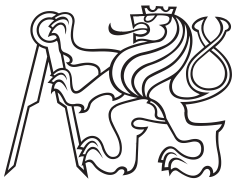
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra kybernetiky

Systém pro tvorbu rozvrhu a rezervaci prostor

Jan Mádle

Vedoucí: RNDr. Marko Genyk-Berezovskyj
Obor: Otevřená informatika
Studijní program: Počítačové vědy
Květen 2019

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Mádle** Jméno: **Jan** Osobní číslo: **461401**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra kybernetiky**

Studijní program: **Otevřená informatika**

Studijní obor: **Informatika a počítačové vědy**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Systém pro tvorbu rozvrhu a rezervaci prostor

Název bakalářské práce anglicky:

Schedule and Reservation System

Pokyny pro vypracování:

Seznamte se s problematikou tvorby rozvrhu a rezervaci prostor. Proveďte rešerši nejrozšířenějších systémů pro tvorbu a prezentaci rozvrhů a rezervaci prostor, identifikujte jejich silné a slabé stránky a vlastnosti. Systémy hodnoťte podle několika hledisek, zejména podle míry automatizace tvorby rozvrhu, podle možnosti přizpůsobení požadavkům uživatelů během provozu a podle uživatelské přívětivosti, případně podle dalších kritérií. Podle výsledku rešerše sestavte návrh vlastností vlastního systému, který implementujte v alespoň základní pilotní podobě.
Pro automatickou tvorbu rozvrhu využijte metod programování s omezujícími podmínkami (constraint programming), případně jiných přístupů pro vyřešení tzv. rozvrhovacího problému (scheduling problem) alespoň pro menší vstupy. Zhodnoťte funkcionalitu a možnosti nasazení buď pilotní verze nebo jejího vhodného rozšíření s ohledem na rozsah požadavků a předpokládanou zátěž systému.
Vytipujte modelovou cílovou skupinu uživatelů své aplikace, proveďte s ní uživatelské testování Vašeho systému. Implementovaný systém doprovoďte uživatelskou a programátorskou dokumentací.

Seznam doporučené literatury:

[1] Pinedo L. Michael – Scheduling Theory, Algorithms, and Systems – Springer, 2008.
[2] Baptiste Philippe, Pape Le Claude, Nuijten Wim – Constraint-based scheduling : applying constraint programming to scheduling problems – Kluwer Academic Publishers, 2001.
[3] Sauro Jeff, Lewis R. James – Quantifying the user experience – Elsevier Inc., 2012.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

RNDr. Marko Genyk-Berezovskyj, katedra kybernetiky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **09.01.2019** Termín odevzdání bakalářské práce: **24.05.2019**

Platnost zadání bakalářské práce: **30.09.2020**

RNDr. Marko Genyk-Berezovskyj	doc. Ing. Tomáš Svoboda, Ph.D.	prof. Ing. Pavel Ripka, CSc.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

Poděkování

Děkuji především svému vedoucímu panu Berezovskému za rady a trpělivost s vedením práce, dále také rodině, přítelkyni a kamarádům, kteří mě drželi nad vodou v negativních chvílích. A v neposlední řadě panu Šůchovi za konzultaci ohledně existujících technologií na téma rozvrhování.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne , 23. května 2019

. . . . . .
podpis autora práce

Abstrakt

Tato bakalářská práce se zabývá problematikou tvorby rozvrhu a rezervace prostor. Seznamuje čtenáře s možnými řešeními, jak tvořit rozvrh automaticky. Uvádí konkrétní řešení s pomocí celočíselného lineárního programování. Prezentuje návrh systému, včetně všech jeho částí (back end, batabáze i front end). Rozebírá implementaci systému v jeho zjednodušené podobě. Uvádí uživatelské testování identifikující nedostatky v implementovaném uživatelském rozhraní a navrhuje jejich řešení. Implementaci doprovází uživatelskou a programátorskou dokumentací.

Klíčová slova: rozvrhování, celočíselné lineární programování, tvorba rozvhu, Node.js, Express, SQL MariaDB, Ejs

Vedoucí: RNDr. Marko Genyk-Berezovskyj
Katedra kybernetiky, FEL ČVUT.
(Karlovo náměstí 13 121 35 Praha 2)

Abstract

This bachelor thesis engages the problem of scheduling and design of reservation system. It introduces the reader existing solutions of automatic timetabling. Provides concrete solution with help of integer linear programming. It represents design of the system along with all of his parts (back end, database and front end). Discusses system implementation in his simplified form. With the help of user interface testing, possible problems and solutions, with the user interface, are discussed. Implementation is accompanied by user and programmer documentation.

Keywords: scheduling, integer linear programming, timetabling, Node.js, Express, SQL MariaDB, Ejs

Title translation: Schedule and reservation system

Obsah

Obrázky

Tabulky

Kapitola 1

Úvod

1.1 Úvod do rozvrhování

Problematice rozvrhování se aktivně v posledních 50 letech věnuje řada odborníků. Z jejich prací vyvstává mnoho různých způsobů řešení této problematiky a některé z nich si popíšeme v kapitole 2.2. Obecně lze říci, že tvorba rozvrhu (Scheduling) je proces, u kterého alokujeme dostupné prostředky k nějakým úkolům (případně k fyzickým místům, strojům aj.) do předurčených časových bloků, za existence různých omezujících podmínek. Snažíme se při tom optimalizovat jeden nebo více stanovených cílů [1].

V praxi mohou mít prostředky a úkoly mnoho forem. Do tématu rozvrhování spadá velké množství problémů, např. industriální řízení, logistika, řízení procesů a další. Stejně jako u prostředků a úkolů, může mít i stanovený cíl mnoho podob. Jedná se například o minimalizaci času potřebného k dokončení cíle, minimalizaci počtu úkolů dokončených po konečném termínu (deadline) či maximalizaci využití zdrojů (využití např. v leteckém logistickém průmyslu). Případné dostupné publikace věnující se této problematice [2][3].

1.2 Úvod do tvorby systému

Při tvorbě softwarového systému je vhodné předem identifikovat cílovou skupinu uživatelů. Software pro potřeby této bakalářské práce se snaží být co nejvíce obecný, aby se mohl aplikovat pro větší množství institucí a mohl být využit větší skupinou uživatelů. Jako příklad uveďme sportovní halu, základní školu nebo jiné instituce pronajímající/spravující prostory.

Tento přístup přináší řadu výhod, jednak využití pro více různých institucí, tak použití na více místech zároveň, bez potřeby spravovat dva různé systémy, oproti jednomu velmi konkrétnímu systému, u kterého bude většina funkcionality uživatelům dostupná, ale ne zcela využívána.

Avšak takto obecně zpracovaný software přináší i jisté nevýhody. V případě řešení velmi specifického problému, např. ověřování splnění studijního plánu, případně jiných osnov, bude nutné systém modifikovat.

1.3 Obecné předpoklady pro vytvářený systém

Před rozborem problematiky, stanovme velmi obecně, co budeme od systému požadovat. Systém musí mít rozhraní pro vstup dat, především pro správu a tvorbu rozvrhu. Dále musí umožňovat manipulaci s těmito daty a zároveň také vytvářet rozvrh na základě zadaných vstupních dat automaticky. Navrhneme a implementujeme všechny potřebné části systému, to znamená front end, databázi i back end. Návrh rozebereme více do hloubky v kapitole 3.

Do tématu rozvrhování spadá mnoho problémů, a proto se omezíme v této bakalářské práci pouze na problematiku alokování místností, případně skupiny místností, kurzům (vedených uživatelem, případně skupinou uživatelů) do předem stanovených časových bloků.

Stejně tak se omezíme podobu stanoveného cíle, který pro nás bude minimalizovat součet počtu nepreferovaných místností/prostor pro všechny uživatele žádající o časový slot. (Každému kurzu se pokusíme přiřadit vyhovující čas.) Případným druhým cílem, bude minimalizovat počet porušení podmínek. Více se této problematice budeme věnovat v kapitole 3.6.

Kapitola 2

Rešerše a rozbor problematiky

2.1 Terminologie

Při tvorbě rozvrhu se setkáváme s důležitými pojmy, které si v této kapitole upřesníme [4]:

- Rozvrh - je soubor provázaných dat, kde k jednotlivým předem definovaným časovým blokům přiřazujeme úlohy (kurzy), podle zadaného optimalizačního kritéria.
- Úplný rozvrh - v rozvrhu jsou umístěny všechny úlohy ze zadání problému.
- Částečný rozvrh - některé úlohy ze zadání problému nejsou umístěny/přiřazeny.
- Konzistentní rozvrh - rozvrh, ve kterém jsou splněna všechna omezení kladená na zdroje a umístěné/přiřazené úlohy.
- Optimální rozvrh - umístění úloh na stroje je optimální vzhledem k zadanému optimalizačnímu kritériu.

Je také potřeba rozlišit dva často používané pojmy:

- Scheduling/plánování - soustředění na uspořádání objektů, se kterými pracujeme, např. pořadí operací.
- Timetabling/rozvrhování - soustředění na konkrétní časové umístnění, často vymezen časový horizont.

Při vysvětlování návrhu využití solveru, mluvíme o následujících pojmech:

- Model - nazýváme reprezentaci dat, včetně podmínek na tato data a optimalizační funkci.
  - Model constants (modelové konstanty) - jsou data na vstupu od uživatelů, která neměníme, ale na jejich základě vyhledáváme optimální řešení.

- Model variables (modelové proměnné) - jsou parametry/proměnné, se kterými solver pracuje, hledá jejich hodnotu takovou, aby byla hodnota cílové funkce co nejlepší (minimální nebo maximální podle definice modelu).
- Feasible solution (proveditelné nebo také použitelné řešení) - je řešení, které lze v praxi využít, protože neporušuje žádné pevné omezení (hard constraint).
- Hard constraints (pevná omezení) - jsou taková omezení, která jsou-li porušena, není výsledné řešení solveru bráno jako proveditelné/použitelné (feasible), jedná se například o podmínku, že nesmí být přiřazeno více kurzů, než je k dispozici místností. Všechna pevná omezení solver bere jako stejná, porušení jednoho pevného omezení je stejné jako porušení jiného pevného omezení.
- Soft constraints (mírná omezení) - jsou taková omezení, kde jejich porušení vede k proveditelnému/použitelnému (feasible) řešení, například přiřazení nepreferovaného času. Na rozdíl od pevných omezení, nemusí být mírná omezení jednotná. Porušení jednoho mírného omezení může vést k větší/menší změně cílové funkce, než porušení jiného mírného omezení.
- Objective function (cílová funkce) - minimalizace nebo maximalizace zadané funkce. Solver hledá takové hodnoty vstupních parametrů/proměnných, ve kterých tato funkce nabývá svého minima nebo maxima za předpokladu, že neporuší žádné hard constraints. Jednou z možných cílových funkcí je počítat počet porušených soft constraints a tuto hodnotu minimalizovat či maximalizovat.
- Relaxace řešení - v případě, kdy řešení ze solveru není feasible, je možné provést relaxaci řešení, solver potom najde řešení takové, kdy se sice poruší podmínky (hard nebo soft), ale poruší se jich co nejméně.

2.2 Metody řešení rozvrhování

Problém rozvrhování lze řešit více metodami. Uveďme některé příklady [4]:

- Přesné řešící metody - např. metoda větví a mezí,
- Heuristiky - např. řídící pravidla (dispatching rules), paprskové prohledávání (beam search) či lokální prohledávání,
- Matematické programování - např. lineární,
- Programování s omezujícími podmínkami,
- Celočíselné programování (integer linear programing),

- Metoda řezné roviny (cutting plane),
- Metoda větví a mezí (branch and bound),
- Hybridní metody.

2.3 Metoda programování s omezujícími podmínkami

Jedním z možných metod, jak řešit automatickou tvorbu rozvrhu je metoda programování s omezujícími podmínkami (constraint programming). Tato metoda spočívá v prohledávání prostoru možných řešení a nalezení takového proveditelného řešení (feasible solution), které neporuší žádnou ze stanovených podmínek. V případě tvorby rozvrhu je však potřeba propojit optimalizační problém (minimalizaci funkce) s nalezením feasible solution, nebo jeho co možná nejlepší aproximaci, tedy řešení takové, které poruší co možná nejméně podmínek.

2.4 Celočíselné programování (integer linear programing - ILP)

Na rozdíl od metody programování s omezujícími podmínkami je ILP kombinací lineárního programování s celočíselným programováním. To znamená, že můžeme definovat funkci, kterou chceme minimalizovat/maximalizovat a zároveň definovat podmínky, které musí být splněny, aby výsledné řešení rozvrhu bylo proveditelné (feasible). To standardně probíhá tak, že porušení soft podmínek je implementováno v cílové funkci. To nám dává možnost určovat, jakou váhu má jaké porušení soft podmínek. Implementace hard podmínek je potom implementována přes nerovnosti díky lineárnímu programování. Celočíselné programování má svůj název, jelikož všechny modelové proměnné mohou nabývat pouze hodnot z množiny celých čísel, tím se výrazně zmenšuje prohledávaný prostor. Prohledávaný prostor lze také omezit určením takzvaných dolních respektive horních mezí, které omezují definiční obor modelových proměnných zdola resp. shora.

Integer linear programing má standardní formu: [5]

$$\begin{aligned} \min c^T x \\ Ax = b \\ x \geq 0 \\ x_i \in \mathbb{Z}, \forall i \in I \end{aligned} \tag{2.1}$$

Kde minimalizujeme cílovou funkci.

Kde x jsou modelové proměnné.

Kde A je matice hodnot (skalárů).

Kde b a c je vektor hodnot (skalárů).

Kde I je množina indexů.

Speciální případ, kdy všechny proměnné jsou pouze hodnoty 0 či 1, se označuje jako "0 - 1 (integer) linear program". V případě, kdy bude daný kurz přiřazen danému času, tak označíme hodnotu $x = 1$, a naopak v případě, kdy nebude, označíme $x = 0$. Tedy upper bound je rovno jedné a lower bound je rovno nule. V práci bude využita tato metoda. Využití je více popsáno v kapitole 3.6.

