VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

WEBOVÁ APLIKACE PRO KOLABORATIVNÍ TVORBU ROZVRHŮ

WEB APPLICATION FOR COLLABORATIVE SCHEDULE MANAGEMENT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **DAVID NOVÁK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. DANIEL DOLEJŠKA**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

144784

Ústav:	Ústav informačních systémů (UIFS)
Student:	**Novák David**
Program:	Informační technologie
Specializace:	Informační technologie
Název:	**Webová aplikace pro kolaborativní tvorbu rozvrhů**
Kategorie:	Informační systémy
Akademický rok:	2022/23

Zadání:

1. Nastudujte technologie pro vývoj webových aplikací a další relevantní technologie dle požadavků vedoucího.
2. Prozkoumejte existující řešení interaktivních kolaborativních editorů, pokud možno se zaměřením na rozvrhy, a s nimi související technologie.
3. Vytvořte návrh aplikace umožňující interaktivní kolaborativní vytváření a editaci rozvrhů. Respektujte požadavky vedoucího a firemního konzultanta.
4. Implementuje aplikační řešení dle návrhu z bodu 3, požadavků vedoucího a firemního konzultanta.
5. Vytvořenou aplikaci z bodu 4 řádně otestujte dle požadavků vedoucího a firemního konzultanta.
6. Zhodnoťte přínosy a nedostatky vytvořeného řešení. Diskutujte další možný vývoj a rozšíření implementované aplikace.

Literatura:

- KRUG, Steve. *Don't make me think, revisited: a common sense approach to Web usability*. Third edition. vyd. Berkeley, Calif.: New Riders, 2014. ISBN 978-0-321-96551-6.
- STONE, Debbie et al. *User Interface Design and Evaluation*. Burlington: Elsevier Science, 2005. ISBN 978-0-08-052032-2.
- dle doporučení vedoucího

Při obhajobě semestrální části projektu je požadováno:
Body 1-3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Dolejška Daniel, Ing.**
Konzultant:	Milan Šorm, RNDr. Ing., Ph.D.
Vedoucí ústavu:	Kolář Dušan, doc. Dr. Ing.
Datum zadání:	1.11.2022
Termín pro odevzdání:	10.5.2023
Datum schválení:	26.10.2022

Abstrakt

Tato práce řeší tvorbu webové aplikace pro tvorbu kolaborativních rozvrhů pro univerzity a jiné instituce. Aplikace má za cíl implementovat editor pro tvorbu rozvrhů, který díky využití protokolu WebSocket umožňuje spolupráci uživatelů v reálném čase a automaticky vyhodnocuje kolize mezi lekcemi. Před vytvořením samotné aplikace byla nastudována existující řešení z oblasti kolaborativních a rozvrhových editorů. Nejpodrobněji bylo studováno řešení firmy IS4U, v jejíž spolupráci je tato práce realizována. Pomocí ER diagramu a wireframů byl vytvořen návrh pro její vylepšení. V implementaci převažuje jazyk TypeScript. Na serverové straně je využito běhového prostředí Node.js a frameworku Express.js. Klient je implementován pomocí frameworku Vue.js. V práci je také provedeno uživatelské testování aplikace a zamyšlení se nad budoucím vylepšením aplikace.

Abstract

This thesis addresses the development of a web application for collaborative schedule management for universities and other institutions. The application aims to implement a schedule editor that allows users to collaborate in real-time using the WebSocket protocol and automatically evaluates conflicts between lessons. Before creating the application itself, existing solutions in the field of collaborative and scheduling editors were studied. The solution of the company IS4U, with whom this thesis is being realized, was studied in the most detail. A design for its improvement was created using an ER diagram and wireframes. The implementation predominantly utilizes TypeScript. On the server side, Node.js runtime environment and the Express.js framework are utilized. The client is implemented using the Vue.js framework. The thesis also includes user testing of the application and considerations for future improvements.

Klíčová slova

kolaborativní aplikace, tvorba rozvrhů, webová aplikace, WebSocket, Vue.js, WebSocket, komunikace v reálném čase, TypeScript, Node, MySQL, Express, hledání kolizí, manuální testování

Keywords

collaborative application, scheduler, web application, WebSocket, Vue.js, real-time communication, TypeScript, Node, MySQL, Express, collision detection, manual testing

Citace

NOVÁK, David. *Webová aplikace pro kolaborativní tvorbu rozvrhů*. Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Daniel Dolejška

Webová aplikace pro kolaborativní tvorbu rozvrhů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Daniela Dolejšky. Další informace mi konzultant poskytl RNDr. Ing. Milan Šorm, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
David Novák
8. května 2023

Poděkování

Chtěl bych poděkovat vedoucímu práce Ing. Danielu Dolejškovi za vedení práce, rady a konzultace a konzultantovi RNDr. Ing. Milanu Šormovi, Ph.D. za konzultace při psaní této práce.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Plánování rozvrhů je velkým tématem nejen v prostředí základních a středních škol, ale také v prostředí univerzit. Před zahájením nového školního roku nebo univerzitního semestru je věnováno velké množství času tomu, aby jednotlivé rozvrhy odpovídaly nejen časovým možnostem a preferencím jednotlivých vyučujících, ale zároveň také časovým možnostem studentů a kapacitám jednotlivých místností. Tento proces je velmi časově a organizačně náročný a často při něm dochází ke vzniku nejrůznějších kolizí. Ideální rozvrh neobsahuje žádné kolize a naopak se snaží v co nejvyšší míře vyhovět preferencím jednotlivých vyučujících a studentů.

Jednotlivé školy či univerzity mají svá specifika, která se odráží určitým způsobem v nástrojích používaných pro tvorby jejich rozvrhů. Přestože v dnešní době existuje velké množství počítačových programů, jejichž cílem je proces tvorby rozvrhů co nejvíce zefektivnit, stále neexistuje komplexní řešení, které by bylo vhodné pro všechny typy škol. Příklady současných řešení jsou popsány v kapitole 2.3.

Tato práce byla tvořena ve spolupráci s firmou IS4U, která vyvíjí komplexní řešení, jehož cílem je v co nejvyšší míře zjednodušit práci během vytváření komplikovaných univerzitních rozvrhů. Její současné řešení je rozebráno v podkapitole 2.4. Toto řešení má v současnosti několik nedostatků a oblastí, které by chtěla firma vylepšit. Největší změnou by měl být přechod z desktopové na webovou aplikaci, aby byla přístupná bez nutnosti instalace na všech operačních systémech. Druhou velkou změnou by mělo být vylepšení editoru umožněním spolupráce více uživatelů v reálném čase. Vzhledem k rozsahu těchto změn plánuje firma vývoj nové aplikace.

Vytvořit kompletní aplikaci je práce pro několikačlenný tým vývojářů na několik měsíců. Tato práce má za úkol sloužit jako důkaz konceptu, pomocí kterého mohou předvést zákazníkům některé změny před tím, než začnou se samotným vývojem nové aplikace.

Požadavky firmy na to, co by měl koncept obsahovat jsou shrnuty v podkapitole 3.1. V podkapitolách 3.2 a 3.3 jsou tyto požadavky zpracovány do návrhu databáze a uživatelského rozhraní. Kapitola 4 popisuje v prvních dvou podkapitolách vybranou architekturu a technologie pro implementaci aplikace. Na tuto část navazuje kapitola 4.2, která popisuje implementaci serverové části aplikace, včetně popisu fungování přesunu lekcí nebo algoritmu pro detekci kolizí. Popis způsobu, jakým jsou zobrazovány rozvrhy, jak funguje editor pro rozvrhování lekcí a jak je zařízeno, že nenastanou nekonzistence při používání editoru více uživateli současně, je popsáno v kapitole aplikace 4.3. Předposlední kapitolou je kapitola 5 testování. Tato kapitola popisuje průběh a vyhodnocení uživatelského testování za pomoci testovacího protokolu přiloženého v příloze A. Nakonec jsou zde navrženy způsoby, kterými by šel koncept ještě rozšířit a zdokonalit.

Kapitola 2

Editory a tvorba rozvrhů

První část této kapitoly je věnována rozboru kolaborativních editorů a metodám, které tyto editory používají k synchronizaci změn. Druhá část je zaměřena na tvorbu rozvrhů a editory k ní používané.

