov a nástrojov angular-calendar[29].

5.3.4 Modul komunikácia

Tento modul zabezpečuje komunikačný kanál systému. V budúcnosti bude možné tento mailový klient napojiť na správy z AIS. Okrem komponentu mailovej komunikácie tento modul obsahuje aj komponent chat-u, ktorý má slúžiť predovšetkým na rýchlu komunikáciu medzi používateľmi. Posledným komponentom komunikačného modulu je zoznam kontaktov na prezeranie. Implementovali sme množstvo okrajových funkcionalít na manipuláciu s mailami a správami, ich vyberanie, označovanie atď. Dáta v podobe správ a kontaktov sme taktiež doťahovali z našej simulovanej databázy.

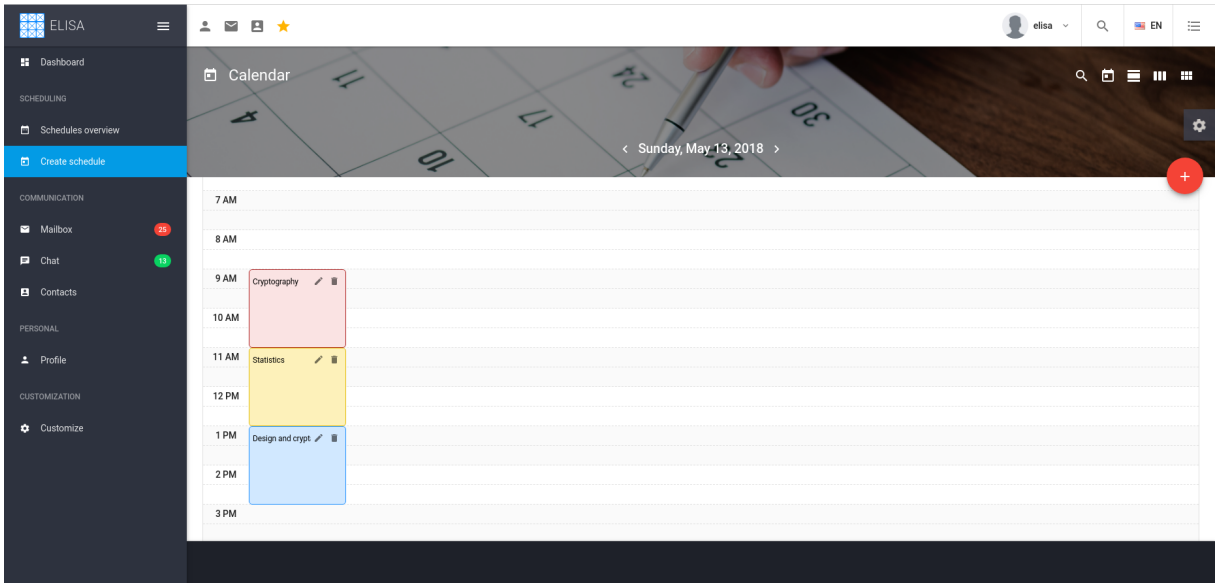
[29] `https://github.com/mattlewis92/angular-calendar`

5.3.5 Modul profil

Modul profilu zobrazuje najzákladnejšie informácie o prihlásenom používateľovi. V budúcnosti tu vzniká možnosť rozšírenia na zobrazovanie profilov aj iných používateľov, napríklad po kliknutí na niektorý kontakt v zozname prihláseného používateľa prípadne po celkovom vyhľadávaní osôb v systéme. Tento modul má zatiaľ iba čisto informatívny charakter o používateľových osobných údajoch.

5.4 Dizajn a konfigurácia používateľského rozhrania

Dizajn používateľského rozhrania sme sa snažili orientovať podľa moderných trendov vývoja webových aplikácií. Kaskádové štýly *scss* sme aplikovali na globálnej úrovni a pre niektoré komponenty so špeciálnymi potrebami sme vytvárali lokálne *scss* súbory.