2.5 Solver

Solver je software, který pomáhá řešit daný matematický problém, v případě této práce minimalizaci porušení vyhovujících časů a splnění omezujících podmínek. Jeho hlavní výhoda spočívá v tom, že není třeba určovat či implementovat jaký algoritmus prohledávání, v některých případech velkého prostoru, se použije. Z toho je odvozen název z anglického slova “solve”, vyřešit. Je nutné pouze definovat problém, odborně nazývaný model, který budeme po solveru chtít vyřešit. Pokud je model napsán správně, tak solver nalezne řešení, nebo oznámí, že najít optimální řešení není možné (problém není feasible). Tato situace nastává, podle autorky systému UniTime [11], při tvorbě rozvrhu často a je potřeba s ní počítat. V tomto případě pak lze relaxovat řešení, to znamená, že povolíme porušení podmínek a solver poté najde takové řešení s nejmenším množstvím porušených podmínek. (V případě této práce nastavíme ať solver přiřadí pouze ty hodnoty, které neporuší žádné hard podmínky. Z toho vyplývá, že v tomto případě některé kurzy nebudou přiřazeny.)

2.6 Volba solveru

Solverů je nepřeberné množství, představme si tři, se kterými jsem se při tvorbě rešerše této bakalářské práce setkal.

2.6.1 OptaPlanner

Optaplanner [6] je constrain programming solver. Jedná se o open source. Tento solver umožňuje popisovat optimalizační problémy v programovacím jazyce Java. Na stránkách autoři uvádí příklady využití: plánování cest vozidel, řízení zaměstnanců, optimalizaci využití cloudových služeb a podobně. Poskytuje velké množství dokumentace s rozsáhlým souborem vysvětlivek a popisů (modelových nebo matematických), avšak nikde není konkrétní

ukázka s dostatečným vysvětlením, jak daný software přesně použít. Nabízí sice sbírku ukázek, ale jedná se pouze o souhrn mnoha souborů bez jakýchkoliv komentářů, která by vedla k pochopení prezentovaného řešení.

2.6.2 Cbc

Jako další možnosti solveru jsem zvažoval open source solver Cbc [7], jedná se již o mixed integer programming solver (MILP). (Integer solver, který dovoluje pracovat i s jinými daty než pouze celými čísly.) Napsaný v C++, u volby tohoto solveru mě odradila skutečnost, že pro výpočet složitějších problémů solver hledá řešení v řádech minut [8].

2.6.3 Gurobi

Nakonec jsem se rozhodl pro solver, který není open source, ale na který mám k dispozici, díky ČVUT, studentskou licenci, solver Gurobi [9]. Solver Gurobi nabízí velkou podporu řešení problémů. Jedná se o Linear Programming (LP), Mixed-Integer Linear Programming (MILP), Quadratic Programming (QP), Mixed-Integer Quadratic Programming (MIQP), Quadratically Constrained Programming (QCP) a Mixed-Integer Quadratically Constrained Programming (MIQCP). Z těchto možností však využijeme pouze MILP. V této volbě mě utvrdila i tato publikace [10], kde je solver Gurobi časově hodnocen lépe než solver Cbc.

2.7 Systém UniTime

Jedním ze systémů, který řeší tuto problematiku je systém UniTime [11]. Jedná se o open source software, napsaný v Javě. Hlavním cílem systému je přiřadit časy a jednotlivé konkrétní místnosti kurzům pro univerzitní studenty s různými existujícími podmínkami. Poskytuje rozhraní pro vstup a manipulaci dat a jejich vizuální reprezentaci. Jedná se o velmi ambiciózní projekt, který byl vyvíjen řadu let (jeho počátky lze zaznamenat v roce 2008), a stále se na tomto softwaru aktivně pracuje [13]. Použití systému na Masarykově universitě je popsán v master thesis [12].

Systém je nyní aktivně využíván Purdue University [14]. Systém na této univerzitě využívá minimálně přes 41 tisíc studentů podle zprávy [15].

Pro nasazení systému je zapotřebí Java Development Kit [16], Apache Tomcat [17] a MySQL [18]. Jelikož všechny zmíněné systémy lze použít na hlavních operačních systémech (Linux, Windows, Mac), lze UniTime nasadit na tyto systémy.

Motivace pro vznik UniTime:

- Minimalizovat konflikt ve studentských rozvrzích pro včasné dokončení studia.
- Využívat omezené zdroje více efektivněji (obsadit více místností rovnoměrněji než např. vystavět nové třídy, aby se daly využívat ve stejný čas) a s tím je spojena i úspora financí za údržbu, jako například za vytápění a podobně.
- Sběr dat o využití, např. pro nového "rozvrháře", nebo pro budoucí plánování či zefektivnění výuky.
- Možnost zkoumat různé nové scénáře, např. když je zavřena jedna budova kvůli havárii, nebo při úpravě harmonogramu nějakého studijního programu, v návaznosti jeho úpravy na využití místností.

UniTime je rozdělen na 4 samostatné části:

- Course timetabling,
- Examination timetabling,
- Student Scheduling,
- Event management.

Tyto části se zpracovávají odděleně, a přestože jsou některé spolu provázány, jako např. Course Timetabling a Student Scheduling, oba řeší odlišný problém. Jmenovitě přiřazení času kurzu a přiřazení studentů do různých tříd probíhajících v době kurzu.

Sběr dat a možnosti vstupů:

Data o místnostech, instruktorech, požadavcích jednotlivých kurzů je možné sesbírat z různých oddělení, nebo také z jednoho centrálního zdroje. Nejpodstatnější jsou data o kurzech, kde se vkládá do systému, jak dlouho kurz trvá, jaké má požadavky a kolik osob se kurzu bude účastnit.

Kurzy se mohou párovat, což znamená, že v případě, kdy si student zvolí jeden předmět, nemůže navštěvovat druhý.

Dalším způsobem vkládaní dat je použití předchozího roku, jako vzor pro rok následující. Lze také nechat sesbírat od studentů jejich požadavky, jaké předměty a časy preferují.

Data od Instruktorů obsahují především: co budou učit, jejich osobní preferované časy a dostupnost.

Každé oddělení má seznam instruktorů, kteří v něm učí. Instruktoři mohou učit i v jiných odděleních, systém UniTime zkontroluje, aby časy, kdy učí, se neprolínaly napříč jednotlivými odděleními.

Také lze nastavit speciální podmínky, jako například: uživatel může učit maximálně 5 hodin denně a jiné.

Sbíraná data o místnostech: maximální počet míst, vybavení místnosti a v jaké budově se místnost nachází. Také se sbírají data o vzdálenostech mezi jednotlivými místnostmi. Toto se řeší přes souřadnice místností, nebo přes proměnou travel times, která určuje v minutách, jak dlouho trvá přesun z jedné místnosti do druhé.

Na místnosti nebo skupinu místností lze dávat požadavek, případně preferenci.

Mezi skupiny místností může patřit např. cvičebna, přenášecí místnost, místnost s počítači a další. Při hledání optimální místnosti se musí počítat s minimální velikostí dle počtu studentů.

Místnosti jsou následně přiřazeny jednotlivým oddělením. Lze nastavit, aby jedna daná místnost byla dostupná pro jedno oddělení pouze v daném čase a v daném dni.

Hlavní tři skupiny preferencí jsou na:

- místnost/budovu,
- skupinu místností - cvičebnu, počítačovou místnost podobně,
- podle vlastností/feature.

Vztahy mezi jednotlivými kurzy lze nastavit podle "které musí být":

- po sobě,
- ve stejné místnosti,
- ve stejný den,
- maximálně X hodin denně,
- mohou sdílet stejnou místnost (např. volné laboratoře bez cvičícího),
- velmi dlouhý seznam dalších podmínek.

Studenti mají jiné podmínky než kurzy (musí naplnit svůj studijní plán, předměty jim musí navazovat na sebe).

Autoři UniTime definují takzvané Data patterns, která určují speciální případy, kdy není kurz každý týden, ale jeho frekvence se liší. Např., jestli nějaký kurz bude jenom každý lichý či sudý týden, každý 5. týden, nebo jen 2 krát za celý semestr.

K hodnocení preferovanosti času je používáno 7 úrovní. Mezi které patří např. "strongly prefered", "prefered" a další. Také "required" respektive "prohibited", které určují nutnou podmínku, že kurz musí být ve stanovený čas, respektive nesmí, tyto informace jsou hlavně potřebné pro algoritmus tvorby rozvrhu, který se podle nich řídí.

Autoři systému UniTime se zmiňují o potřebě relevantních vstupních dat. Pokud jsou data na vstupu nepřesná nebo chybná, použití sebelepšího algoritmu, nevytvoří dobrý rozvrh.

Mezi chyby ve vstupních datech:

- Žádné preference - může dojít ke stanovení kurzu ve špatném čase, dále např. přiřazení špatné místnosti s nedostatečným vybavením.
- Velké množství preferencí - např. pouze v jednu hodinu, v jeden den, z tohoto důvodu není možné vytvořit *úplný rozvrh*, dále může dojít u mnoho studentů ke konfliktu se třídami.
- Zneužití systému uživateli - uživatel úmyslně nastaví sebou preferované časy jako časy dostupné, a ostatní nastaví jako nedostupné.

Po vstupech dat systém umožňuje:

- Vytvoření rozvrhu automaticky.
- Vytvoření rozvrhu manuálně.
- Vytvoření rozvrhu automaticky a poté dodělat úpravy manuálně (doporučený postup).

Automatická tvorba rozvrhu:

Při tvorbě rozvrhu se Solver (definice viz 2.5) snaží minimalizovat Hard/Soft podmínky.

Hard podmínky jsou takové, které systém předpokládá, že nelze porušit a snaží se nejprve najít takové řešení, které je neporuší. Mezi tyto hard podmínky patří: velikost místnosti (například kurz o velikosti 100 studentů nemůže být v místnosti s kapacitou 50), možnost sdílení místnosti, dostupnost (jestli lze danou místnost v daný čas použít) a také “required”, “prohibited” časy v preferencích uživatelů (viz popis výše).

Soft podmínky jsou takové, u kterých se počítá s jejich porušením: Konflikty v rozvrhu studentů, kdy kurz není optimální, ale stále se bere jako použitelný. Např. jedna hodina velmi brzo ráno a následující ve večerních hodinách. Přiřazení nepreferovaných/málo preferovaných místností a časů. A mnoho dalších, např. zda je preferované, aby místnost s velkou kapacitou měla nižší prioritu než ta, která více odpovídá počtu studentů, nebo když vyučující učí ihned po sobě dva kurzy tak, aby místnosti nebyly příliš vzdálené od sebe.

Kroky solveru samotného:

1. Solver musí zjistit, jestli se vůbec dá problém vyřešit, jinými slovy, jestli máme dost kapacit na to, naplnit dostupné třídy v časech, které máme k dispozici - v této fázi solver bere v potaz pouze hard podmínky.
2. Pokud nelze rozvrh vytvořit, musíme upravit vstupní data - UniTime k tomu poskytuje rozhraní, které dovolí vyhledat konflikty.
3. Poté máme možnost nechat vytvořit rozvrh, zkontrolovat a podívat se na případné chyby. Vstupní data nemusí být v tomto případě kompletní, následně lze rozvrh ručně upravit a nechat systém vytvořit upravený rozvrh znovu. Tento krok probíhá v tolika iteracích, kolik je potřeba.

Krok 3 je doprovázen statistikou počtu konfliktů, naplnění místností a také možností, podívat se na rozvrh z pohledu studentů, tříd nebo učitelů.

4. Posledním krokem je udělat manuální změny v systému - ty jsou podle autorů v praxi často potřeba, UniTime nabízí možnosti nápovědy, kam přesunout daný kurz, respektive na jaký čas, dokonce nabízí i nápovědu na prohození s jiným kurzem tak, aby změna nevyvolala žádné nové kolize.

K tomu jsou k dispozici 3 možnosti:

- Minimal perturbation mode - když jde o velké masivní změny, v podstatě se přepracuje celý rozvrh, příklad: poškození místnosti - upravuje solver automaticky.
- Interaktivní mód - dovoluje porušovat soft podmínky tak, aby se dosáhlo na vyhovující rozvrh, nebo také hard podmínky, když vedoucímu kurzu dojde ke změně harmonogramu v době, kdy byl dříve nedostupný a již dostupný je - tuto změnu upravuje uživatel.
- Po zveřejnění systému - upravují správci systému a již neprobíhá kontrola studentských konfliktů ani časů dostupností lektorů - počítá se s tím, že uživatel provádějící změnu vše zkontroloval.

Solver může řešit pouze části některých oddělení, každé oddělení si tak může udělat vlastní rozvrh s pomocí solveru, a nebo existuje i společný režim, kde si jedno oddělení udělá vlastní rozvrh a následně si další jiné oddělení tyto výsledky převezme (vytvořené již solverem) a postaví na nich svůj rozvrh.

Další možností je režim, kde se vytvoří společně rozvrh pro všechny, a ten si poté mohou jednotlivá oddělení upravovat, ale pouze ty kurzy a místnosti, ke kterým mají přístup.

Poslední možností je vytvoření rozvrhu omezeného jen pro jeden typ místnosti, v praxi to jsou např. velké místnosti (přednášecí místnosti). Většinou je tím pověřena speciální osoba, která má na starost správné rozložení místností.

Vytvořený rozvrh systém umožňuje exportovat do externích programů. Lze použít data z předchozích let, aby se zamezilo nutnosti každý rok znovu vkládat všechna vstupní data.

Role:

- Guest - pouze vidí data, nemůže nic měnit.
- Dept - schedule manažer systému - může editovat svá oddělení, ostatní oddělení jsou editovatelná pro tuto roli pouze v případě, že se jedná o vstup dat (při vytváření oddělení), poté se pravomoce nastaví na jinou osobu.
- Llr - (large lecture room manager) - může editovat všechny místnosti s určitým označením “flagem”, např. velké místnosti.

- Exam manager - upravuje pouze data ze zkoušek a jejich výsledky (nepracuje na tvorbě rozvrhu).
- Event manager - přidává, upravuje, schvaluje nebo odmítá jednorázové akce, ty nejsou součástí tvorby rozvrhu, ale zabírají místo pouze v určitých časech.
- Admin - může upravovat veškerá data.
- Instruktoři - mají přístup jenom ke svým (personifikovaným) rozvrhům a ke svým schváleným/neschváleným/odmítnutým akcím, také mohou dávat požadavek na jednorázové akce (např. na termín zkoušky), které pak musí event manager schválit.
- Student - vidí jen svůj rozvrh, může také dávat požadavek na speciální akce, které schvaluje buď administrátor (dept) nebo event manager.