2.1 Kolaborativní editory

Před dvaceti lety bylo s tehdejšími technologiemi velmi náročné vytvořit software, který by umožňoval spolupráci v reálném čase. V současnosti je již zcela běžné spolupracovat online prostřednictvím mnoha aplikací. [6] Jednou z nejznámějších je v dnešní době **Google Workspace**[1], díky kterému existují spousty aplikací, které umožňují spolupráci v reálném čase, ať už tvoříme prezentaci, diagram nebo cokoliv dalšího. Podobné řešení přináší i Microsoft, který jej nazývá **Co-authoring**. [22]

Vytvořit editor, ve kterém spolupracuje několik uživatelů upravujících stejný obsah v jednom okamžiku není technicky jednoduché. Vzhledem k tomu, že přesun informací o změnách není bezprostřední, je nutné vždy, když provedeme změnu, dočasně vytvořit lokální verzi obsahu. Hlavním problémem tedy je, aby se tyto lokální verze navzájem nepřepisovaly a aby konvergovaly vždy ke stejné, správné verzi obsahu. [6]

2.2 Přístupy k synchronizaci změn

První z možných přístupů je práce uživatelů na lokálních kopiích, kde po určité době dojde k **manuální synchronizaci**. Příkladem tohoto přístupu je většině programátorům známý verzovací nástroj git[2]. Tento systém umí zapracovat nekolizní změny dohromady, ale v případě konfliktu je nutná manuální asistence uživatele. [9]

Dalším přístupem je **porovnávání verzí**. Server při příchozí změně porovná obsah s tím, který má ve své lokální kopii a odešle aktualizovanou verzi uživatelům. Ke zlepšení uživatelského zážitku je však lepší přidat mechanismus, který zabrání situaci, kdy během upravování části obsahu uživateli dorazí aktualizace od jiného uživatele, jíž jsou právě provedené změny smazány. [6] Řešením může být použití vzájemného vyloučení. V průběhu práce s obsahem nebo jeho částí je ostatním uživatelům znemožněno jej upravovat do té doby, než je práce s ním dokončena. Toto nazýváme **pesimistické zamykání**. Další možností

[1] https://workspace.google.com/
[2] https://git-scm.com/

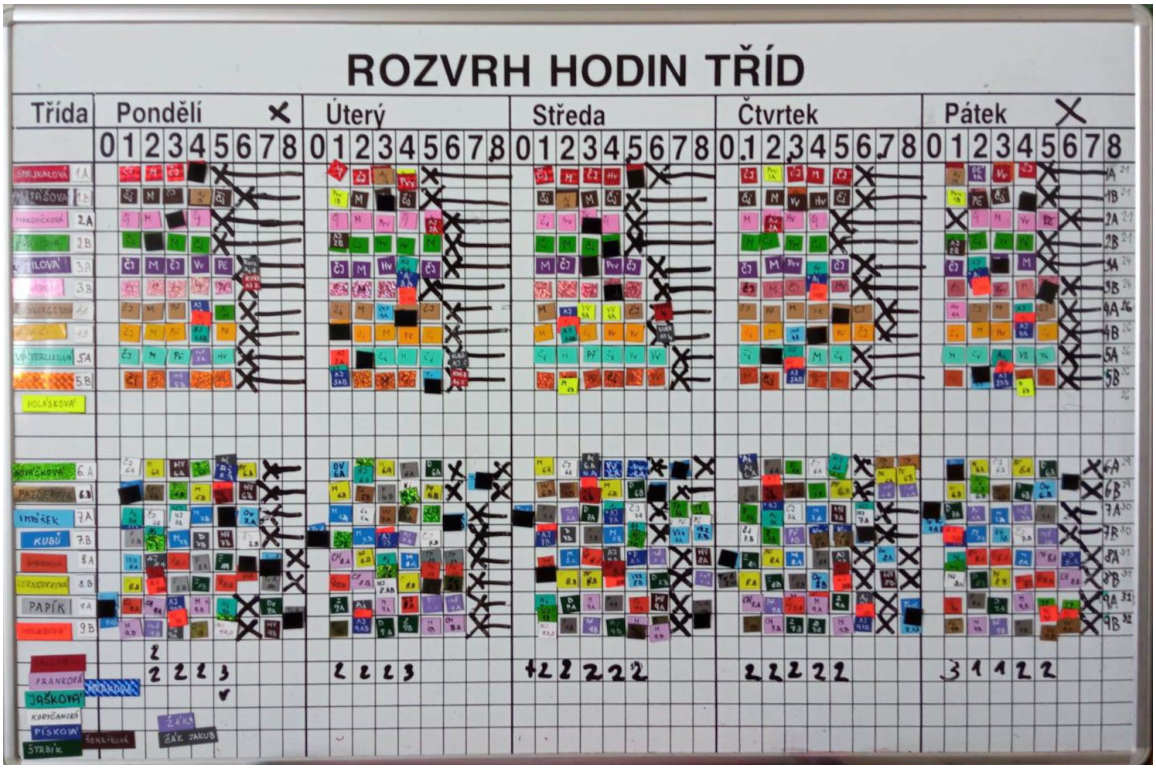
je **Optimistické zamykání**, které umožňuje pokusy o úpravu všem uživatelům současně, zamyká obsah pouze v okamžiku ukládání změny. [12]

Kolaborativní editory využívající platformu Google Workspace používají algoritmus **operační transformace** (OT)[6], ve kterém je každá elementární změna v obsahu popsána pomocí operace. Pokud jiný uživatel provede změnu, která se týká stejného místa v obsahu, OT provede transformaci tak, aby obě změny byly aplikovány v pořadí, ve kterém byly provedeny, a aby výsledkem byla korektní verze obsahu. [9]

V této práci je pro přesuny lekcí použito Pesimistické zamykání, protože přesun je ve své podstatě elementární operace a není žádoucí její přerušení případnou úpravou. Při nahrazování lekce nebo při úpravě dat pomocí formulářů je použit mechanismus podobný Optimistickému zamykání. Algoritmus OT není použit kvůli jeho komplikovanosti pro relativně jednoduchou problematiku synchronizace, kterou práce řeší.

2.3 Současné postupy tvorby rozvrhů

Pro proces tvorby rozvrhů existují různé počítačové programy, které tuto činnost zjednodušují. Ne všude jdou však používány. Některé školy stále používají metody bez využití moderních prostředků.

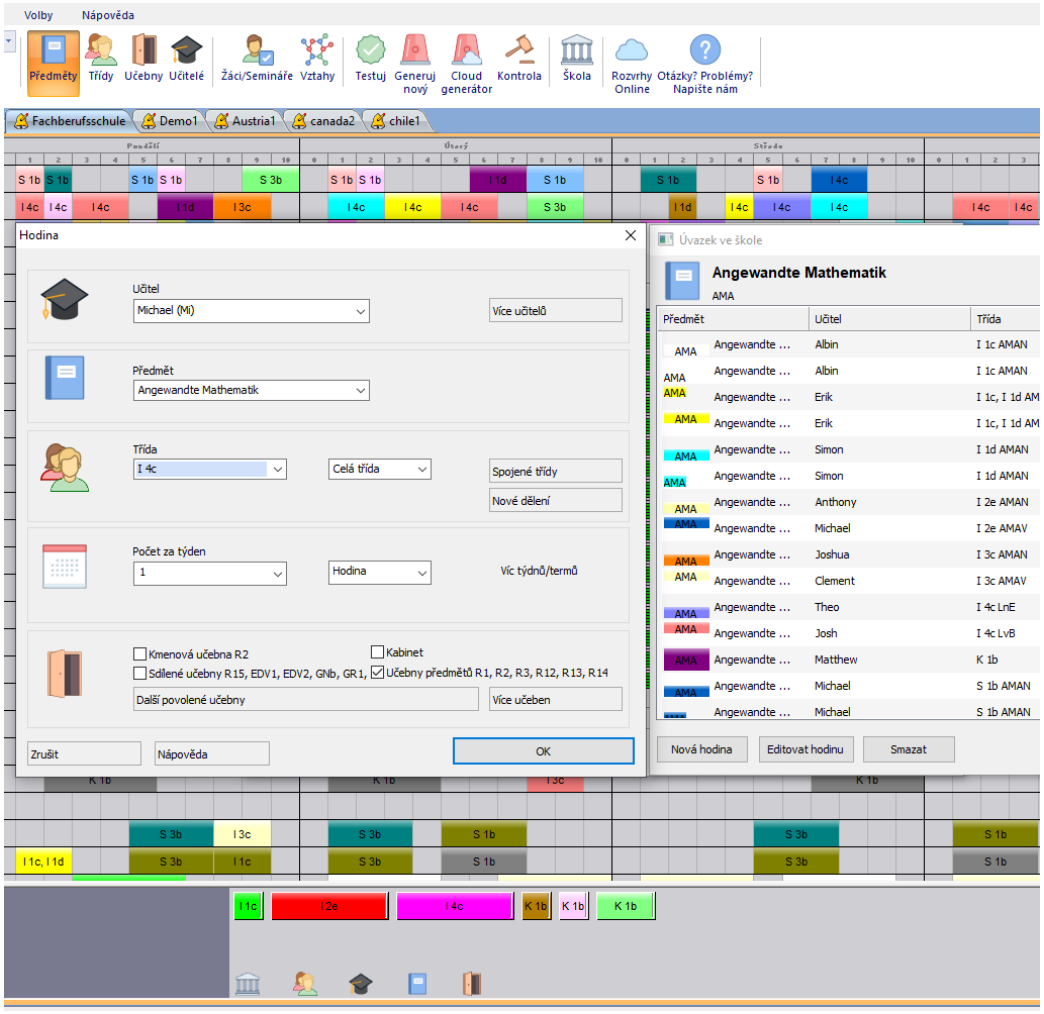


Obrázek 2.1: Tvorba letošního rozvrhu na ZŠ Hroznová v Brně (2022/2023)

Na obrázku 2.1 je příklad tvorby rozvrhu pro ZŠ Hroznová v Brně. Vytvoření rozvrhu pomocí magnetické **tabule s lístečky** trvá téměř dva týdny a i přes pečlivé kontroly dochází v prvním týdnu výuky ke kolizím a následným změnám. Zdrojem těchto informací a obrázku 2.1 je osobní konzultace s Mgr. Lenkou Krumpochovou.