Obrázok 29: Dizajn používateľského rozhrania kalendára udalostí

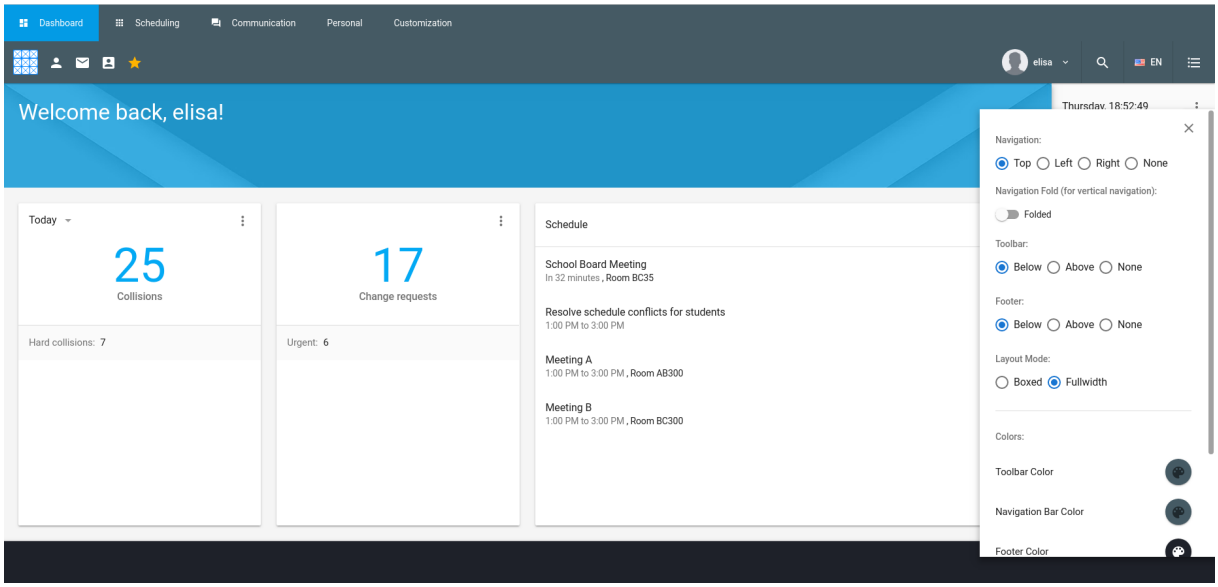
Celá aplikácia je responzívna, to znamená, že je schopná sa prispôsobovať rôznym rozlíšeniam zariadení pri zachovaní plnej funkcionality aplikácie. To v budúcnosti eliminuje potrebu vyvíjania dodatočnej mobilnej aplikácie. Responzívnosť používateľského rozhrania sme zabezpečili pomocou externých direktív vytvorených v nástroji Angular Flex-Layout[30].

[30] `https://github.com/angular/flex-layout`

```
<div id="calendar" class="page-layout simple fullwidth" elisaPerfectScrollbar>

    <!-- HEADER -->
    <div class="header p-16 p-sm-24" [ngClass]="viewDate | date:'MMM'">

        <div class="header-content" fxLayout="column" fxLayoutAlign="space-between">

            <div class="header-top" fxLayout="row" fxLayoutAlign="space-between center" fxLayout.xs="column">

                <div class="logo mb-16 mb-sm-0" fxLayout="row" fxLayoutAlign="start center">
                    <mat-icon class="logo-icon" *elisaIfOnDom [@animate]="{value:'*',params:{delay:'50ms',scale:'0.2'}}">today</mat-icon>
                    <span class="logo-text" *elisaIfOnDom [@animate]="{value:'*',params:{delay:'100ms',x:'-25px'}}">Calendar</span>
                </div>

                <!-- TOOLBAR -->
                <div class="toolbar" fxLayout="row" fxLayoutAlign="start center">

```

Obrázok 30: Ukážka použitia direktív pre zachovanie responzívnosti komponentu

Jednou z funkcionalít, ktorú naša aplikácia poskytuje, je možnosť konfigurácie zobrazovania elementov v používateľskom rozhraní. Používateľ si môže napríklad zmeniť rozloženie navigačných panelov, vie si určiť, či majú byť dané panely roztiahnuté alebo skryté, môže si prispôsobovať farebné prevedenie panelov podľa svojich potrieb a preferencií, prípadne si vie meniť zobrazenie aplikácie na zúžené alebo v plnom rozlíšení.