Kritické hodnocení pro potřebu této práce:

Systém je svým rozsahem a možnostmi velmi nepřehledný pro nového uživatele, systém nelze použít na jiné účely než vytváření rozvrhu pro studenty. Na druhou stranu nabízí velkou možnost modifikace rozvrhu za provozu, k tomu využívá speciálně upravený solver, který pomáhá s automatickou tvorbou rozvrhu, zároveň umožňuje ruční manipulaci pro upřesnění výsledného rozvrhu.

Kapitola 3

Návrh systému

3.1 Uživatelé

Před tvorbou návrhu systému je vhodné zauvažovat nad cílovou skupinou, která bude systém využívat. Pro naše potřeby se bude jednat o administrátory. Ti budou mít na starosti práci se systémem a budou zastávat roli ”rozvrhářů” pro jednotlivé rozvrhy (Timetable). Druhou podstatnou skupinou jsou lektoři nebo vedoucí, osoby, které vedou kurzy, které požadují časový úsek v rozvrhu. Třetí a poslední skupinou jsou účastníci kurzů, kteří mají k sobě přiřazené kurzy, ti mají pouze možnost zobrazovat svůj rozvrh.

3.2 Přístup k systému

Jelikož se jedná o systém pro více uživatelů, předpokládejme, že uživatelé budou chtít využívat svoje zařízení pro zobrazení a manipulaci s rozvrhem. Proto je nutné navrhnout server-klient systém, systém s jednou centrální jednotkou, která drží data a odesílá je dle potřeby jednotlivým klientům [21]. Server bude webový a přístup k systému bude přes klasickou webovou stránku.

3.3 Funkcionalita

Představme návrh složitějšího rozsáhlejšího systému, který bude následně implementován v základní pilotní podobě. Funkcionalitu popíšeme pomocí akcí nebo činností, které lze provádět na jednotlivých částech systému.

Vstup uživatelů do systému:

- Registrace nového uživatele do systému.
- Přihlášení registrovaného do aplikace.

- Odhlášení přihlášeného uživatele.

Vkládání dat:

- Tvorba/přidání rozvrhu, obecně se musí nastavit pro jaké dny se bude rozvrh vytvářet, v kolik hodin bude rozvrh začínat, jak dlouho budou trvat časové úseky (sloty) pro přiřazení kurzu, jak dlouhá bude pauza a kolik časových úseků bude pro každý den. Pro zjednodušení počítáme, že každý den začíná ve stejnou dobu a má stejný počet časových slotů.
- Tvorba/přidání typů místností pro následnou tvorbu rozvrhu, musíme mít nejprve možnost vytvořit jednotlivé typy místností.
- Tvorba/přidání kurzů, zde budou vstupní informace, kdo vede kurz, jaký uživatel, jaký typ místnosti kurz využívá a do jakého rozvrhu kurz patří.
- Tvorba/přidání konkrétních místností, každá místnost má určitý typ, který zde nastavíme.
- Testy - do této kategorie začleníme také možnost načtení testových vstupů, které by v eventuálním ostrém provozu sloužily jako demonstrace systému.

Editace dat - po vložení dat bude k dispozici rozhraní pro jejich editaci, a to jednotlivě:

- Editace typů místností, včetně jejich počtu pro jednotlivé časy, počet místností, které jsou k dispozici a v jaký čas a také jejich odstraňování.
- Editace kurzů, změna uživatele, který kurz vede, změna typu místnosti, která je pro kurz potřeba, změna do jakého rozvrhu kurz spadá, a také jejich odstraňování.
- Editace místností, změna názvu či změna jejich typu, a také jejich odstraňování.
- Editace uživatelské dostupnosti, každý uživatel bude mít možnost nastavit v jaké časy je dostupný pro všechny časy ve vytvořených rozvrzích.
- Editace uživatelských preferencí, každý uživatel bude mít možnost nastavit jaké časy preferuje pro svoje kurzy pro všechny časy ve vytvořených rozvrzích.

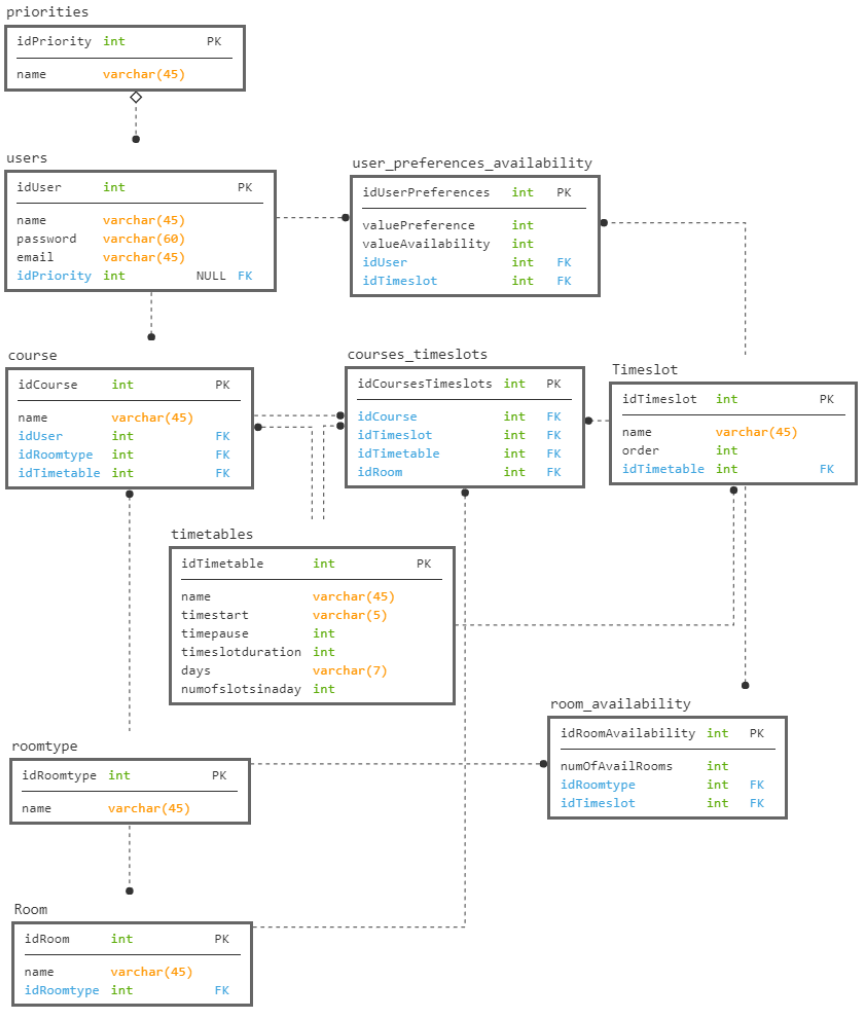
Využití solveru - musíme být schopni zadat příkaz k vyřešení rozvrhového problému, který solver vyřeší a načte výsledná data. Navrhneme tento krok tak, aby se vždy vytvářel pouze jeden zvolený rozvrh, protože budeme předpokládat, že rozvrhů může být více. Pro zjednodušení nebudeme předpokládat, že jeden lektor bude mít více požadavků na více kurzů ve více rozvrzích, ale že bude využívat jenom jeden rozvrh. Lze si to představit např. tak, že jeden rozvrh je vždy pro jedno dané oddělení.

Manipulace s vytvořeným rozvrhem - nejpodstatnější částí je tvorba rozhraní pro reprezentaci rozvrhu (tabulka), ve které lze manipulovat s kurzy, přiřazovat

jim jednotlivé časy a konkrétní místnosti.

3.4 Návrh databáze

Jako databázovou část aplikace lze využít více technologií, zvolme SQL databázi, oproti jiným přístupům např. NoSQL MongoDB, která využívá k reprezentaci javascriptové objekty key-pair value, z důvodu rychlosti čtení a celkovému návrhu aplikace, kde se očekává více dotazů na čtení než na zápis na což je SQL databáze rychlejší než zmiňované jiné technologie [22]. Použijme stejnou terminologii, kterou používá NPM knihovna sequelize, aby bylo jasné až rozebereme implementaci v dalších kapitolách. K vizuální reprezentaci použité SQL databáze byl použit Modeler SqlDBM [23], viz obrázek níže.



Obrázek 3.1: Databázový návrh

Popišme jednotlivé tabulky a data, která reprezentují. Každá tabulka má

stejnojmenný index ve tvaru id + název tabulky, ten slouží na rychlé vyhledávání. Zkraťme zápis využitím zkratek, kde cizí klíč (foreign key) značme jako FK. Řetězec (String) značme Str. Číslo (Integer) značme Int.

Kurzy (Courses)

- jméno (name) - Str sloužící pouze jako identifikátor,
- FK User - uživatel je žadatelem časového slotu pro tento kurz,
- FK Roomtype - typ místnosti, který tento kurz bude využívat,
- FK Timetable - rozvrh, do kterého se bude časový slot zařazovat,
- FK Room - místnost, která je přiřazena danému kurzu.

Uživatelé (Users)

- jméno (name) - Str sloužící jako identifikátor a zároveň login při přihlašování do systému,
- heslo (password) - Str zahashovaný (se solí), více v kapitole o bezpečnosti 4.7, sloužící k ověření přihlášení,
- email (email) - v budoucí verzi systému může sloužit k obnově hesla,
- FK priority - priorita sloužící k identifikaci, jaké pravomoci uživatel má, tedy k jaké části serveru má přístup.

Priority/Pravomoce (Priorities)

- jméno (name) - Str sloužící pouze jako identifikátor.

Časové sloty (Timeslots)

- jméno (name) - Str sloužící pouze jako identifikátor ve tvaru den + číslo slotu v daném dni indexovaným od nuly, např. MO2 (Pondělí 3.slot),
- pořadí (order) - slouží k určení, kam vložit do tabulky daný slot pro rozvrh, indexován od 0 do počet slotů ve dni krát počet dnů,
- FK Course - kurz přiřazený tomuto časovému slotu.

Rozvrhy (Timetables)

- jméno (name) - Str sloužící pouze jako identifikátor,
- počáteční čas (timestart) - Str, čas kdy rozvrh začíná ve standardním formátu “XX:XX” např. “08:00”,
- délka pauzy (timepause) - Int v minutách, určuje délku pauzy mezi jednotlivými kurzy,
- dny (days) - Str, dny, kdy bude rozvrh dostupný jsou ve formátu pondělí až neděle 7 po sobě jdoucích čísel, které značí 1 - den je v rozvrhu, 0 - den není v rozvrhu, např. všechny dny kromě úterý jsou označeny jako “1011111”,
- počet časových bloků v dni (numofslotsinaday) - Int, počet časových bloků v dni (pro všechny dny jsou stejné).

Typy místností (Roomtypes)

- jméno (name) - Str sloužící pouze jako identifikátor.

Místnosti (Rooms)

- jméno (name) - Str sloužící pouze jako identifikátor,
- FK roomtype - slouží k určení jakého typu je místnost.

Uživatelské preference a dostupnosti (UserPreferencesAvailibilities)

- hodnota preference (valuePreference) - Int, hodnota buď 0 - daný časový slot daný uživatel nepreferuje nebo 1 - daný uživatel daný časový slot preferuje,
- hodnota dostupnosti (valueAvailability) - Int, hodnota buď 0 - daný uživatel je v daný časový slot nedostupný nebo nebo 1 - daný uživatel je v daný časový slot dostupný,
- FK user - daný uživatel,
- FK timeslot - daný časový slot.

Dostupnosti místností (RoomAvailability)

- počet dostupných místností (numOfAvailRooms) - Int, celkový počet dostupných místností daného typu v daném časovém slotu,

- FK roomtype - daný typ místnosti,
- FK Timeslot - daný časový slot.

Kurzy-Časové sloty(CoursesTimeslots)

- Přiřadí danému kurzu, u daného rozvrhu daný časový slot,
- FK Course - daný kurz,
- FK timeslot - daný časový slot,
- FK timetable - daný rozvrh.

3.5 Vstupní data a pojmy pro návrh funkce solveru

Pro automatickou tvorbu rozvrhu je potřeba nejprve určit, jaká vstupní data se budou shromažďovat pro práci solveru, nebo také, co se myslí danými pojmy a jak bude vypadat výstup v aplikaci. Přesnou implementaci definujeme v pozdějších kapitolách. Prozatím budeme předpokládat, že pracujeme s těmito daty:

- Časy (nazývané také jako sloty) - základní stavební kámen rozvrhu, jednotka, do které lze přiřadit kurz.
- Kurz (course) - kurz, který přiřazujeme k nějakému časovému slotu, je vytvořen uživatelem a může mít různé podmínky (v tomto návrhu počítejme pouze s jednou podmínkou - typem místnosti), jeden kurz může mít pouze jeden typ místnosti.
- Typ místnosti (roomtype) - typ místnosti, který popisuje určitou skupinu místností podle společných znaků, např. konferenční místnost.
- Rozvrh (timetable) - jedná se o sadu časových slotů, logicky označených, aby bylo zřejmé, kdy jaký slot trvá (den a čas) a jaký kurz je pro jaký slot přiřazený.
- Uživatel (user) - vytváří kurzy, ke kterým chce přiřadit časový slot v rozvrhu.
- Preferovaný čas uživatele (preference) - vlastnost každého uživatele pro každý časový slot - sděluje nám informaci, kdy uživatel chce mít vedený kurz v rozvrhu.
- Dostupný čas uživatele (user availability) - vlastnost každého uživatele pro každý časový slot - sděluje nám informaci, kdy uživatel může mít vedený kurz v rozvrhu.

- Dostupnost místností (room availability) - vlastnost každého časového slotu pro každý typ místnosti - číslo, které určuje kolik místností daného typu je k dispozici v daný čas.