U některých základních škol je důvodem použití ruční metody velké množství specifických požadavků, které buď trvá do programu zadat příliš dlouho, nebo software není schopen rozumně data vyhodnotit. U malých základních škol se investice nevyplatí zejména časově. Pokud rozvrh není komplikovaný, jeho vytvoření pomocí ruční tzv. lístečkové metody je otázkou jen několika dnů. [19]

Byly doby, kdy se plně manuální řešení používalo i na vysokých školách. V dnešní době již existuje velké množství programů, které mohou univerzitám tento úkol zjednodušit. Většina těchto těchto programů je licencovaných a nemají dostupnou zkušební verzi, případně u některých není modul pro tvorbu rozvrhů součástí zkušební verze. U žádné z nich nebyla z dostupných zdrojů dohledána informace o tom, že může na tvorbě rozvrhů spolupracovat více uživatelů současně.



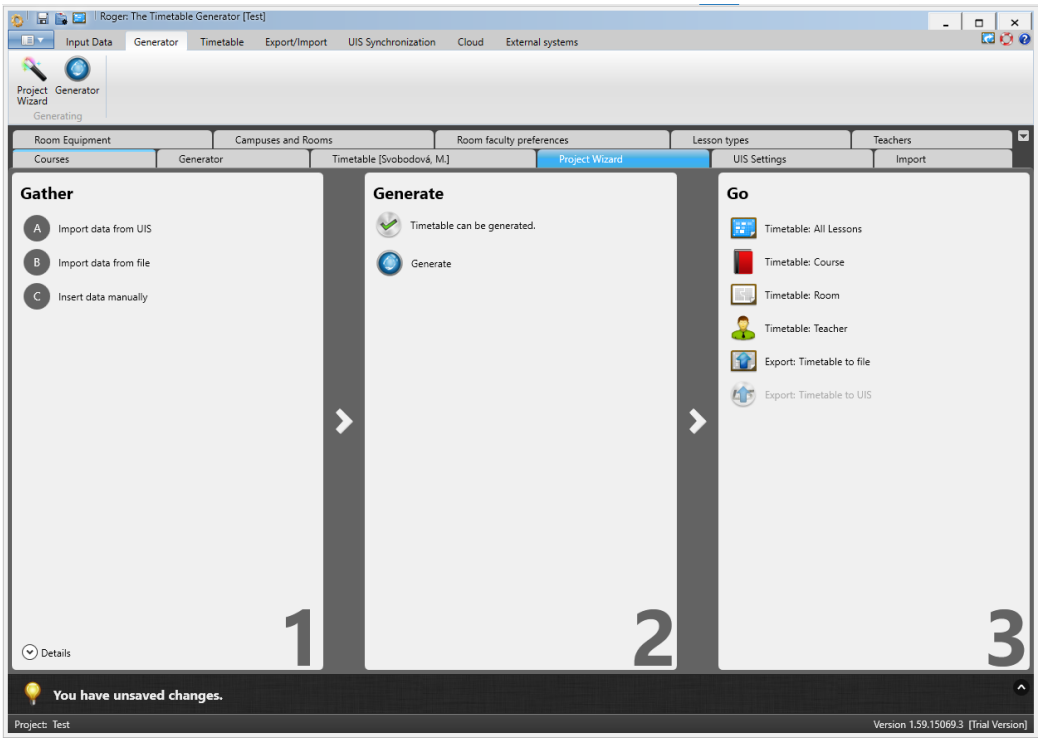
Obrázek 2.2: Formulář pro přidávání lekcí v aplikaci aSc timetables

- **Untis**[3] nemá ve zkušební verzi modul pro tvorbu rozvrhů. Z dostupných informací má tento software i webovou verzi a jeví se jako solidní řešení.

[3]https://www.untis.at/en/products/webuntis/online-timetable-scheduling

- **aSc timetables**[4] je velmi kvalitní program s intuitivním designem a bohatou funkcionalitou. Vzhledem k tomu, že se jedná o desktopovou aplikaci, je omezena pouze na operační systém Windows. S touto prací má společný systém přetahování lekcí ze zásobníku (spodní část obrázku 2.2), který je zde uplatněn na případné manuální úpravy po automatické generaci rozvrhu. Pro vkládání veškerých dat obsahuje aplikace mimo import přehledné formuláře, jako příklad je na obrázku 2.2 ukázán formulář pro vkládání lekcí.

2.4 RoGeR - software firmy IS4U



Obrázek 2.3: Úvodní obrazovka desktopové aplikace RoGeR

Firma IS4U nabízí v současné době **desktopovou aplikaci** RoGeR[5]. Její název vznikl ze slov Rozumný Generátor Rozvrhů.

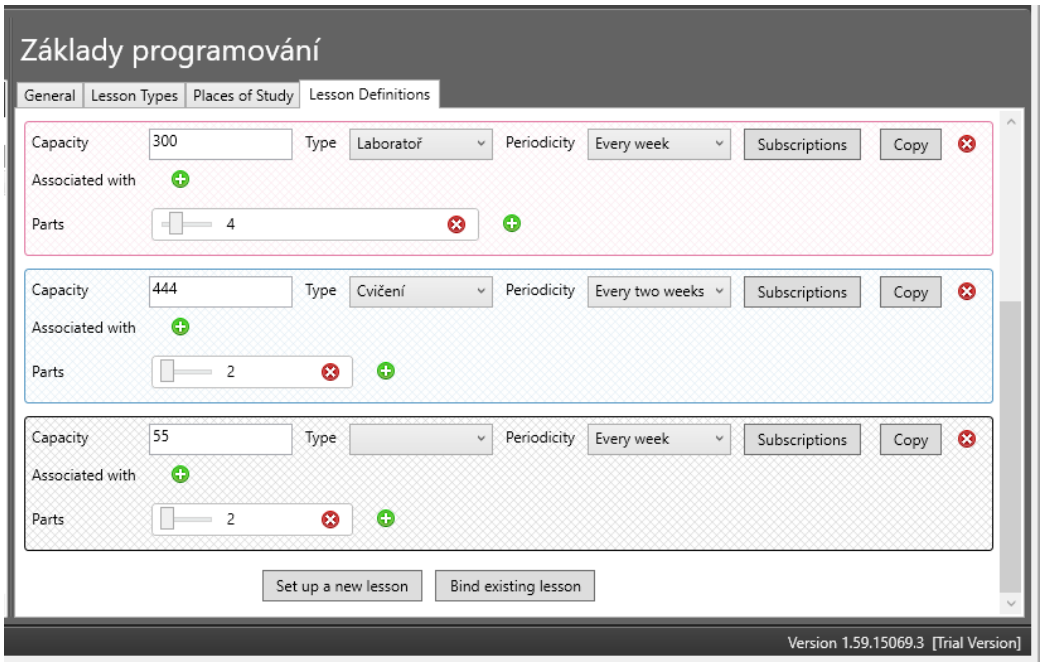
Velkou nevýhodou této aplikace je to, že je desktopová. Webové aplikace mají mnoho výhod. Jsou dostupné v každém zařízení s internetovým připojením a není nutná jejich instalace, aktualizace se provádí centrálně a není potřeba aktualizace na každém zařízení zvlášť. Navíc jsou webové aplikace nezávislé na operačním systému, a i díky tomu je zálohování a sdílení dat mezi uživateli mnohem jednodušší.

Po instalaci a spuštění aplikace se objeví úvodní obrazovka naznačující základní fungování aplikace, viz obrázek 2.3. V první fázi je nutné do systému vložit data. To lze provést třemi způsoby.

- Načtení dat přímo z univerzitního informačního systému.

[4] https://www.asctimetables.com/#!/home/info
[5] https://www.rozvrhy.eu/cs/index

- Import z textového souboru s určitou strukturou CSV nebo XML dat, JSON formát chybí.
- Vložení dat pomocí sady mnoha formulářů.