Obrázok 31: Príklad vlastnej konfigurácie používateľského rozhrania

Záver

Hlavným cieľom našej práce bolo aktualizovať existujúci stav projektu a pokračovať v jeho vývoji, s dôrazom kladeným na používateľské rozhranie.

Najskôr sme analyzovali problematiku tvorby rozvrhov a oboznámili sme sa s niektorými existujúcimi riešeniami. Zosumarizovali sme si celý proces tvorenia rozvrhu na FEI a jednotlivé požiadavky kladené na systém. Z oboch predchádzajúcich prác sme si osvojili aktuálny stav riešenia a navrhli zmeny potrebné na docielenie stanovených cieľov používateľského rozhrania systému.

Našou navrhovanou zmenou boli používané technológie a zmena architektúry v klientskej časti. Pripomenuli sme si používané nemenené technológie na strane servera z predchádzajúcej diplomovej práce a popísali sme si jednotlivé nami navrhované technológie. Po rozbehaní systému z predchádzajúcej práce sme sa pustili do implementácie nového používateľského rozhrania. Priblížili sme si implementáciu komponentov, z ktorých sme vyskladávali jednotlivé moduly a taktiež implementáciu niektorých kľučových funkcionalít.

Systém je po oboch predchádzajúcich prácach a teraz po pridaní našej časti na veľmi dobrej ceste k používaniu v budúcnosti, no vyžaduje si ešte veľmi veľa práce a úsilia. Softvér stále nie je plne dokončený, nakoľko sa jedná o skutočne robustný systém s množstvom potrebných a plánovaných funkcionalít. Samotný vývoj si vyžadoval veľmi veľa času, nakoľko sme sa rozhodli pre súčasné moderné technológie a momentálne je ešte pomerne náročné nájsť k nim relevantné zdroje a podrobné postupy v komunite na internete. Taktiež splnenie požiadaviek na responzívnosť nám zabralo približne 30% času celého vývoja. To sa nám však podarilo a do budúcna nevznikne potreba pre vznik mobilnej aplikácie systému.

Ako návrh na ďalší postup by sme uviedli potrebu zapracovať na tzv. middleware vrstve, a to na prepojení serverovej časti implementovanej v predchádzajúcej práci s klienstkou časťou implementovanou v našej práci.

Medzi možné vylepšenia systému navrhujeme:

- Offlina funkcionalita
- Zapracovanie používateľských práv do množiny ponúkaných funkcionalít
- Integrácia systému so systémom AIS (napr. v komunikačnom moduli)
- Prepojenie doplnkových informačných komponentov na služby tretích strán

Zoznam použitej literatúry

1. WILLEMEN, R.J. *School timetable construction : algorithms and complexity*. 2002. ISBN 90-386-1011-4. Dizertačná práca. Technische Universiteit Eindhoven.
2. 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.
3. *UniTime* [online] [cit. 2017-03-21]. Dostupné z: `http://www.unitime.org/present/unitime-highlights.pdf`.
4. *CELCAT Timetabler*. Dostupné tiež z: `http://www3.imperial.ac.uk/pls/portallive/docs/1/10225698.PDF`.
5. *Asc TimeTables*. Dostupné tiež z: `help.asctimetables.com/pdf/asc_timetables_en_P1.pdf/`.
6. *Mimosa - Scheduling Software for School and University Timetables* [online] [cit. 2017-04-04]. Dostupné z: `http://www.mimosasoftware.com/`.
7. KNAPEREKOVÁ, Emília. *Rozvrhový systém pre vysoké školy*. 2014. Diplomová práca. Slovenská technická univerzita v Bratislave. EČ: FEI-5384-56111.
8. RAČÁK, Martin. *Rozvrhový systém pre FEI*. 2017. Diplomová práca. Slovenská technická univerzita v Bratislave. EČ: FEI-5384-53920.
9. MACCAW, Alex. *JavaScript Web Applications*. First Edition. O'Reilly Media, 2011. ISBN 978-1-449-30351-8.
10. PIISPANEN, Mark. *Modern architecture for large web applications*. 2017. Dostupné tiež z: `https://jyx.jyu.fi/dspace/bitstream/handle/123456789/54129/URN:NBN:fi:jyu-201705272524.pdf?sequence=1`.
11. *ReduxJS* [online] [cit. 2018-04-02]. Dostupné z: `https://redux.js.org/`.
12. *Building a Redux application with Angular 2*. Dostupné tiež z: `https://www.pluralsight.com/guides/front-end-javascript/building-a-redux-application-with-angular-2-part-1`.
13. FIELDING, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures*. 2000. Dizertačná práca. University of California.