Nyní si popišme konkrétní funkce, jejich parametry a hodnoty, které funkce vrací:

- courseUser(course) - Vstupním parametrem je kurz (course). Hodnotou, kterou vrací, je uživatel (user), který vede daný kurz.
- userAvail(user,timeslot) - Prvním vstupním parametrem je uživatel (user), druhým pak časový slot. Hodnota, kterou vrací, je buď 1 - a to v případě, že daný uživatel je v daný časový slot dostupný, 0 - v případě, že není dostupný.
- userPref(user,timeslot) - Prvním vstupním parametrem je uživatel (user), druhým pak časový slot. Hodnota, kterou vrací, je buď 1 - a to v případě, že daný uživatel daný časový slot preferuje, 0 - v případě, že nepreferuje.
- courseRoomtype(course) - Vstupním parametrem je kurz (course), hodnota, kterou vrací, je typ místnosti (roomtype) pro daný kurz.
- roomAvail(roomtype,timeslot) - Prvním vstupním parametrem je typ místnosti (roomtype), druhým pak časový slot. Hodnota, kterou vrací, je číslo počtu dostupných místností daného typu v daný čas.

3.6 Návrh funkce solveru

Jak jsme již zmínili v předcházejících kapitolách, pro automatickou tvorbu rozvrhu využijeme metodu celočíselného lineárního programování (ILP). Díky použití solveru odpadá nutnost řešit jaký algoritmus na řešení problému použít, stačí pouze definovat správný model. Návrh modelu je založený na řešení prezentované v publikaci [8]. Tato publikace je inspirovaná dalšími pracemi [19]. Přestože se zmíněné publikace věnují TTP z pohledu čistě pro university, je možné definice upravit, aby ji bylo možno použít na řešení obecného TTP, které budeme prezentovat zde. Zmíníme, že jeden model se vždy řeší pro jeden konkrétní rozvrh (Timetable).

Popišme slovy, jak budeme daný problém řešit a následně popišme jeho matematickou notaci. Budeme chtít přiřadit každému uživateli, který má vytvořený kurz nějaký čas. Každý kurz přiřadíme právě jednou. Nebude nás zajímat do jaké konkrétní místnosti v přiřazený čas bude daný kurz patřit, postačí nám stačit ověřit, aby bylo dost dostupných místností určitého typu v daný čas, kdy kurz přiřazujeme. V přiřazený čas musí být uživatel, který ho vede dostupný, také nesmí být jeden stejný uživatel přiřazen do více různých kurzů ve stejný čas. Časy budeme přiřazovat tak, aby co nejvíce kurzů bylo přiřazeno v časech preferovaných uživateli, kteří ho vedou.

3.6.1 Modelové proměnné

Když kurz c je přiřazen časovému slotu s potom platí:

$$X_{s,c} = 1 \tag{3.1}$$

Když kurz c není přiřazen časovému slotu s potom platí:

$$X_{s,c} = 0 \tag{3.2}$$

3.6.2 Model Cílové funkce

$$\min \sum_{s \in Slots} \sum_{c \in Courses} (userAvail(courseUser(c), s) - userPref(courseUser(c), s)) \cdot X_{s,c} \tag{3.3}$$

Kde *Slots* je množina všech časů a *Courses* je množina všech kurzů v jednom rozvrhu. Tuto cílovou funkci se budeme snažit v naše problému minimalizovat. Popišme slovy, co funkce dělá: Jedná se o sumu přes všechny kurzy a všechny časové sloty. Pokud pro daný čas není přiřazen daný kurz, je X = 0, tudíž hodnota pro tuto iteraci je nula. Pokud je daný kurz přiřazený danému času, potom je X = 1, potom nastávají čtyři možnosti, jmenovitě:

- Daný uživatel je v daný čas dostupný, daný uživatel daný čas preferuje: jedná se o rovnici 1 - 1 = 0, toto je optimální přiřazení (hodnota cílové funkce se nezvýší).
- Daný uživatel není v daný čas dostupný, daný uživatel daný čas preferuje: tato situace by neměla nikdy nastat, musíme jí ošetřit podmínkou (Podmínka 2 viz 3.5).
- Daný uživatel je v daný čas dostupný, daný uživatel daný čas nepreferuje: jedná se o rovnici 1 - 0 = 1, toto je validní hodnota funkce, (hodnota cílové funkce se zvýší), jedná se o porušení preferovaného času, které se snaží solver minimalizovat.
- Daný uživatel není v daný čas dostupný, daný uživatel daný čas nepreferuje: opět tato situace by neměla nikdy nastat, musíme jí ošetřit podmínkou (Podmínka 2 viz 3.5).

Čím větší je hodnota této funkce, tím horší (méně optimální) je rozvrh, který vytváříme. Více uživatelů má zvolené nepreferované časy, avšak stále se jedná o nejlepší možné řešení.

3.6.3 Modelové podmínky

Podmínka 1:
Každý kurz je přiřazen právě jednou.

$$\forall c \in courses : \sum_{s \in slots} X_{s,c} = 1 \tag{3.4}$$

Pro budoucí vývoj aplikace, v případě, kdy bude potřeba přiřadit v jeden čas více kurzů než jeden, stačí nahradit pravou stranu rovnice požadovaným číslem/funkcí.

Podmínka 2:
Každý uživatel je dostupný v přiřazeném čase.

$$\forall c \in courses, \forall s \in timeslots : \sum_{c \in courses, courseUser(c)=u} X_{s,c} \leq userAvail(courseUser(c), s) \tag{3.5}$$

u v rovnici značí jednoho z možných uživatelů

Podmínka 3:
Je k dispozici dostatečný počet místností, daného typu v čase přiřazeném danému kurzu.

$$\forall c \in courses, \forall s \in timeslots : \sum_{c \in courses, courseRoomtype(c)=rt} 1 \cdot X_{s,c} \leq roomAvail(courseRoomtype(c), s) \tag{3.6}$$

rt v rovnici značí jeden z možných typů místností (roomtype)

Jednička na levé straně rovnice představuje číslo, kolik místností je potřeba pro daný kurz v daném čase. (Pro zjednodušení budeme předpokládat, že každý kurz požaduje jednu místnost.)

Podmínka 4:
Pokud je daný kurz v daném čase přiřazen danému uživateli, který ho vede, potom ten stejný uživatel nesmí být přiřazen ve stejný čas u žádného jiného kurzu, který také vede.

$$\begin{aligned} &\forall c \in courses, \forall s \in timeslots : \\ &když(X_{s,c} = 1) potom : \\ &\forall c2 \in courses, \forall s2 \in timeslots : \\ &když((courseUser(c) = courseUser(c2)) \wedge (c \neq c2)) potom : \\ &X_{s,c} = 0 \end{aligned} \tag{3.7}$$

Kapitola 4

Implementace

4.1 Úvod

Implementovaný systém řeší výše popsanou problematiku s tvorbou rozvrhu, vstup dat a jejich reprezentaci v databázi a jejich následnou manipulaci. Používá zmíněný model solveru na tvorbu rozvrhu automaticky. Používá také řadu technologií, které jsou popsané v následujících kapitolách. V kapitole 4.11 lze najít výčet všech potřebných kroků pro jeho nasazení. Tato pilotní verze softwaru řeší problém z pohledu administrátorů, to znamená, že přiřazujeme kurzy jenom takovým uživatelům, kteří tyto kurzy vedou. Přidávání uživatelů, kteří tento kurz pouze navštěvují implementováno není.

Aby byl jednoznačně odlišený kód, který byl napsán pro potřeby této bakalářské práce od souborů importovaných (NPM knihoven popsaných v kapitole 4.2.3), využijme obrázku souborového systému 4.5. Všechny soubory, kromě souborů ve složce *node_modules* značící NPM knihovny, byly napsány v rámci této práce. Soubory jsou popsány v kapitole o programátorské dokumentaci v sekci implementované soubory 4.10.1.

4.2 Back end

Pojem Back end označuje část vývoje softwaru na serverové straně, tedy logiku řízení systému uvnitř a manipulaci s daty, jejich ukládání, upravování a odesílání zpět ke klientům.

4.2.1 Nodejs

Pro tvorbu systému byl využit server Nodejs [24], jedná se moderní runtime, prostředí postavené na Chrome V8 JavaScript engine, které umožňuje spuštění javascriptového kódu mimo prohlížeč, je obohacené o mnoho metod

potřebných pro běh serveru, které nejsou v prohlížecích implementovány kvůli bezpečnosti, například práce se souborovým systémem. Jedním z jeho výhod oproti konkurenci je využití neblokujícího asynchronního provádění kódu, díky tomu se nevytváří nové vlákno pro každé nové připojení, ale používá se request queue. Při každém novém requestu se registruje takzvaný *callback*, který se obslouží teprve tehdy, když je jeho operace hotová, tj. dojde tento *callback* do request queue na řadu a je zpracován. Díky tomuto přístupu je jakákoliv (serverová) aplikace napsaná v nodejs velmi responzivní a dobře škálovatelná. Přináší to však i jisté nevýhody, jelikož se využívá pouze jedno vlákno, nelze využít výkon celého procesoru (za předpokladu, že máme více jádrový procesor). Nodejs je bezpečný, jelikož vývojáři aktivně pracují na nových verzích a opravují objevené bezpečností chyby. Nodejs má také velkou podporu ze strany komunity a s tím souvisí i jeho Node package manager.

4.2.2 Express

Nodejs Express [25] je webový framework, který značně zjednodušuje vývoj v Nodejs, představením takzvaných Middleware funkcí. Middleware funkce úzce souvisí s průběhem programu v NodeJs, kde při přijetí libovolného HTTP requestu tento request protéká serií předem definovaných funkcí, nazývaných middleware, dokud jeden z těchto předem definovaných middleware tento tok neukončí a to tak, že odešle zpět odpověď. Middleware má 3 parametry, jedním z nich je request samotný obsahující data přijatá od uživatele. Response objekt, který slouží jako reference na odesílání odpovědi uživateli a next značící následující middleware. Díky této architektuře lze jednoduše obsloužit příkaz na správném místě.

Jako příklad poslouží následující funkce implementovaná v systému v souboru system/routes/redirections.js:

```
//redirects not logged user to login page
redirectLogin = (req, res, next) => {
    if (!sessionController.isLogged(req)) {
        res.redirect('/login');
    }
    else {
        next();
    }
};
```

Funkce má tři výše popsané argumenty, req (request), res (response) a next. Při zavolání této funkce se tedy zkontroluje, zda-li je už uživatel přihlášen, pokud není, tak se odešle odpověď ve formě přesměrování na stránku s přihlášením, tím se tok ukončí a dotaz je uzavřen. Pokud uživatel je přihlášený, tak se zavolá další (next) middleware, který obslouží tento dotaz, nebo případně zavolá znovu další middleware.

4.2.3 Node package manager (NPM)

Node package manager (NPM) [26] řídí správu knihoven pro Nodejs, skládá se ze 2 hlavních částí, command-line klienta a internetové databáze knihoven. V této databázi se nachází několik stovek tisíc knihoven, které poskytují různorodou přídavnou funkcionalitu pro Nodejs. Verze a stahování těchto knihoven řídí NPM sám, verze lze však manuálně nastavit v .json souboru, který NPM vytvoří/upraví při stažení nové knihovny. Novou knihovnu lze stáhnout velmi lehce, právě přes zmiňovaný command-line jednoduchým příkazem *'npm install nazevknihovny'*. Je vhodné zmínit, že řada dříve externě implementovaných knihoven byla začleněna do Nodejs Expressu či do jádra Nodejs a není již potřeba je stahovat přes NPM. Tyto knihovny jsou uvedeny pro úplnost.

4.2.4 Využité NPM knhovny

Bcrypt

Bcrypt [27] je knihovna sloužící k hashování a porovnávání hashů, v případě této práce je použita na autentifikace, přesněji na hashovaní a porovnávání hashovaných hesel v databázi. Více je tato problematika rozebrána v kapitole bezpečnost 4.7.

Ejs

Ejs [28] je knihovna pro front end, slouží pro dynamickou-generaci HTML dokumentu při odesílání odpovědi na request (dotaz). Více u kapitoly front end 4.3.

Ejs-lint

Ejs-lint [29] je nově upravený kontrolor syntaxe pro ejs, originální ejs syntax kontrolor, který se nainstaluje s instalací knihovny Ejs, ten v některých případech poskytuje velmi nekonkrétní chybové výpisy. Ejs-lint tyto výpisy více rozvádí.

Express-session

Express-session [30] je knihovna navržena pro práci s express. Slouží k manipulaci a výrobě takzvaných sessions, což je část paměti, která je držena na serverové straně a uživatel do ní nemá přístup, vždy jeden session je spojený s jedním uživatelem a lze zde ukládat data, to může sloužit např. k ověřování,

zda je uživatel přihlášený, jelikož nehrozí, že uživatel svým dotazem zasáhne do paměti vyhrazené pro session.

Moment

Moment [31] knihovna slouží pro manipulaci s časem. Je použita pro výpočet délky trvání časových slotů v tabulce zobrazující rozvrh viz 4.10.1.

Nodemon

Knihovna Nodemon [32] slouží na automatické restartování serveru při změně v libovolném javascriptovém souboru v kořenovém adresáři, kde sídlí NodeJs systém. Velmi zpříjemňuje práci, jelikož není nutnost při každé změně ručně zapnout a vypnout server, aby se projevily změny. Tato knihovna není zamýšlena pro použití v produkci, ale pouze pro vývoj softwaru.

Sequelize

Sequelize [33] slouží pro komunikaci s databází a také pro návrh vnitřních modelů databáze, bez nutnosti psát SQL dotazy přímo, všechna data mají escaped characters, to znamená, že jsou upravena tak, aby se nedala splést s SQL dotazem, což zabraňuje možnosti SQL injection útoku na databázi. Více je tato problematika rozebrána v kapitole o bezpečnosti 4.7.2. Díky Sequelize, není potřeba psát složité join SQL dotazy přes více tabulek, Sequelize umožňuje tyto dotazy napsat velmi jednoduše. Největší výhodou je kompatibilita napříč těmito databázemi: Postgres, MySQL, MariaDB, SQLite a Microsoft SQL Server. Tyto databáze lze používat bez zásadní zásahy do kódu. (Samozřejmě je potřeba změna připojení.)