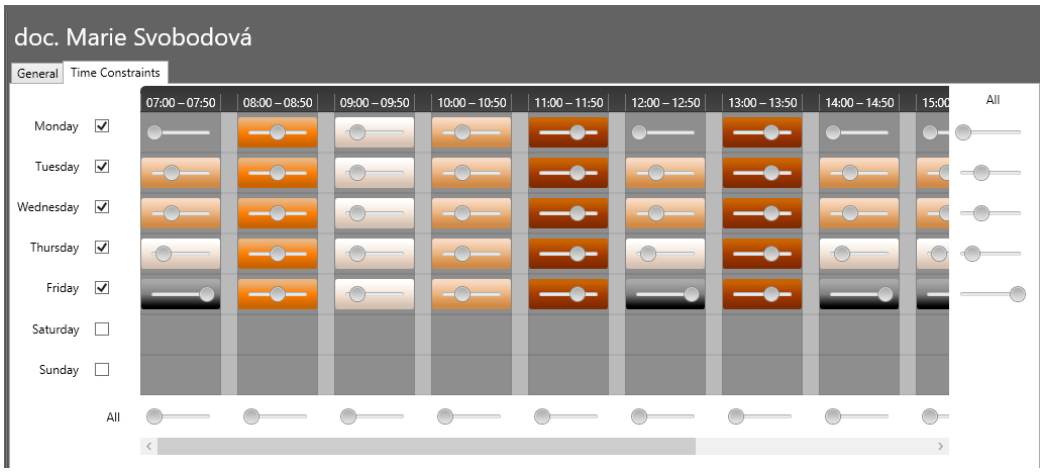


Obrázek 2.4: Při nevyplnění políčka pro typ lekce se nezobrazí žádné upozornění

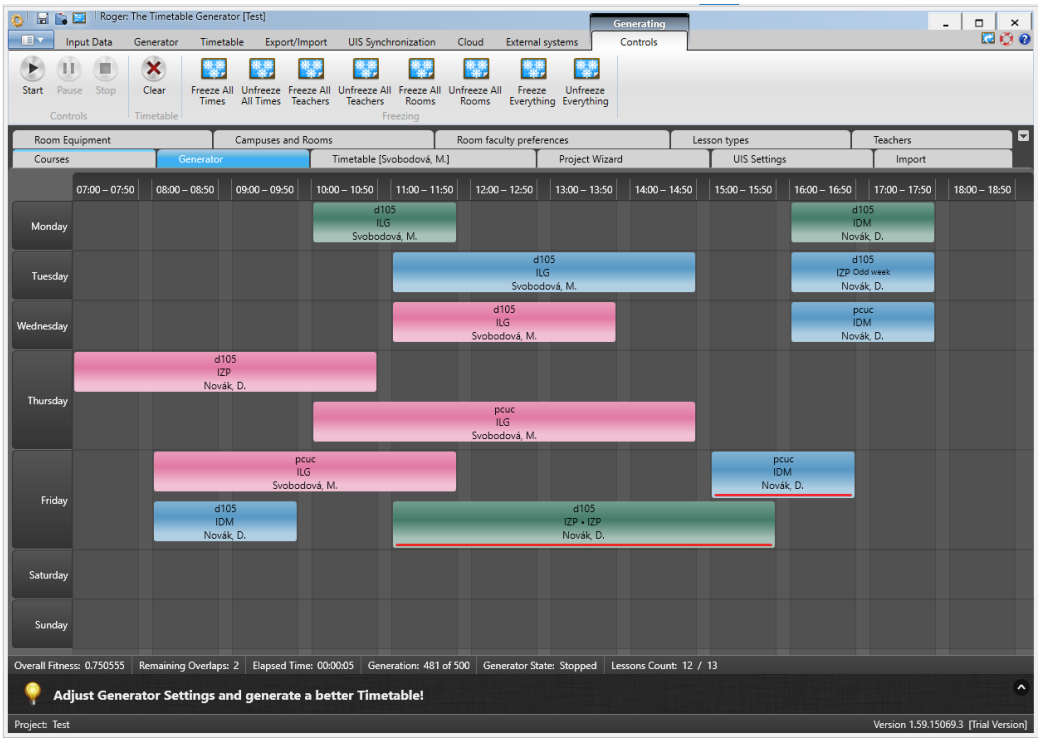
Pomocí těchto formulářů je možné data i modifikovat nebo odstranit. Příklady jsou uvedené na obrázcích 2.4 a 2.5. Hlavním problémem je nedostatečné upozornění uživatele na to, zda je vše vyplněno správně. Na tento nedostatek upozorňuje pouze malý vykřičník v levé části formuláře. Další nedostatek je ten, že lze vyplnit některá data před tím, než jsou do systému přidány jejich závislosti. Například na obrázku 2.4 je formulář, ve kterém lze vytvořit lekci, i pokud v systému neexistují žádné typy lekcí nebo jí žádný typ nepřiřadíme. V tomto případě se nezobrazí nikde žádné varování. A při následném pokusu o generaci rozvrhu aplikace spadne bez zobrazení jakékoliv hlášky. Dále si v tomto formuláři můžeme všimnout, že pole `Periodicity` může nabývat pouze dvou hodnot: `Every week` a `Every two weeks`. Toto je velké omezení, jelikož nelze zadávat méně pravidelné, či blokové lekce.

Zajímavé řešení, kterým je inspirována i tato práce, je **vkládání časových možností** pro přednášející (obrázek 2.5). Jako alternativa k posuvníkům, které reagují na kliknutí pouze na zlomku plochy pole, je v této práci použito klikání na celou buňku. Podle toho, jestli použijeme pravé nebo levé tlačítko myši, je preference snížena nebo zvýšena.

Jakmile se podaří do aplikace zadat úspěšně všechna data potřebná ke generaci, pomocí jednoho tlačítka se spustí **generace rozvrhu** a během krátkého času je vytvořen rozvrh (viz obrázek 2.6), který můžeme dále upravovat tažením myší, popřípadě některé lekce zmrazit pomocí tlačítek v horní nabídce a nechat vygenerovat zbytek rozvrhu znovu. Poté lze rozvrh exportovat do souboru nebo do univerzitního informačního systému.



Obrázek 2.5: Formulář pro úpravu časových možností přednášejícího v aplikaci RoGeR



Obrázek 2.6: Okno pro generování a následnou úpravu rozvrhů v aplikaci RoGeR

Kapitola 3

Specifikace požadavků a návrh aplikace

V následujících odstavcích jsou shrnuty požadavky od firmy **IS4U**, která je **zadavatelem** bakalářské práce a byly dohodnuty především na osobních setkáních s konzultantem práce, doktorem Milanem Šormem.

Jedná se o software pro tvorbu rozvrhů pro univerzity. Firma má stávající řešení, jejímž je desktopová aplikace RoGeR. Ta je ale už staršího data a nemá takovou funkcionalitu, jakou by chtěla firma svým zákazníkům poskytovat. Protože se jedná o netriviální a rozsáhlou aplikaci, není mým úkolem vytvořit finální verzi, ale pouze **demonstraci konceptu**, která bude použita při rozhodování, zda se vývoji nové aplikace věnovat.

3.1 Požadavky firmy

Cílem bylo vytvořit HTML5 nativní **editor**, který umísťuje jednotky vyučování či zkoušek do polopevné tabulky rozvrhu z nějakého **zásobníku** akcí. Polopevná tabulka znamená to, že primárně umisťujeme do rozvrhu jednotky na celé hodiny podle toho, na kterou buňku v tabulce rozvrhu jednotku umístíme. Sekundárně však můžeme ve výjimečných případech umístit jednotku na konkrétní hodinu a minutu, mimo definovanou pevnou osu. Aplikace musí podporovat dva typy rozvrhu.

- **Periodický**, kde se jednotky rozvrhu umisťují do časové mřížky každý týden, nebo s určitou četností, typicky každý lichý nebo každý sudý týden.
- **Blokový**, který se umisťuje na konkrétní den a v rámci celého rozvrhu se neopakuje.

V jednom okamžiku se používá jedna **časová osa**, která definuje začátky a konce hodin, přičemž ne každá univerzita používá stejné intervaly, proto bylo domluveno, že stačí mít osu po jednotlivých hodinách. Na univerzitě, pro kterou se rozvrh sestavuje, existují místnosti, které jsou dané svým názvem a svou kapacitou. Existuje **fyzická kapacita a virtuální kapacita**. U nepovinných lekcí se rozvrhuje na virtuální kapacitu, protože se dá očekávat určité procento studentů, kteří nechodí na přednášky.

V editoru je velký **zásobník lekcí**. To jsou de facto rozvrhové lístečky, zmíněné v předchozí sekci 2.3 o manuální tvorbě rozvrhů. Tyto lekce obsahují údaje, které jsou viditelně vidět a jsou potřeba pro tvorbu rozvrhu. Každý předmět má několik lístečků neboli lekcí, každá lekce má nějakou **délku**, **předmět** ke kterému lekce náleží, seznam **přednášejících** na dané lekci, **typ lekce**, například přednáška, cvičení, laboratoř, jiné. Bylo by vhodné,

kdyby to byl nějaký číselník. Dále má **počet studentů**, který lekci studuje, informaci zda je **povinná** a ve kterých týdnech se koná. Přednášky se v rozvrzích a na zásobníku **barevně odlišují**.