14. RESCHKE, J. *The 'Basic' HTTP Authentication Scheme* [online]. RFC Editor, 2015 [cit. 2017-03-20]. ISSN 2070-1721. Dostupné z: https://www.rfc-editor.org/rfc/pdfrfc/rfc7617.txt.pdf. RFC. RFC Editor.

15. *AUTHENTICATION SCHEMES IN REST API.* Dostupné tiež z: https://payatu.com/authentication-schemes-rest-api/.

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

17. 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.

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

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. KUBALA, Juraj. *Progressive web app with Angular 2 and ASP.NET.* 2017. Dostupné tiež z: https://www.theseus.fi/bitstream/handle/10024/139795/Kubala_Juraj.pdf?sequence=1.

22. *RxJS.* Dostupné tiež z: http://xgrommx.github.io/rx-book/index.html.

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

24. *Angular.* Dostupné tiež z: https://angular.io/guide/architecture.

Prílohy

A Technická dokumentácia

Systém si na svoju funkčnosť vyžaduje prostredie, na ktorom je podporovaný Python 3 a databáza PostgreSQL. V prípade potreby vyvíjať bude nevyhnutná inštalácia Node.js. Postup konfigurácie a spustenia vývojového prostredia sa nachádza popísaný na priloženom médiu v priečinku so zdrojovými súbormi aplikácie v súbore README.md jednotlivo pre server a pre klienta. Pre prípad potreby sme nami používané prostredie popísali v kapitole 4.3.

Na vývoj, prípadne testovanie, je možné aplikáciu spúšťať lokálne prostredníctvom web serverov. Na samotné produkčné nasadenie celého systému je však potrebné použitie reálneho plnohodnotného servra, na ktorý sa následne nahrajú všetky potrebné statické súbory výslednej aplikácie (HTML, CSS, JS, ...) a budú servírované ako webové služby. Pri tomto nasadení odporúčame postupovať podľa platnej dokumentácie Django [31].

Kvôli bezpečnosti systému je vhodné na komunikáciu medzi klientom a servrom využiť niektorú so šifrovacích služieb, napr. službu Let's Encrypt[32].

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

B Štruktúra elektronického nosiča

Štruktúra a prehľad základných 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 serverovej časti
|   |   |-- requirements.txt       # Python pip závislosti
|   |   |-- school                 # balíček na prácu so školskými dátami
|   |   |-- timetables             # balíček na prácu s rozvrhmi
|   |-- elisa-client-new           # klientská časť
|   |   |-- package.json           # metadáta npm balíčku
|   |   |-- tsconfig.json          # metadáta kompilátora
|   |   |-- README.md              # dokumentácia klientskej časti
|   |   |-- src                    # zdrojový kód klienta
|   |   |   |-- @elisa             # doplnkové súbory a  štýly
|   |   |   |-- app                # hlavné zdrojové súbory
|   |   |   |-- assets             # obrázky, ikony, príklady
|   |   |   |-- index.html         # inicializačný zdrojový súbor
|-- thesis                         # súbory diplomovej práce
```