Moebius/http-graceful-shutdown

Moebius/http-graceful-shutdow [34] je použit v případě načtení nových testových dat, viz kapitola 4.6, kdy je potřeba celý celý systém restartovat. Při ukončení systému se však může stát, kvůli asynchronnímu provádění kódu, že existují ještě nějaké neuzavřené a neukončené requesty (dotazy) od uživatelů. Tato knihovna všechny tyto requesty obslouží, aby nečekaly na další spojení, i když je server již odpojen.

Forever

Forever [35] je použit pro automatické restartování serveru při načtení nových testových dat, když se celý systém restartuje.

4.3 Front end

Front end označuje veškerou vizuální část aplikace, tedy hlavně reprezentaci dat, rozhraní pro manipulaci s těmito daty a také komunikaci mezi serverem a klientem.

Pro popis HTML struktury dokumentů byl použit EJS Templating language. Template language slouží k dynamickému-generovaní HTML, to znamená, že pro každého uživatele můžeme vygenerovat jeho stránku na míru pouze z jednoho zdrojového kódu. EJS používá velmi jednoduchou syntax, a to kombinaci čistého HTML kódu a JavaScriptového (Js) kódu. Jeho syntax se skládá z bloků, které ohraničují Js kód od běžného HTML kódu. K tomu jsou k dispozici 3 značky. Js kód zanořený v těchto značkách se provádí na straně serveru, není tedy potřeba se obávat o bezpečnost v těchto blocích. Jejich použití se nejlépe vysvětlí na příkladech:

```
<% Js kod %>
```

Tyto dvě značky ohraničují čistý javascriptový kód, který neprodukuje žádný výstup do výsledného HTML souboru, slouží pouze k logické řízení programu a ohraničuje výstupy v HTML, např:

```
<% if(true) {%>
    <p>Tento odstavec bude ve výsledném HTML</p>
<% } else { %>
    <p>Tento odstavec Nebude ve výsledném HTML</p>
<% } %>
```

Dalším blokem je výstup z proměnné, který má escaped characters. To znamená, že se veškeré znaky, které by se jinak v HTML/CS/JS reprezentovaly jako kód, se nahradí svým HTML kódem, není tedy možnost u tohoto příkazu takzvaný Cross-site-scripting útok (více v kapitole o bezpečnosti 4.7). Tento blok se se značí:

```
<%= název promenné %>
```

Jelikož se jedná o bezpečný výstup, je jeho využití velmi časté v kódu, který je použit v implementovaném systému a může obsahovat vstup od uživatelů.

Posledním blokem v Ejs je výstup, který nemá escaped characters, který se používá převážně (a v případě této práce pouze) v kombinaci s příkazem include, který vloží jiný .ejs soubor na dané místo:

```
<%- název promenné %>
nebo
<%- include('název souboru') %>
```

Díky příkazu include se velmi zjednoduší struktura všech HTML dokumentů a případné úpravy, které je nutno provést na všech stránkách. Lze s pomocí

include upravit pouze jeden soubor a na všech stránkách, kde je příkaz include, se tyto změny projeví.

4.4 Databáze

K implementaci databáze byla použita NPM knihovna sequlize, která dovoluje implementovat SQL modely bez psaní SQL kódu a zaručuje kompatibilitu mezi jednotlivými SQL databázemi, viz kapitola 4.2.4.

Jako SQL databáze byl použit MariaDB server společně s Apache v balíku XAMPP [36].

Příklad implementace:

Nejprve je potřeba importovat objekt sequelize. Což je nastavené připojení na databázi v souboru system/controllers/database.js:

```
// automaticly creates connection pool
const sequelize = new Sequelize('testuserDB','root','',
  {dialect: 'mariadb', host: 'localhost', logging: false});
module.exports = sequelize;
```

Jako příklad poslouží následující model časového slotu (Timeslot) implementovaný v systému v souboru system/models/tymeslot.js:

```
const sequelize = require('../controllers/database');

sequelize.define('timeslot', {
    idTimeslot: {
        primaryKey: true,
        type: Sequelize.INTEGER,
        autoIncrement: true,
        allowNull: false
    },
    name: {
        type: Sequelize.STRING,
    },
    order: {
        type: Sequelize.INTEGER
    }
    //FK of Timetable
});
```

Jeho implementace a implementace všech modelů v databázi odpovídá jejich návrhu v kapitole 3.4.

Vazby na cizí klíče jsou následně nastaveny v souboru souboru system/models/relations.js. Jako příklad poslouží vztah mezi časovým slotem (timeslots) a rozvrhem (Timetable):

```
// Timeslot
// add FK of Timetable to Timeslot
Timeslot.belongsTo(Timetable, {foreignKey: 'idTimetable',
foreignKeyConstraint: true});
Timetable.hasMany(Timeslot, {foreignKey: 'idTimetable',
foreignKeyConstraint: true});
```

4.5 Implementace modelu celočíselného programování pro automatickou tvorbu rozvrhu

Implementace modelu celočíselného programování využívá model popsaný v kapitole 3.6. Pro komunikaci se solverem Gurobi je navržen skript v jazyce Python. Lze ho nalézt ve složce system/models/tymeslot.js:

```
from gurobipy import *

# set of values
timeslots = set()
courses = {}
userPref = {}
userAvail = {}
rooms = {}
combinationslist = []

# load timeslots from file
with open('timeslots.txt', "r") as openfileobject:
    for line in openfileobject:
        timeslots.add(int(line.rstrip()))
```

První řádek načte funkce z Gurobi. Další řádky vytvoří proměnné pro potřebné hodnoty. Následují 4 bloky kódu, které načtou do těchto proměnných data ze souboru, to jsou data, která uloží systém z databáze před tím, než se spustí tento skript. V ukázce je uveden pouze první blok, který načte časové sloty do vytvořené proměnné.

```
for c in courses:
    for t in timeslots:
    combinationslist.append(tuple((c,t)))

combinations = tuplelist(combinationslist)
```

Následně se vytvoří všechny možné kombinace kurzů a časů a uloží do listu, který se následně převede na tuplelist, což je speciální list od Gurobi, který umožňuje rychlejší manipulaci s daty.

Následuje vytvoření modelu:

```
model = Model('Timetable scheduling')

# x[c,s] = 1 if course c is assigned to timeslot s
x = model.addVars(combinations, lb=0, ub=1, name='x')

# objetive function, minimalization of users uprefered times
model.setObjective(quicksum((userAvail[courses[c]["user"]][s] -
 userPref[courses[c]["user"]][s]) * x[c,s] for c,s in combinations),
   GRB.MINIMIZE)
```

Příkaz model() vytvoří model, který následně upravují další příkazy. Model.addVars() přidává do modelu takzvané modelové proměnné, které jsou definované v definici modelu pro ILP 2.4. Proměnná lb značí lower bound a proměnná ub značí upper bound.

Následuje příkaz Model.setObjective(), který nastaví objektive function na minimalizaci a s pomocí dříve vytvořených modelových proměnných a načtených dat se definuje její podoba stejně jako je v matematické notaci.

Následují 4 podmínky:

```
# Condition 1
for c in courses:
    #every course is allocated only once,
    #if we replace right side with different number,
    #then we get how many times we need to allocate course
model.addConstr(quicksum(x[c,s] for s in timeslots)==1)

# Condition 2
for c, s in combinations:
    # every user is available in the assigned time
model.addConstr(x[c,s] <= userAvail[courses[c]["user"]][s])

# Condition 3
for s in timeslots:
# there is enough rooms of certain type in time that is
# assigned to some course
    model.addConstr(sum(x[c, s] for c in courses)
      <= rooms[s][courses[c]["roomtype"]])

# Condition 4
# if course is assigned to some user than that same user
# cannot be assigned to different course, that has the same time
for c, s in combinations:
    if x[c,s] == 1 :
        for c2, s2 in combinations:
            if ((courses[c]["user"] == courses[c2]["user"]) and (c != c2)) :
                model.addConstr(x[c2,s]==0)
```

Kde příkaz model.addConstr() vkládá podmínku na předem definované modelové proměnné. Stejně jako výše jsou všechny podmínky implementovány podle návrhu v kapitole 2.4.

Nyní je již model kompletní a lze ho optimalizovat:

```
model.optimize()
try:
model.printAttr('x') # slack variables
except:
print("model is not feasible, relaxing")
model.feasRelaxS(0, False, True, True)
model.optimize()

model.write("timetabling.sol")
```

Příkaz model.optimize() započne a vyřeší definovaný optimalizační problém. Jelikož je možné, že problém nebude feasible, tedy některá z podmínek bude muset být porušena pro vytvoření rozvrhu. To se dá zjistit tak, že příkaz model.printAttr() vyhodí chybu, protože v sobě nebude mít uložené žádné hodnoty. V tom případě lze problém relaxovat a i tak nalézt řešení pomocí příkazu model.feasRelaxS(). Současná implementace relaxace dovoluje přiřazení proměnných tak, že se přiřadí pouze ty časy, které neporuší žádnou z hard podmínek (bere se v potaz pouze cílová funkce, jelikož všechny podmínky 1 až 4 jsou hard podmínky). Poslední příkaz model.write("timetabling.sol") uloží výsledný model do souboru.

4.6 Testová data

V systému jsou pro prezentaci připraveny testová data. Jeden z testů se vždy načítá ze souboru při spouštění serveru test.txt. Při spuštění nového testu se kompletně vymaže celá databáze. Tyto testová data lze načíst do systému přes rozhraní v menu>tests.

4.7 Bezpečnost

Přestože bezpečnost webového systému není hlavní součást této práce, jedná se o neodlučnou problematiku při tvorbě jakéhokoliv systému, který je vystaven přístupu od uživatelů zvenčí. V následujících třech kapitolách jsou uvedeny problémy, které jsou řešeny v této práci.

4.7.1 Hashování hesel

Při jakémkoliv ukládání uživatelských dat a především hesel, je na správci systému udržovat tato hesla v takové podobě, aby případný útočník, který by mohl získat přístup k těmto heslům, měl co největší problém tyto data využít pro svůj prospěch. Vzniká také otázka, zda by tato hesla měla být vůbec přístupná samotnému správci. Samozřejmě pokud hesla ukládáme, musíme s nimi manipulovat a mít k nim přístup. Pro tyto účely je potřeba všechna hesla od uživatelů takzvaně hashovat, používá se k tomu hasovací funkce.

Hashovací funkce je taková funkce, která vezme data na vstupu, v tomto případě řetězec znaků a změní řetězec na jiný řetězec (*hash*) tak, aby nebylo možné zjistit, jaká data byla na vstupu do této funkce. Podstatnou informací je, že tento krok je jednostranný. (Z řetězce znaků lze získat hash, avšak z hashe nelze získat zpět řetězec znaků). Při registraci nového uživatele se jeho heslo vloží do hashovací funkce a výsledný hash se uloží do databáze. Při přihlašování se opět heslo, které uživatel vložil do přihlašovacího formuláře, vloží do hashovací funkce a výsledek se porovná s hashem uloženým v databázi. Díky tomu případný útočník a ani správce systému nevědí heslo uživatelů, avšak lze stále uživatele autorizovat při přihlášení.

Tento přístup fungoval relativně dobře, dokud útočníci nezačali generovat všechny možné kombinace hashů a ukládat si k ním řetězec, ze kterého tento hash vznikl. Potom útočníkovi, který se dostal k hashům stačilo porovnat databázi těchto vygenerovaných hashů a mohl zjistit hesla uživatelů. Proto se dnes při každé registraci nového uživatele vygeneruje nový náhodný řetězec označený jako *sůl* (*salt*), který se přidá k řetězci od uživatele a celý se takto vloží do hashovací funkce. Výsledný hash se uloží společně se *solí*, pro ověření při přihlášení. Díky tomu je dnes již o dost obtížnější předem vygenerovat všechny hashe (minimálně se současným hardwarem, záleží na algoritmu hashování). Avšak s rostoucím výkonem hardwaru je nutnost zvyšovat složitost algoritmů a velikost *soli*. Na tomto přístupu pracuje NPM knihovna Bcrypt použita v této práci 4.2.4. S rostoucím výkonem hardwaru dovoluje zvyšovat velikost *soli*, která přímo ovlivňuje délku trvání hashovací funkce. Lze tedy s časem zvýšit složitost výpočtu podle potřeby.

4.7.2 SQL injection

Při použití libovolné SQL databáze je systém vystaven možnosti útoku nazývaného SQL injection. Ten spočívá v tom, že útočník vloží do vstupu, který se používá ke komunikaci s SQL databází (takzvaný *SQL dotaz (SQL query)*) takový řetězec znaků, který příkaz ovlivní nebo provede příkaz útočníka. Potencionální útočník se takto může dostat k citlivým datům na stránce, nebo s databází manipulovat. (Například ji celou smazat, přihlásit se špatným heslem a podobně.) Zabránit útoku lze relativně snadno, stačí veškeré znaky, které by jinak SQL dotaz mohl přeložit jako speciální znaky ovlivňující dotaz,

nahradit jejich ekvivalentní formou (escaped characters), kterou SQL dotaz přeloží jako čistý text. O tento překlad se v této práci stará knihovna Sequelize 4.2.4.

4.7.3 Cross-site scripting (XSS)

Cross-site scripting je objemné téma, které představuje celou řadu nebezpečí, jedná se o kód útočníka, který je vložen do samotného kódu stránky a tím ovlivňuje jeho chování. Podobně jako u SQL injection, lze i tomuto útoku, alespoň částečně předejít nahrazením znaků, které by se jinak při zobrazení na stránce chovaly jako HTML/CSS/JS příkazy a ovlivňovaly její chování, ekvivalentní formou (escaped characters), které jsou tak přeloženy jako čistý text. O tento překlad se stará templatovací jazyk Ejs použitý v práci, jak je popsáno v kapitole 4.3.

4.8 Uživatelské testování

Na téma uživatelské testování lze napsat samostatnou bakalářskou či jinou práci. Proto bylo provedeno pouze zkrácené uživatelské testování, které si dává za cíl odhalit ty největší problémy s rozhraním. Metodika byla založena na základě materiálů dostupných v předmětu uživatelské testování (TUR) na ČVUT [39].

4.8.1 Cílová skupina

První cílovou skupinou jsou uživatelé používající rozvrhový systém, např. studenti ve věkovém rozmezí mezi 18 - 26 lety. Druhou možnou skupinou jsou správci těchto systémů ve věkovém rozmezí mezi 18 - 60 lety. Předpokládá se znalost anglického jazyka, případně testovaným bylo rozhraní přeloženo.