Editor by měl automaticky kontrolovat všechny typy **kolizí** a problémy mezi jednotlivými lekcemi. Lekce přitom mohou mít **další omezení** různých typů.

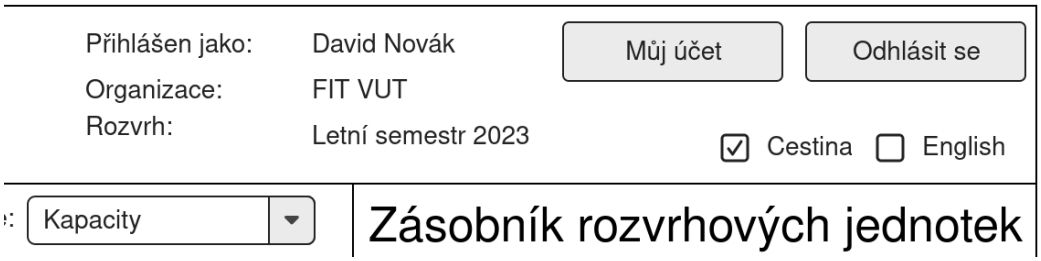
- Určitý typ lekce u určitého předmětu je vhodné učit v určitých místnostech.
- Přednášející si může určit, kdy opravdu **nemůže učit** (výčtem z osnovy periodického rozvrhu) a kdy mu to **pouze nevyhovuje**. Zde může být použito až pět různých stupňů intenzity. Na jejichž základě je vytvořena pro každého přednášejícího mapa vhodných a nevhodných časů, kdy může učit.
- Kdy nelze učit v konkrétní místnosti, tzn. je blokována pro něco jiného.
- Kolize studentů pro dva předměty, respektive procento studentů, kteří studují oba předměty současně.

Vstup dat, ze kterých se budou rozvrhy následně skládat, by měl být následující. Existují dvě možnosti, jak data vložit. Lze je zadat manuálně přes formuláře, nebo je importovat v JSON formátu, případně využít kombinaci těchto dvou metod. Data by mělo být možné přidávat, měnit a upravovat kdykoliv v procesu tvorby rozvrhů.

Na závěr je nutné vše online synchronizovat (například přes WebSocket) tak, aby v editoru mohlo pracovat více lidí současně.

Co nemusí aplikace řešit

V první iteraci návrhu aplikace bylo naplánováno velké množství funkcionalit, od kterých bylo po konzultacích upuštěno, jelikož by rozsah takové práce nebyl zvládnutelný. V prvotních wireframech[1] se objevila například funkcionalita **přepínání jazyků** aplikace mezi angličtinou a češtinou (obrázek 3.1), což se pro účel této práce, důkaz konceptu, ukázalo jako zbytečné.



Obrázek 3.1: Výřez z wireframu s přepínáním jazyků a s prvky autorizačního systému

Další věc, která na doporučení konzultanta nebyla implementována, je **autorizační systém** (jeho prvky jsou též na obrázku 3.1). Sice samotný název práce Webová aplikace pro **kolaborativní** tvorbu rozvrhů vytváří dojem, že by mělo existovat v systému více uživatelů, ale nemusí tomu tak být. Spolupracováním je zde myšlena zejména, možnost práce na rozvrhu současně z několika prohlížečů tak, aby data v editoru byla aktualizována v reálném čase. K těmto účelům proto není implementace uživatelů nutná.

[1] Wireframy jsou více rozebrány v kapitole 3.3

Dalším zjednodušením je fixní délka vyučovacích hodin[2]. Jelikož aplikace má být všeobecná a ne specifická pro nějakou konkrétní univerzitu, bylo by složité počítat se všemi různými variantami. Napříč univerzitami jsou různé délky vyučovacích hodin a různě dlouhé přestávky. Zároveň existují univerzity, kde jsou různě dlouhé přestávky, například delší přestávky v době oběda nebo je každá druhá přestávka delší. Přesto to bylo při návrhu aplikace částečně zvažováno a dodatečná implementace by neměla být příliš složitá.

Poslední věcí je tvorba rozvrhů pro více areálů, nebo tvorba více variant stejného areálu. Ani zde by implementace nebyla nijak náročná, ale pro demonstrativní účely není potřebná.

3.2 Návrh databáze

Datový model můžeme znázornit pomocí **Entity Realtionship Diagram**, zkráceně se označuje ERD. Je to abstrakce, která popisuje, jak jsou data uložena v databázi. Znázorňuje jejich strukturu a vztahy mezi nimi. ERD je hojně využíván v procesu tvorby aplikací. Jeho velkou výhodou je, že není nutné se při jeho tvorbě soustředit na konkrétní implementační detaily, v důsledku čehož řešíme pouze data, jejich strukturu a vazby mezi nimi.

Na obrázku 3.2 je ERD mojí webové aplikace pro kolaborativní tvorbu rozvrhů, který byl vytvořen před implementací databáze. Využit k tomu byl nástroj pro tvorbu diagramů a grafů, webová aplikace **diagrams.net**, dříve známá jako draw.io[3]. Pro znázornění vazeb mezi entitními množinami byla využita notace **Crow's foot**, jejíž historie sahá k článku Gordona Everesta z roku 1976. [7] Na obrázku je již finální verze, ale v průběhu vývoje aplikace se ERD měnil, aby umožnil implementaci nových funkcionalit, se kterými původní verze nepočítala.

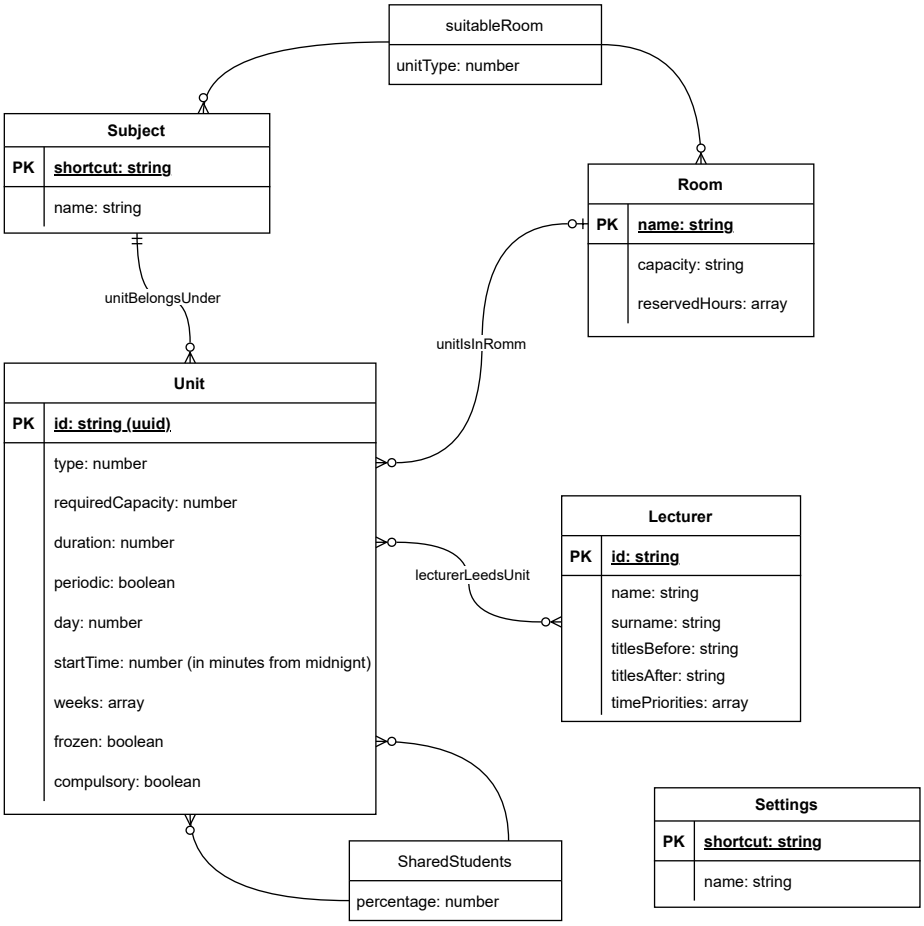
Lekce

Jsou zde čtyři entitní množiny: **Unit** (lekce), **Subject** (předmět), **Lecturer** (přednášející) a **Room** (místnost). První z nich, Lekce, používá jako identifikátor UUID, protože žádný atribut není unikátní. Její atributy jsou následující:

- `type` (typ) je v databázi číslo. V aplikaci je následně reprezentován výčtovým datovým typem `UnitType`. Označuje typ lekce ve smyslu přednáška, cvičení, zkouška atd. V průběhu navrhování bylo zvažováno vytvořit namísto atributu `type` další entitní množinu **Unit type** (typ jednotky). Umožnilo by to přidávání a odebírání typů lekce za běhu aplikace. Nakonec bylo rozhodnuto od tohoto řešení upustit s tím, že pokud bude vestavěná nabídka typů aplikace dostatečně široká, nebude potřeba vytvářet další typy.
- `requiredCapacity` (požadovaná kapacita) je počet studentů studujících danou lekci, nebo také kapacita místnosti, která je potřeba, aby se v ní mohla konat tato lekce.
- `compulsory` (povinnost lekce) nese pravdivostní hodnotu, zda je lekce povinná nebo ne. Tento údaj je důležitý jak v kontextu toho, zda můžeme rozvrhovat lekce na virtuální nebo reálnou kapacitu, tak i kvůli kontrole, zda se současně nekonají povinné lekce, které musí navštěvovat ti stejní studenti.
- `periodic` (periodičnost) udává, zda je lekce periodická nebo bloková.