4.8.2 Případy použití

1. Registrace nového uživatele - je prvním nezbytným krokem pro každého nového uživatele v implementovaném systému. Je to nutná podmínka pro další možnou práci s tímto systémem.
2. Vytvoření nového rozvrhu a následně nového typu místnosti, místnosti a kurzu - je jedním z hlavních náplní práce pro administrátora, avšak mělo by být intuitivní i pro uživatele, kteří se setkali s libovolným rozvrhovacím systémem.
3. Přiřazení času kurzu ve vytvořeném rozvrhu - je poslední krok, představující manipulaci se samotným rozložením kurzů ve vytvořeném rozvrhu.

4.8.3 Testování kognitivním průchodem

Tato metoda uživatelského testování spočívá v simulaci průchodu programem při kterém je simulován uživatel z cílové skupiny. Pro tyto účely bylo testování provedeno s 3 uživateli, kde dva z nich patří do první cílové skupiny a třetí uživatelka patří do druhé cílové skupiny. Cílem metody je zjistit, zda je uživatel schopný zvládnout daný úkol, identifikovat situace a problémy, které při simulaci vznikají, kdy se uživatel odchyluje od předpokládaného průchodu programem. V každém kroku je testovaný/á dotázán/a na tyto otázky:

1. Otázka: Je Vám jasné, co přesně udělat pro dosažení kroku?
2. Otázka: Spojil/a jste si popis (menu/část webové stránky) s krokem, který se snažíte dosáhnout?
3. Otázka: Dostal/a jste dostatečnou zpětnou vazbu?

4.8.4 Test Case 1

Řeší se otázka, zda bude testovaný schopen registrovat se na stránce.

1. Otázka:

Odpovědi: Ano, Ano, Ne.

Popis: Uživatelům není přímo zřejmé, kde najít formulář k registraci.

Doporučení: Pod formulář přihlášení vložit odkaz na registraci.

2. Otázka: Ano, Ano, Ano.

Popis: Po nalezení registračního formuláře uživatelům nečinil problém si spojit registrační formulář s registrací.

3. Otázka: Ano, Ano, Ano.

Popis: Po registraci je uživatel automaticky přihlášen a v levé části obrazovky vidí své jméno.

4.8.5 Test Case 2

Řeší se otázka, zda bude testovaný schopen vytvořit nový rozvrh v systému, přidat typ místnosti, místnost a kurz.

1. Otázka:

Odpovědi: Ne, Ne, Ne.

Popis: Uživatelům není přímo zřejmé, že položka v menu s názvem ‘add‘ (přidat) slouží i pro tvorbu rozvrhu.

Doporučení: Jednou z možností je použít jiné označení např. ‘create‘ (vytvořit).

2. Otázka: Ano, Ano, Ano.

Popis: Po nalezení formulářů pro přidávání uživatelům nečinil problém si spojit toto rozhraní pro přidávání dalších částí.

3. Otázka: Ano, Ne, Ne.

Popis: Po vložení je testované/mu dána zpětná vazba o úspěšném vytvoření rozvrhu, avšak dva ze tří testovaných se setkali s chybovým hlášením, které pro pochopení jejich chyby, nebylo dostatečně podrobné. Druhým pozorováním bylo, že testovaným nebylo rozhraní pro zadávání jednotlivých dnů v rozvrhu intuitivní. Třetím pozorováním bylo, že uživatelé si nebyli jistí, co zadat do kolonky ‘name‘, jméno.

Doporučení: V prvním pozorování zlepšit zpětnou vazbu (učinit ji více konkrétní), nebo zlepšit ukázky. V druhém pozorování změnit rozhraní pro zadávání jednotlivých dnů na jednodušší rozhraní. Ve třetím poskytnout více informací, co kolonka ‘name‘ označuje.

4.8.6 Test Case 3

Řeší se otázka, zda bude testovaný schopen přiřadit vytvořenému kurzu nový čas.

1. Otázka:

Odpovědi: Ano, Ne, Ano.

Popis: Uživatelům nečinil problém najít rozhraní pro manipulaci s rozvrhem, je možné, že tuto informaci získali, když díky předchozím úkolům procházeli jednotlivé části stránek. Jednoho z testovaných překvapilo indexování časových slotů od nuly.

Doporučení: Indexování od nuly je zlozvyk kvůli indexaci v programovacích jazycích, kde se indexuje od nuly, avšak uživatelům přijde indexace od nuly jako neintuitivní. Doporučením je indexovat od jedné.

2. Otázka: Ano, Ano, Ano.

Popis: Po nalezení formuláře uživatelům nečinil problém si spojit Timetabling formulář s manipulací s rozvrhem.

3. Otázka: Ano, Ano, Ne.

Popis: Po změně času kurzu se změna automaticky projeví do rozvrhu, jeden z testovaných si toho však nevšiml a čekal důraznější zpětnou vazbu.

Doporučení: Zdůraznit právě aplikované změny.

4.8.7 Shrnutí testování

Testování systému odhalilo menší problémy v uživatelském rozhraní. Problémy se objevovaly především v nedostatečném popisu nebo zpětné vazbě uživateli.

Následující výčet tyto problémy popisuje a navrhuje doporučení pro jejich řešení:

- Případ použití: Registrace nového uživatele. Problém: Uživatelům není přímo zřejmé, kde najít formulář k registraci. Řešení: Pod formulář přihlášení vložit odkaz na registraci.
- Případ použití: Vytvoření nového rozvrhu a následně nového typu místnosti, místnosti a kurzu. Problém: Uživatelům není přímo zřejmé, že položka v menu s názvem ‘add‘ (přidat) slouží i pro tvorbu rozvrhu. Řešení: Použít jiné označení např. ‘create‘ (vytvořit).
- Případ použití: Vytvoření nového rozvrhu a následně nového typu místnosti, místnosti a kurzu. Problém: Dva ze tří testovaných se setkali s chybovým hlášením, které nebylo dostatečně podrobné. Řešení: Zlepšit zpětnou vazbu (učinit ji více konkrétní), nebo zlepšit ukázky.
- Případ použití: Vytvoření nového rozvrhu a následně nového typu místnosti, místnosti a kurzu. Problém: Testovaným nebylo rozhraní pro zadávání jednotlivých dnů v rozvrhu intuitivní. Řešení: Změnit rozhraní pro zadávání jednotlivých dnů na jednodušší.
- Případ použití: Vytvoření nového rozvrhu a následně nového typu místnosti, místnosti a kurzu. Problém: Uživatelé si nebyli jistí, co zadat do kolonky ‘name‘, jméno. Řešení: Poskytnout více informací, co kolonka ‘name‘ označuje.
- Případ použití: Přiřazení času kurzu ve vytvořeném rozvrhu. Problém: Jednoho z testovaných překvapilo indexování časových slotů od nuly. Řešení: Indexovat od jedné.
- Případ použití: Přiřazení času kurzu ve vytvořeném rozvrhu. Problém: Po změně času kurzu se změna automaticky projeví do rozvrhu, jeden z testovaných si toho však nevšiml a čekal důraznější zpětnou vazbu. Řešení: Zdůraznit právě aplikované změny.

4.9 Uživatelská dokumentace

Následující vyčerpávající výčet popisuje jednotlivé stránky, tak jak je uvidí uživatel, a jejich účel a funkcionalitu podle přihlášení:

Nepřihlášený uživatel:

- Login - slouží k přihlašování uživatele,

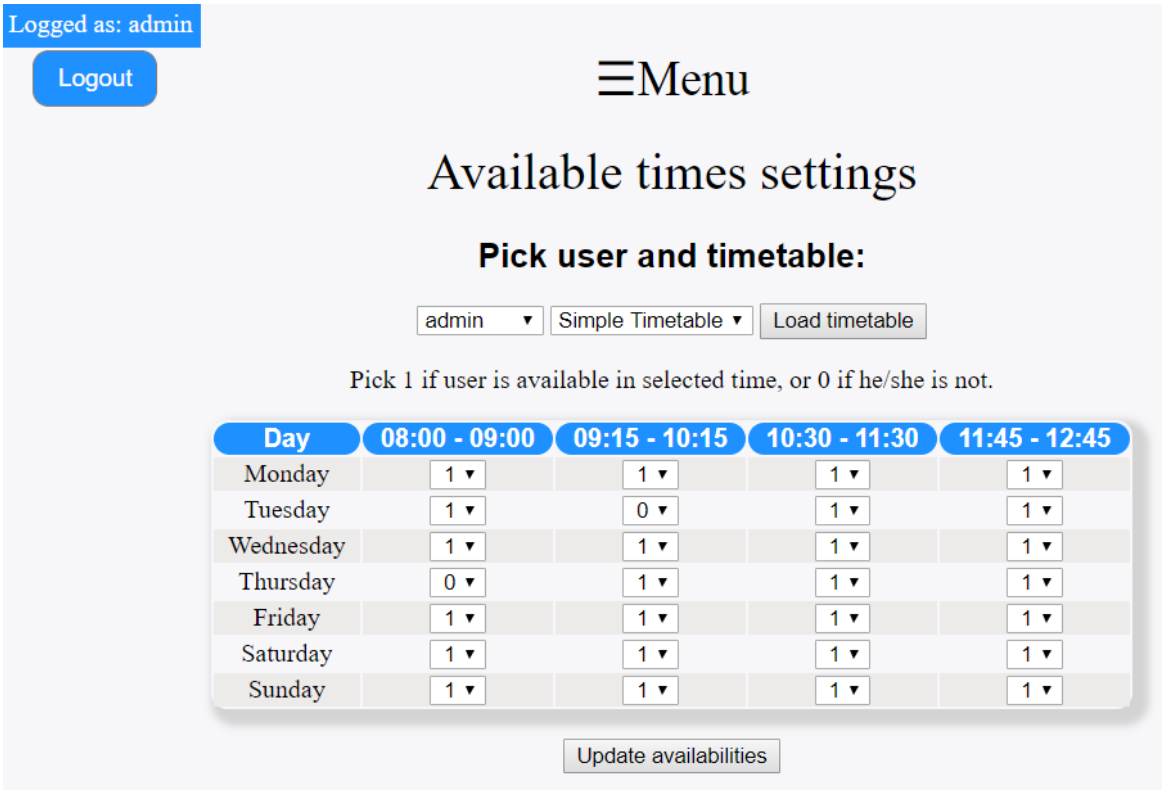
- Register - slouží k registraci uživatele,
- About - poskytuje krátký popisek softwaru.

Přihlášený uživatel:

- Home - slouží jako takzvaná landing page - rozcestník po přihlášení,
- Timetable - slouží k náhledu rozvrhu a umožňuje manipulaci s přiřazenými kurzy, zde se vytváří rozvrh.
- Edit
  - Edit Room Types - umožňuje nastavení již přidaných typů místností,
  - Edit Room (Type) Availability - umožňuje nastavení dostupností jednotlivých typů místností pro jednotlivé časy v rozvrhu,
  - Edit Courses - umožňuje nastavení kurzů, volbu jiného uživatele, jiného typu místnosti a posledně rozvrhu,
  - Edit User Availablity - umožňuje nastavení časů, kdy je uživatel dostupný,
  - Edit User Preferences - umožňuje nastavení časů, které uživatel preferuje,
  - Edit Rooms - umožňuje nastavení jiného typu místnosti.
- Add
  - Add Course - slouží k přidání nového kurzu,
  - Add Room Type - slouží k přidání nového typu místnosti,
  - Add Room - slouží k přidání nové místnosti,
  - Add Timetable - slouží k přidání nového rozvrhu.
- Tests - umožňuje načtení prezentovaných dat v kapitole 4.6,
- Solver - slouží k tvorbě automatického rozvrhu, načte data do solveru a vyřeší timetabling problem,
- Logout - slouží k odhlášení uživatele,
- About - zůstává zobrazitelná i pro přihlášeného uživatele, obsahuje krátký popisek o systému a autorovi.

Následující ukázky prezentují využití stránky, nejedná se o všechny rozhraní, ale pouze vybranou množinu:

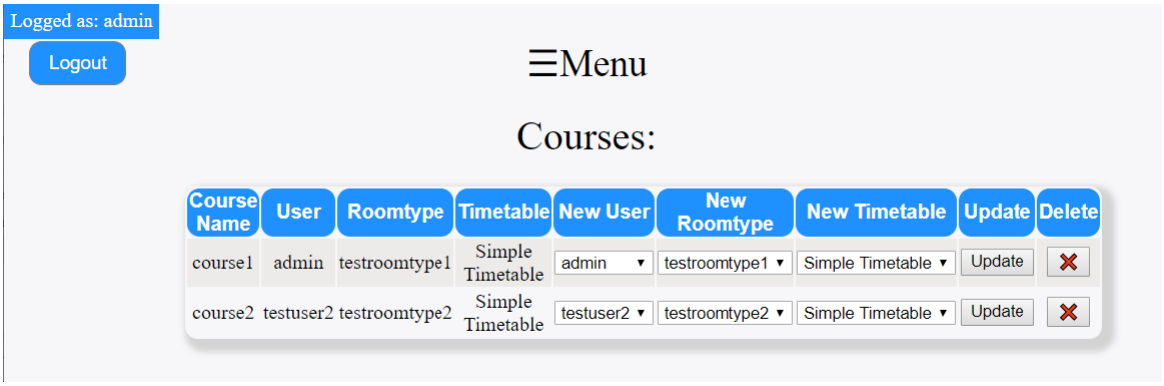
menu>edit>edit user availabilities



Obrázek 4.1: Rozhraní pro nastavování časů dostupnosti

Toto rozhraní slouží pro nastavení časů, kdy je uživatel dostupný. Prvním krokem je zvolit, jakého uživatele a jaký rozvrh chce editovat. Následně se klikne na tlačítko Load timetable a zobrazí se tabulka. Po zobrazení tabulky lze nastavit číslo preferovaného času na 0 nebo na 1 a kliknout na tlačítko Update availabilities, které upravené hodnoty uloží. Naprosto totožně vypadající rozhraní slouží i pro nastavení časů, které uživatel preferuje.

menu>edit>edit courses



Obrázek 4.2: Rozhraní pro editování kurzů

Toto rozhraní slouží k úpravě informací o kurzech. V tabulce jsou vidět všechna data a lze zvolit data nová. Po kliknutí na Update se daný řádek uloží s novými daty. Tlačítko delete celý kurz smaže.

menu>add>add Timetable

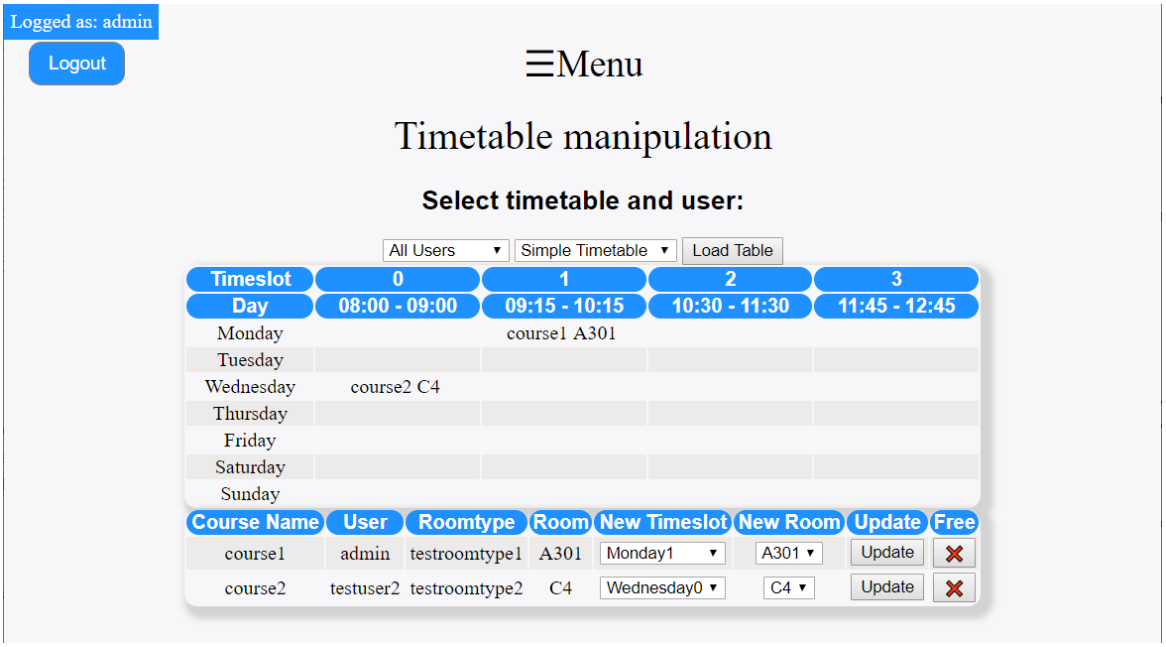
Obrázek 4.3: Rozhraní pro přidání nového rozvrhu

Toto rozhraní slouží k vytvoření nového rozvrhu. Každá kolonka má pod sebou informaci s příkladem, co vložit. Po vyplnění dat je potřeba kliknout na tlačítko add timetable, rozvrh se vytvoří a přidá do dat.

Jelikož se jedná o podstatnou část systému, ukažme, jak se rozvrh vytváří na příkladu: Chceme mít rozvrh každý den včetně víkendu, který začíná v osm hodin a každý den budeme mít kurzy trvající přesně jednu hodinu. Mezi každým kurzem chceme patnácti minutovou pauzu. Každý den chceme mít

čtyři kurzy. Do kolonek v tomto příkladu vložíme: Name - jméno rozvrhu ‘Simple timetable‘, Timestart - ‘08:00‘, Lenght of pause - ‘15‘, Length of one timeslot - ‘60‘, v rozhraní pro dny zvolíme postupně všechny dny podržením ctrl a kliknutím myši, number of timeslots - ‘4‘. Poté klikneme na add timetable, čžo přidá rozvrh.

Jak tento vytvořený rozvrh vypadá můžeme vidět na stránce menu>timetable:



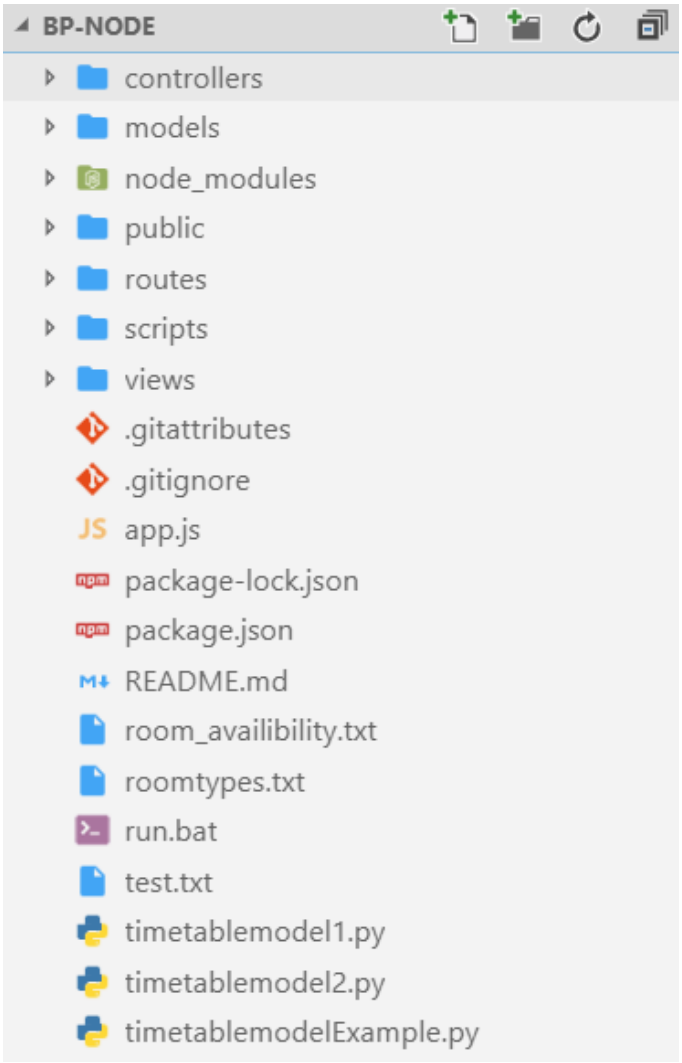
Obrázek 4.4: Rozhraní pro zobrazení rozvrhu a přiřazování kurzů

Nejprve je potřeba zvolit uživatele, jehož kurzy chceme v rozvrhu vidět a potom dotyčný rozvrh. Lze zvolit, že chceme vidět kurzy všech uživatelů. V této ukázce můžeme vidět i dva vytvořené kurzy, které jsou přiřazené jednotlivým časům. Podobně jako u nastavování editace kurzů, lze zde zvolit nové časy a místnosti a po stisknutí na update se tyto data uloží. Tlačítko Free uvolní kurz pryč z rozvrhu.

4.10 Programátorská dokumentace

4.10.1 Implementované soubory

Představme souborový systém, včetně všech jeho složek a popišme funkce jednotlivých souborů:



Obrázek 4.5: Souborový systém

V této části práce je důležité zmínit, jak fungují větší javascriptové (Node.js) projekty. Na rozdíl od projektů napsaných v jazyce Java, kde každý soubor je sám o sobě třídou, jsou projekty v Node.js napsány tak, že každý soubor je sám o sobě skript obsahující funkce, případně objekty, které se cíleně exportují. Podobně jako *public* funkce v jazyce Java.

V kořenovém adresáři lze nalézt 13 souborů, vyobrazených na obrázku 4.1, následující výčet vysvětlí jejich každého z nich:

.gitignore, .gitattributes, README.md

Jsou soubory sloužící pro manipulaci se systémem GitHub, který byl v případě této práce použit jako zálohovací program.

app.js

App.js je hlavní skript systému, který se volá při spouštění serveru konzolovým příkazem:

```
npm run start
```

Má na starost importování správných knihoven, nastavení uživatelských sessions, vytvoření a připojení k databázi, nastavení routeru pro správné routování adres (vysvětleno v kapitole 4.9) a následně zapne poslouchání pro requesty na daném portu.

package.json, package-lock.json

Package.json je soubor obsahující seznam použitých npm knihoven, včetně jejich verzí. Vytváří se a upravuje automaticky pomocí NPM příkazů.

Package-lock.json je soubor obsahující přesné verze použitých npm knihoven, včetně adres, odkud je lze stáhnout. Vytváří se a upravuje také automaticky.

run.bat

Run.bat je krátký skript spouštějící pythonový soubor *timetablemodel1.py*, jeho kód a využití je popsáno v kapitole 4.5.

*.py

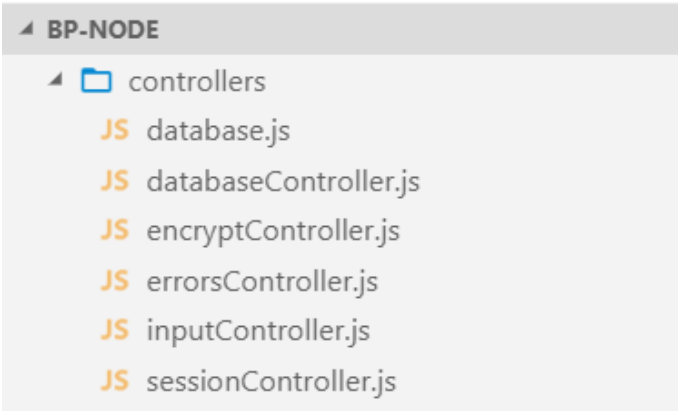
Soubory končící příponou *.py* slouží k využití solveru, definují jeho model popsaný v kapitole 4.5, nebo případně slouží jako ukázka využití solveru.

*.txt

Ostatní soubory končící příponou *.txt* slouží k ukládání dat z databáze a nahrávání do pythonovského scriptu (modelu solveru), mimo souboru test.txt, ten slouží k nahrání správného testu do databáze (více v kapitole 4.6).

Nyní si rozebereme soubory v jednotlivých složkách:

První složkou v kořenovém adresáři implementovaného systému je složka controllers:

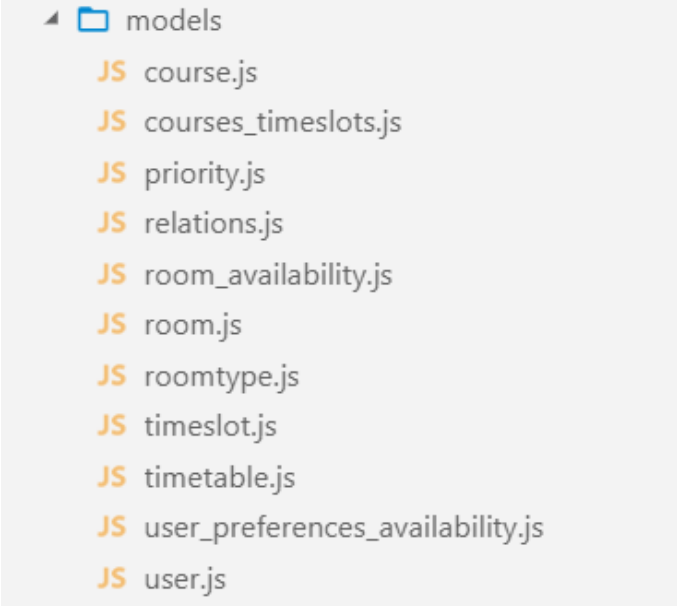


Obrázek 4.6: Složka controllers

Tato složka v sobě obsahuje .js skripty sloužící k ověřování správnosti dat, načítání a ukládání dat z databáze a další:

- database.js - Slouží k definici databáze a k vytvoření připojení. Kód je popsán v kapitole 4.4.
- databaseController.js - Obsahuje všechny funkce, které nějakým způsobem manipulují s databází, odpovídají za ukládání dat, úprava dat, odstraňování dat nebo jejich načítání.
- encryptController.js - Obsahuje funkce pro vytvoření hashe při registraci a porovnání hashe při přihlášení uživatele.
- errorsController.js - Obsahuje funkce pro kontrolu chyb, v současné implementaci je to jen jedna funkce, sloužící v případě, kdy se uživatel snaží dostat na neexistující stránku, k přesměrování na stranu s kódem 404, tedy stránka neexistuje.
- inputController.js - Obsahuje funkce pro kontrolu uživatelského vstupu, zda zadaná data od uživatele jsou validní, aby je šlo uložit do databáze.
- sessionController.js - Obsahuje funkce pro kontrolu, zda je uživatel již přihlášen, nebo se musí teprve přihlásit.

Druhou složkou v kořenovém adresáři implementovaného systému je složka models:



Obrázek 4.7: Složka models

Složka models obsahuje jednotlivé scripty pro definici modelů podle databázového návrhu. Podstatným souborem je soubor relations.js, který kromě definice vazeb mezi jednotlivými modely také obsahuje definované triggery (v sequelize nazývané hooks), které při vytvoření provedou další 'skrytý' příkaz, který provede přidání nebo úpravu dat v databázi.

Následující kód v souboru relations ukazuje jeden z triggerů, který zvětší počet dostupných místností o plus jedna pro všechny časy při přidání nové místnosti:

```
// Room hook
// hook for updating number of rooms when new room is added
Room.afterCreate(room => {
    const getAllRoomAvails = RoomAvailability.findAll(
       {where:{idRoomtype: room.idRoomtype}});
    Promise.all([getAllRoomAvails]).then(responces => {
        const roomavails = responces[0];
        roomavails.forEach(roomavail => {
            roomavail.numOfAvailRooms = roomavail.numOfAvailRooms + 1;
            roomavail.save();
        });
    })
});
```

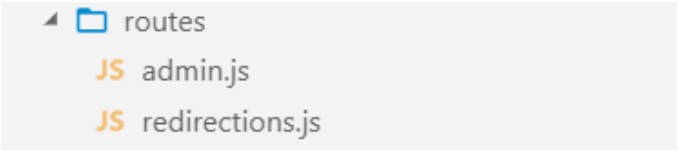
Třetí složkou v kořenovém adresáři implementovaného systému je složka public:

Obrázek 4.8: Složka public

Složka public obsahuje všechny soubory, které jsou veřejně dostupné, jedná se většinou a statické soubory css/js definující funkcionalitu stánek. V případě této práce byly tyto soubory napsány na základě různých tutoriálů na stranách MDN [37] a W3schools [38], v případě použití větší část kódu bez úpravy je uvedený konkrétní odkaz přímo v kódu na místě, kde se vyskytuje.