[2] Zde v tomto kontextu je vyučovací hodinou myšlena velikost mřížky v rozvrhu. Přednášky nebo laboratoře zpravidla trvají i několik těchto vyučovacích hodin.

[3] https://www.diagrams.net/



Obrázek 3.2: Entity relationship diagram databáze

- `day` (den) je den v týdnu, kam je lekce v rozvrhu zařazena. V aplikaci je reprezentována pomocí výčtového typu `WeekDaysType`.
- `startTime` (čas začátku) vyjadřuje čas, kdy lekce začíná. Aby nebylo nutno použít složitější datový typ, byla použita inspirace z Unixového času a čas začátku lekce je ukládán jako počet minut od půlnoci.
- `weeks` (týdny) je pole týdnů, ve kterých se lekce koná. U periodických se rozvrhováním lekce týdny v poli nemění, blokové lekce je mají prázdné a rozvrhnutím do konkrétního týdne se do pole vloží číslo týdne, ve kterém se konají.
- `frozen` (zmrazená) udává pomocí pravdivostní hodnoty, zda je lekce zmrazená (nelze s ní v editoru hýbat).

Entitní množina lekce má rekurzivní vazbu **SharedStudents** (sdílení studenti) sama se sebou, respektive s jinou instancí stejné entitní množiny. Tato vazba má jeden atribut `count` (počet), který udává počet studentů, kteří současně studují obě lekce.

Přednášející

Dále má Lekce vztahy se všemi třemi ostatními entitními množinami. S přednášejícím je M:N vazba, protože nejen že jeden přednášející pochopitelně může vést více lekcí, ale apli-

kace myslí také na možnost, kdy je lekce vedena více přednášejícími. Atributy přednášejícího jsou:

- `id` (identifikátor) je UUID, stejně jako u lekce. Slouží jako primární klíč.
- `name` jméno přednášejícího
- `surname` příjmení
- `titlesBefore` tituly před jménem
- `titlesAfter` tituly za jménem
- `timePriorities` (časové priority) Je to pole 168 číselných hodnot (7 dnů krát 24 hodin). Ty značí pro každou jednotlivou hodinu, zda přednášející nemůže učit nebo rozpětím 5-ti hodnot, zda se mu to nehodí, až po to, že mu daný čas plně vyhovuje. Tento systém šesti hodnot je navržen na základě specifikace požadavků od firmy a byl již použit v jejich desktovové aplikaci RoGeR.

Místnost

Další vazbu **unitIsInRoom** (lekce je v místnosti) má lekce s místností. V Místnosti může být mnoho lekcí, ale lekce se může konat pouze v jedné místnosti. Místnost má tři atributy:

- `name` (název) je unikátní, a proto je i primárním klíčem této entitní množiny.
- `Capacity` je skutečná kapacita místnosti. Od ní se poté počítá virtuální kapacita, která slouží pro rozvrhování nepovinných lekcí.
- `Reservations` (rezervace) jsou, podobně jako priority přednášejícího, pole 168 hodnot, kde každá hodnota značí, zda je během dané hodiny místnost k dispozici pro rozvrhování v této aplikaci, nebo je rezervována pro něco jiného, popřípadě je uzavřen areál, ve kterém se místnost nachází.

Předmět

Poslední vztah lekce je s předmětem. Předmět má zpravidla více lekcí, lekci však může zastřešovat pouze jeden předmět.

Předmět má dva atributy. První, který je současně primárním klíčem, je **abbreviation** (zkratka). Tato zkratka se používá v aplikaci mnohem častěji než druhý atribut, kterým je **name** (jméno).

Poslední vazba **preferedRoom** (vhodná místnost) mezi předmětem a místností s atributem **unitType** (typ lekce) nese záznam, zda je daná místnost pro výuku tohoto předmětu a typ jeho lekce vhodná.

Nastavení

Samostatně stojící entitní množina `Settings` ukládá nastavení aplikace pomocí atributů `name` a `value`.

- `name` ukládá název nastavení.

- `value` nese hodnotu nastavení, která je uložena jako řetězec ve formátu JSON. Toto řešení umožňuje flexibilně ukládat jakýkoliv datový typ nastavení. Řetězec není nejlepší způsob, jak ukládat data, protože zabírá více paměti než například číslo. Jelikož však tato tabulka nebude při používání aplikace nijak narůstat, je toto řešení přijatelné.

V budoucnu při implementaci produkční aplikace, nebo při rozšíření konceptu o autorizační systém nebude tato entitní množina bez vazby, ale bude navázána na uživatele.

Časové priority přednášejících a rezervace místností

Tato data je vhodné reprezentovat pomocí nějaké matice nebo pole. Bylo na výběr, jestli maticovou strukturu zanést do databáze podobným způsobem, jako bylo navrženo u sdílených studentů, nebo ji do databáze zanést zjednodušeně. U sdílených studentů byl hlavní důvod ten, že velikost matice závisí na počtu lekcí a může libovolně narůstat. Neznáme tedy maximální velikost a proto je nutné dynamické řešení. Naopak časové priority a rezervace mají pevný počet hodnot, proto lze tolerovat v databázi reprezentaci pomocí řetězce. Navíc TypeORM má pro ukládání jednoduchých polí jako řetězců vestavěnou funkcionalitu.

3.3 Návrh uživatelského rozhraní

K vytvoření návrhu uživatelského rozhraní byly využity **wireframy**. Slovo wireframe by se dalo do češtiny volně přeložit jako kostra stránky nebo základní rozvržení. Je to vizuální reprezentace návrhu webové stránky nebo aplikace, která ukazuje, jak bude výsledný produkt vypadat po stránce funkcionality. U wireframů se neklade důraz na grafické detaily, například na barvy tlačítek nebo styly písma. Pro tento účel by sloužil **Mockup**. Ke tvorbě wireframů byl vyžit webový online nástroj **Moqups**[4]. Jeho bezplatně dostupná verze má sice omezené možnosti, ale základní funkcionalitu pro rychlou tvorbu přehledného wireframu poskytla. Na internetu je dostupných mnoho podobných nástrojů, vždy s různě omezenou funkcionalitou bezplatné verze. Moqups byl vybrán proto, že ve své bezplatné verzi poskytoval veškerou funkcionalitu a předdefinované komponenty v editoru, které bylo třeba použít. Jediná nevýhoda byla chybějící možnost exportu wireframů ve vektorovém formátu. Pro větší projekty by však tato aplikace byla ve své neplacené verzi téměř nepoužitelná.

Návrh editoru

Na obrázku 3.3 je první návrh, jak by aplikace mohla vypadat. Byla do něj vhodně zapracována veškerá požadovaná funkcionalita ze sekce 3.1. Nahoře byl ponechán horní panel se základní navigací mezi jednotlivými částmi aplikace. Spodní část byla rozdělena na dvě části. V pravé, menší části je zásobník rozvrhových jednotek, ze kterého se jednotky posouvají tažením do části levé, ve které se nachází plátno editoru. To poskytuje dva režimy zobrazení - celosemestrální a týdenní. Tyto režimy jsou popsány v kapitole 4.3.2.

Po konzultacích byly odstraněny některé funkce, které byly již zmíněny v sekci 3.1. Také se objevila potřeba místa pro zobrazení kolizí, jelikož to bylo v původním návrhu opomenuto. V ostatních aplikacích, které byly v rámci průzkumu prozkoumány, byl přístup k nim

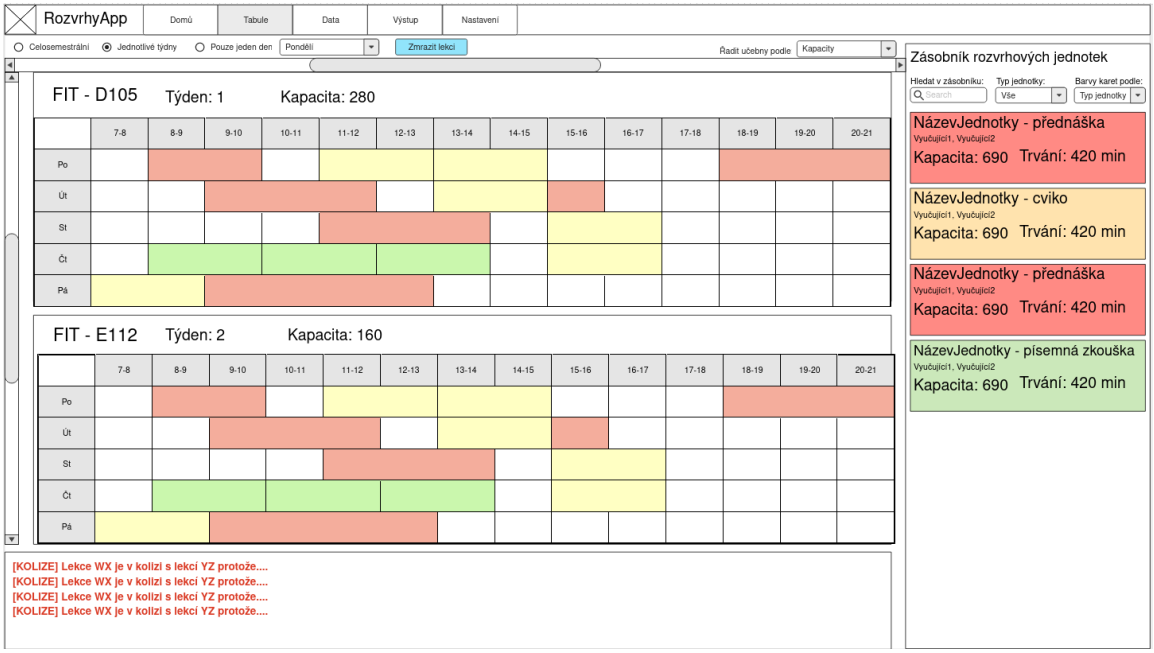
[4] https://moqups.com/

Obrázek 3.3: První verze návrhu editoru

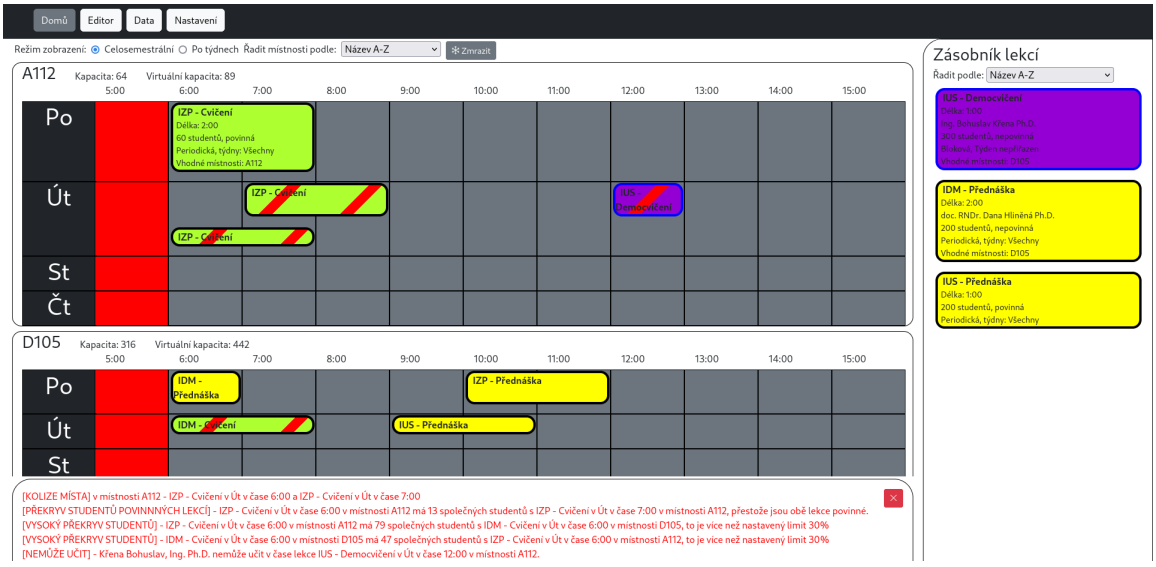
po kliknutí, nebo po najetí myší ve vyskakovacím okně, případně v nějaké záložce mimo editor. Nejpřehlednější řešení bylo zobrazení kolizí v seperátním okně. Byla proto přidána pod plátno zobrazovací konzole. Této konzoli musela ustoupit nahoře lišta s orientací v týdnech v týdenním režimu zobrazení. Tento údaj se přesunul do hlavičky rozvrhu k informacím o učebně, kde je na to dostatek místa.

Pro ještě větší ušetření místa byly v pozdější fázi vývoje skryty přebytečné informace na kartách umístěných do rozvrhu tak, aby nezabíraly příliš mnoho místa, a aby byl editor přehlednější. Skryté informace se zobrazí jen v momentu, kdy je na kartu najeto myší.

Další funkcionalita, která přibyla je tlačítko zmrazit lekci, wireframe na obrázku 3.4 světle modře zobrazené. Když je aktivováno, kliknutím na jakoukoliv lekci ji zmrazíme a nelze s ní hýbat, dokud není zase identickým postupem rozmrazena. V průběhu implementace již nedošlo k žádným velkým změnám. Výsledná podoba editoru je na obrázku 3.5.



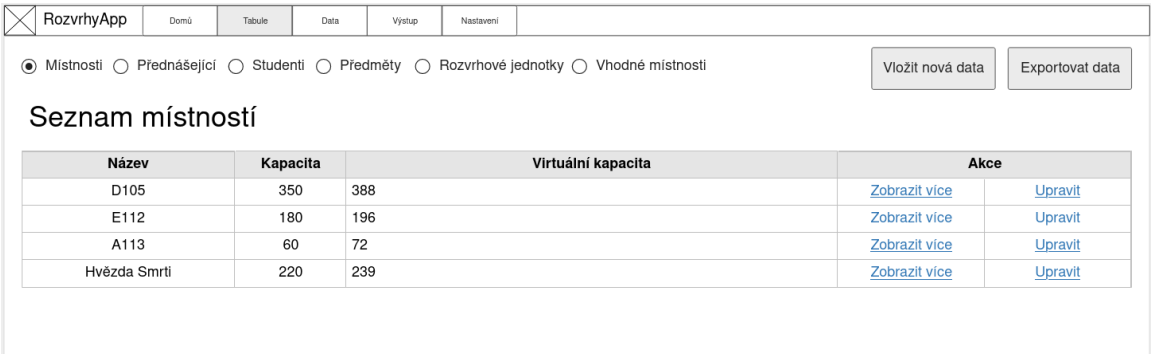
Obrázek 3.4: Poslední iterace wireframu s návrhem editoru



Obrázek 3.5: Výsledná podoba editoru

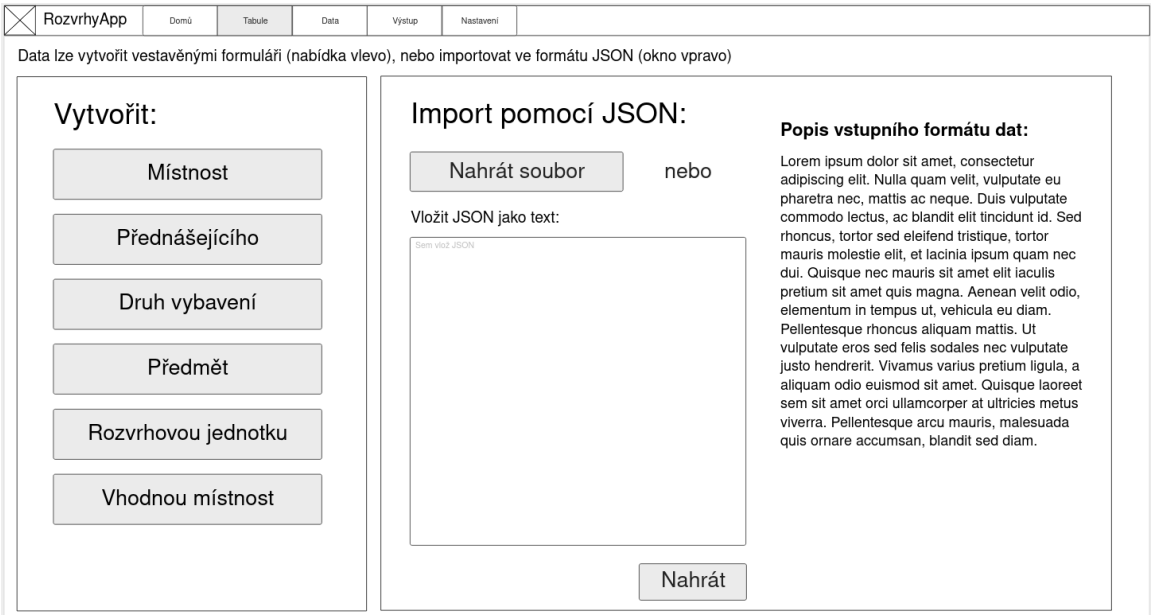
Návrh ostatních prvků aplikace

V následujících wireframech je uvedena pouze finální verze, neboť v nich už nedocházelo ke změnám v průběhu jednotlivých iterací.