- main.css - Slouží k definici hlavního vzhledu všech stránek.
- sidenav.css - Slouží k definici vzhledu menu, jehož struktura je popsána u kapitoly 4.9.
- table.css - Slouží k definici vzhledu tabulek, hlavně tabulky pro reprezentace rozvrhu.
- requstScripts.js - Je javascriptový soubor poskytující funkcionalitu pro použití AJAX komunikace se serverem a načítání dat do stránky dynamicky s pomocí Javascriptu.

Čtvrtou složkou v kořenovém adresáři implementovaného je složka routes:



Obrázek 4.9: Složka routes

Přestože tato složka obsahuje pouze dva soubory, jedná se o nejdelší část kódu v celém systému. Tyto soubory řeší problém routování, způsob, která stránka a s jakými daty se uživateli zobrazí při přijatém requestu. Použití souboru redirection.js bylo již naznačeno v kapitole 4.2.2, slouží ke přesměrování nepřihlášeného uživatele nebo naopak přihlášeného uživatele přesměruje z formuláře přihlášení na úvodní stránku systému. Následuje jednoduchá ukázka z druhého souboru (admin.js):

```
router.get('/home', redirections.redirectLogin, (req, res) => {
    res.render('home', {user: sessionController.getLoggedUser(req)});
```

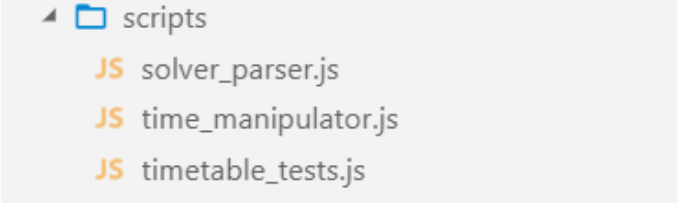
```
});
```

Router je objekt, který zjednodušuje přijímání a odesílání dat, např. není potřeba parsovat data, o to se postará router. V případě této funkce, když přijde GET příkaz od libovolného uživatele s adresou '/home', se nejprve provede middleware redirectLogin, který uživatele v případě, když není přihlášen přesměruje na stránku s přihlášením. Pokud se však zavolá v middleware funkci redirectLogin příkaz next(), tak se spustí tato funkce, kde se s pomocí příkazu

```
res.render('home', {user: sessionController.getLoggedUser(req)});
```

dynamicky vytvoří HTML ze souboru .ejs s daty, která jsou uvedena ve funkci, v tomto případě s daty 'user'. Následně je vygenerovaný HTML odeslán uživateli.

Pátou složkou v kořenovém adresáři implementovaného je složka scripts:

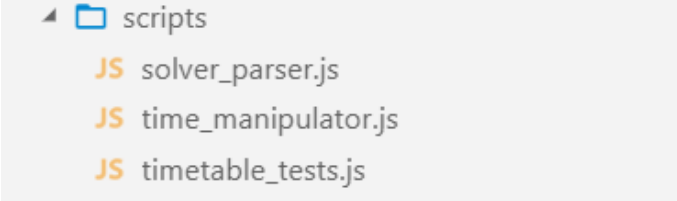


Obrázek 4.10: Složka scripts

Tato složka v sobě obsahuje skripty dostupné pouze na serverové straně, řešící zmíněnou problematiku:

- solver_parser.js - obsahuje funkce sloužící k ukládání dat z databáze do souboru .txt pro použití v python skriptu pro solver Gurobi, řešící automatickou tvorbu rozvrhu a také funkce, které nahrají do databáze výsledek solveru.
- time_manipulator.js - obsahuje funkce sloužící ke správné reprezentaci času v tabulce, která je generována pro zobrazení rozvrhu, např. správné délky jednotlivých bloků a jejich umístění v tabulce rozvrhu.
- timetable_tests.js - obsahuje funkce sloužící k načítání testových dat, více popsáno v kapitole 4.6.

Šestou složkou v kořenovém adresáři implementovaného je složka views:



Obrázek 4.11: Složka views

Tato složka v sobě má 4 další podsložky, dvě z nich ('adds' a 'edits') reprezentují jednotlivé stránky, které jsou při správném příkazu použity pro vygenerování HTML dokumentu, který je následně odeslán uživateli.

Další dvě složky 'includes' a 'loads' slouží k jinému účelu:

Složka includes v sobě obsahuje jednotlivé části stránky, které se opakují na všech stránkách, a proto jsou v odděleném souboru, aby se lépe spravovaly a kód byl jenom na jednom místě.

Složka loads v sobě obsahuje pouze definice tabulek (tables), slouží při AJAX komunikaci, kdy není potřeba posílat data celé stránky, ale pouze nově vygenerovanou (upravenou) tabulku.

4.11 Nasazení systému

Pro nasazení systému je zapotřebí nainstalovat několik programů/knihoven. Systém byl testován a vyvíjen na operačním systému Windows 10.

- XAMPP - obsahující MariaDB a jednoduché Apache rozhraní pro případnou manipulaci s daty, pro testování a implementaci nové funkcionality.
- Node.sj - Runtime pro spuštění serveru, obsahuje Node package manager (NPM).
- Python - je nutno nainstalovat verzi, kterou podporuje solver Gurobi (v této práci verze 3.7 64 bit).
- Gurobi solver - k jeho instalaci je zapotřebí licence, tato práce využívá studentské licence dostupné pro studenty ČVUT.

Po instalaci podporované SQL databáze je potřeba nahradit přihlašovací údaje v souboru '/system/controllers/database.js', včetně proměnné dialect, podle použité databáze. Pro více informací je vhodné zkontrolovat dokumentaci knihovny Sequelize.

Po instalaci podporované verze Python je potřeba nastavit cestu k nainstalované verzi v souboru '/system/run.bat', kde první odkaz musí být na Python runtime a druhý na 'timetablemodel1.py' soubor podle aktuálního adresáře.

Po instalaci potřebného softwaru je potřeba se pomocí příkazů dostat v CMD (command-line) do souborového systému a zadat příkaz:

```
npm install
```

Tento příkaz nainstaluje chybějící node moduly podle souboru 'package.json'.

Dalším krokem je nainstalovat knihovnu forever globálně, jelikož se stará o restartování serveru na pozadí.

```
npm install forever -g
```

Následujícím příkazem lze zapnout server:

```
npm run start
```

Server je nutno ukončit příkazem:

```
forever stop app.js
```

V některých případech je potřeba otevřít CMD (command-line) s volbou ‘jako správce‘, jelikož se spouští Windows shell script (.bat), což potřebuje dodatečná práva.

4.12 Licenční prohlášení

Rád bych upozornil, že pro takové využití solveru Gurobi, které prezentuji v této práci, je potřeba zakoupení serverové licence. Více lze nalézt na oficiálních stránkách firmy Gurobi [20]. Veškeré testování prezentované v této práci bylo použito v souladu s licenčními podmínkami, a to čistě pro akademické účely.

Kapitola 5

Závěrečné zhodnocení výsledků

5.1 Zhodnocení práce

Tato práce poskytla úvod do problematiky tvorby rozvrhu a systému pro jeho správu. S pomocí rešerše byl vytvořen návrh systému. Významnou částí této práce je implementace systému využívajícího teoretický model pro tvorbu rozvrhu automaticky. Bylo také provedeno uživatelské testování a odhaleny chyby v uživatelském rozhraní. Cíle, které si práce zadala jsou splněny. Avšak pro nasazení systému do ostrého provozu, by bylo potřeba systém více rozšířit, především o funkcionalitu umožňující nastavování rolí a přiřazování činností k těmto rolím.

5.2 Možnosti budoucí práce

Práce byla implementována podle návrhu v základní pilotní verzi. Návrh systému lze snadno rozšířit o mnoho nových poznatků, např. přidat funkcionalitu pro přiřazování jednotlivých účastníků kurzů, přidat možnosti nastavování pravomocí na stránce. Návrh využití solveru lze také snadno rozšířit o další funkcionalitu, např. přidat více možných podmínek na místnosti, změnit cílovou funkci na upřednostnění jiného kritéria než pouze nepreferovaných časů, případně model upravit, aby šlo nastavit více úrovní preferovanosti.

Literatura

[1] Pinedo L. Michael. *Scheduling Theory, Algorithms, and Systems*. Springer, 2008.

[2] Richard W. Conway, William L. Maxwell, Louis W. Miller. *Theory of Scheduling*. Dover Publications, June 9, 2003.

[3] Muth, John F., Thompson, Gerald Luther. *Industrial scheduling*. Englewood Cliffs, N.J. : Prentice-Hall, 1963.

[4] Hana Rudová. Rozvrhování. `https://www.fi.muni.cz/~hanka/rozvrhovani/prusvitky/all-start.pdf`, 21. února 2019. Přístup: 18. května, 2019.

[5] Javier Larrosa, Albert Oliveras, Enric Rodríguez-Carbonell. Mixed Integer Linear Programming. `https://www.cs.upc.edu/~erodri/webpage/cps/theory/lp/milp/slides.pdf`, April 26, 2019. Přístup: 18. května, 2019.

[6] OptaPlanner. `https://www.optaplanner.org/`, 2006-2019. Přístup: 18. května, 2019.

[7] Cbc (Coin-or branch and cut). `https://projects.coin-or.org/Cbc`, February 27, 2019. Přístup: 18. května, 2019.

[8] Iulian Ober. A variant of the high-school timetabling problem and a software solution for it based on integer linear programming. `https://www.patatconference.org/patat2016/files/proceedings/paper_23.pdf`, August 23–26, 2016. Přístup: 18. května, 2019.

[9] Gurobi solver. `http://www.gurobi.com/index` Přístup: 18. května, 2019.

[10] H. D. Mittelmann. Latest Benchmarks of Optimization Software. `http://plato.asu.edu/talks/informs2017.pdf`, 23 October, 2017. Přístup: 18. května, 2019.

[11] Unitime. https://www.unitime.org/, April, 2019. Přístup: 18. května, 2019.

[12] Unitime (github). https://github.com/UniTime/unitime Přístup: 18. května, 2019.

[13] Matej Lukáš. Course timetabling at Masaryk University in the UniTime system. https://is.muni.cz/th/y0gxo/Master_Thesis.pdf Přístup: 18. května, 2019.

[14] Purdue University. https://www.purdue.edu/ Přístup: 19. května, 2019.

[15] Dave Bangert. Journal and Courier. Purdue's record freshmen class way bigger than campus expected. https://eu.jconline.com/story/news/2018/05/04/purdues-record-freshmen-class-way-bigger-than-campus-expected/581051002/, April 11, 2018. Přístup: 19. května, 2019.

[16] Java development kit. https://www.oracle.com/technetwork/java/javase/downloads/index.html Přístup: 19. května, 2019.

[17] Apache Tomcat. http://tomcat.apache.org/ Přístup: 19. května, 2019.

[18] MySQL databáze. https://dev.mysql.com/ Přístup: 19. května, 2019.

[19] Kristiansen, Simon; Sørensen, Matias; Stidsen, Thomas Riis. Integer programming for the generalized high school timetabling problem. http://orbit.dtu.dk/files/128525231/Integer_Programming_MIP_XHSTT_journal_of_scheduling.pdf Přístup: 19. května, 2019.

[20] Gurobi licencing. http://www.gurobi.com/products/licensing-pricing/licensing-overview Přístup: 19. května, 2019.

[21] Distributed Application Architecture. https://web.archive.org/web/20110406121920/http://java.sun.com/developer/Books/jdbc/ch07.pdf Přístup: 19. května, 2019.

[22] Mihir Shah. MongoDB vs MySQL: A Comparative Study on Databases. https://www.simform.com/mongodb-vs-mysql-databases/ Přístup: 19. května, 2019.

[23] SqlDBM SQL Database Modeler. https://sqldbm.com/Home/ Přístup: 19. května, 2019.

[24] JavaScript runtime. https://nodejs.org/en/ Přístup: 19. května, 2019.

[25] Express web framework for Node.js. https://expressjs.com/ Přístup: 19. května, 2019.

[26] Node package manager (NPM). https://www.npmjs.com/ Přístup: 19. května, 2019.

[27] Bcrypt - nodejs package. https://www.npmjs.com/package/bcrypt Přístup: 19. května, 2019.

[28] Ejs - nodejs package. https://ejs.co/ Přístup: 19. května, 2019.

[29] Ejs-lint - nodejs package. https://www.npmjs.com/package/ejs-lint Přístup: 19. května, 2019.

[30] Express-session - nodejs (Express) package. https://www.npmjs.com/package/express-session Přístup: 19. května, 2019.

[31] Moment - nodejs package. https://momentjs.com/docs/ Přístup: 19. května, 2019.

[32] Nodemon - nodejs package. https://nodemon.io/ Přístup: 19. května, 2019.

[33] Sequelize - nodejs package. http://docs.sequelizejs.com/ Přístup: 19. května, 2019.

[34] Moebius/http-graceful-shutdown - nodejs package. https://github.com/moebius-mlm/http-graceful-shutdown Přístup: 19. května, 2019.

[35] Forever - nodejs package. https://www.npmjs.com/package/forever Přístup: 19. května, 2019.

[36] XAMPP, Apache + MariaDB + PHP + Perl. https://www.apachefriends.org/index.html Přístup: 19. května, 2019.

[37] MDN web docs. https://github.com/moebius-mlm/http-graceful-shutdown Přístup: 19. května, 2019.

[38] W3schools, web developer site. https://www.w3schools.com/ Přístup: 19. května, 2019.

[39] Adam J. Sporka a další. [Webové stránky předmětu] Testování uživatelského rozhraní. https://cent.felk.cvut.cz/courses/Y39TUR/?page=home Přístup: 19. května, 2019.

Příloha A

Obsah CD

- **madleja2_BP.pdf.** Text bakalářské práce v .pdf formátu.
- **\system** Kořenová složka implementovaného systému.