Obrázek 3.6: Návrh hlavní obrazovky záložky pro správu dat

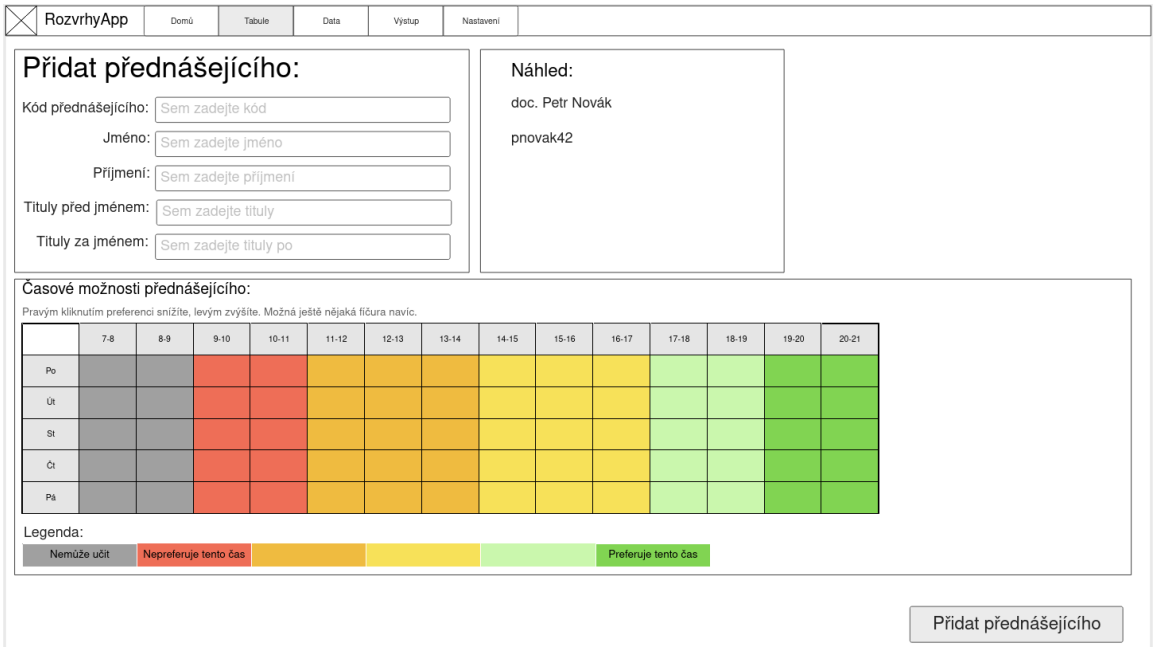
Záložka data slouží pro **správu dat**. Na hlavní obrazovce této záložky, viz 3.6 si můžeme zobrazit souhrn všech dat vybraného typu v tabulce. Zde si můžeme si o jednotlivých položkách zobrazit více informací, nebo je smazat. V pravém horním rohu se můžeme pomocí tlačítek přesměrovat na stránky s přidáváním a exportem dat.



Obrázek 3.7: Návrh menu pro přidávání dat

Na wireframu 3.7 je navrženo menu pro **vkládání dat** do aplikace. Byly implementovány dvě možnosti, a to buď přidávat data pomocí formulářů, nebo pomocí souboru ve formátu JSON s pevně danou strukturou.

Návrhy všech **formulářů** jsou velmi podobné, lišící se prakticky pouze v typu, názvu a počtu vstupů, které uživatel zadává. Ani wireframy proto nebyly tvořeny pro každý z nich, ale pouze pro nastínění konceptu u komplikovanějších z nich.



Obrázek 3.8: Návrh formuláře pro přidávání přednášejících

Jako příklad je zde popsáno přidávání přednášejících, viz obrázek 3.8. První políčko, kód přednášejícího, bylo v průběhu implementace odebráno kvůli konzistenci s ostatními formuláři, jelikož se manuální zadávání identifikátoru jinde též nevyžaduje. Místo toho se přednášejícímu v databázi vygeneruje UUID, stejně jako u některých jiných dat. Zbylá políčka jsou poměrně přímočará. Zajímavá je tabulka na **vkládání časových možností** a priorit přednášejícího. Každé políčko této tabulky je obarveno barvou podle toho, jak přednášejícímu daný čas vyhovuje. Tabulka je vždy implicitně vyplněna podle vzoru uloženého v nastavení. Tyto implicitní hodnoty lze dále upravovat pomocí tlačítek myši. Pravým kliknutím tlačítkem myši na buňku tabulky se preference sníží, levým tlačítkem zvýší. Při přetečení nebo podtečení se cyklicky přeskočí na opačnou stranu spektra.

Kapitola 4

Implementace

Tato kapitola popisuje vývoj aplikace na základě návrhů zpracovaných v kapitole 3. Zajímavé části implementace jsou popsány detailně, zbylé jen okrajově.

4.1 Základní architektura a použité technologie

Volba vhodné architektury, frameworků a knihoven je jednou ze základních věcí při vývoji aplikace. Odvíjí se od nich **struktura** zdrojového kódu **aplikace** a jeho dělení do souborů a adresářů. Dobře zvolená architektura zlepšuje čitelnost a srozumitelnost kódu a snižuje riziko chyb v implementaci aplikace. Vhodně zvolené frameworky či knihovny usnadňují práci díky tomu, že poskytují předem vytvořené funkce, které není nutné implementovat od základu.

Klient-server

Webová aplikace je v této práci postavena na architektuře klient-server, která spočívá v tom, že aplikace (klient) komunikuje s jinou aplikací (serverem) prostřednictvím počítačové sítě. Aplikace běží na jednom serveru, se kterým může komunikovat více klientů současně. Nejčastěji komunikuje klient se serverem pomocí tzv. **HTTP Push and Pull**, kdy klient zasílá serveru požadavky a server na ně odpovídá. Tento přístup má jednu zásadní nevýhodu, a to tu, že komunikace je vždy inicializována klientem. Server tedy nemůže klientovi zaslat data kdykoliv, ale jen když je o to žádáno. Vhodné je i využití AJAXu, který umožňuje klientovi aktualizovat pouze část webové stránky, bez nutnosti načítat vždy veškerý obsah znovu. Dále existuje **Long polling** a **WebSocket**. Tyto metody umožňují serveru odesílat data, kdykoliv je server potřebuje odeslat. [3] Jsou popsány v sekci 4.1.

MVC

Architektonický vzor je obecně řešení často se vyskytujícího problému v architektuře softwaru. Existuje množství architektonických zdrojů. V odvětví vývoje aplikací se nejčastěji používá MVC, na kterém je založena i aplikace v této práci. V architektuře MVC je oddělena logika aplikace od uživatelského rozhraní, a díky tomu jsou aplikace jednodušší na vývoj a údržbu. [23] MVC je iniciálová zkratka slov Model, View, Controller. Tento vzor vytvořil Trygve Reenskaug na konci sedmdesátých let. Dříve se používal při tvorbě grafických uživatelských rozhraní desktopových aplikací, dnes už je rozšířený i ve webových aplikacích. [23] Architektonický vzor MVC rozděluje aplikaci na tři propojené části:

- **Model** reprezentuje data a bussiness logiku. Ukládá data a provádí nad nimi různé operace.
- **View** (česky pohled) je zodpovědný za renderování uživatelského rozhraní. Zobrazuje data uživateli, uživatel skrz něj interaguje s celou aplikací.
- **Controller** (česky řadič) funguje jako prostředník mezi pohledem a modelem. Přijímá vstupy od pohledu, aktualizuje podle nich Model a reflektuje změny zpátky na pohled.

Podobným architektonickým vzorem je **MVVM**, vytvořený softwarovými architekty ze společnosti Microsoft. Narozdíl od kontroléru u MVC používá ViewModel, který je zodpovědný za správu dat a jejich vystavení na pohled.[10]

TypeScript

JavaScript je moderní objektově orientovaný programovací jazyk, který dnes používá takřka každá webová stránka. Umožňuje webovým stránkám větší responsibilitu a atraktivitu k používání. Spousta tradičních desktopových aplikací se přesouvá na web, protože poté není potřeba je pro použití stahovat a instalovat. To dělá JavaScriptu ve světě informatiky silnou pozici. Zároveň běhové prostředí Node.js umožňuje JavaScriptu běžet i na serverové straně, od interakcí s databází, až po renderování stránek pro zobrazení v prohlížeči. Také v něm mohou být psány mobilní aplikace, mikrokontroléry a další. [20]

Jedna z nevýhod JavaScriptu je ta, že je netypový. Tuto nevýhodu vyřešil, alespoň co se týče statické typové kontroly, jazyk TypeScript. Ten byl vydán v roce 2012 a velmi rychle ovlivnil JavaScriptovou vývojářskou komunitu. Mnohé projekty velkých nadnárodních společností se na něj brzy rozhodly přejít, jako například Adobe nebo Mozzila. [20]

TypeScript je **jazyk** a zároveň **sada nástrojů** pro generování JavaScriptu. Byl navržen Andersem Hejlsbergem v Microsoftu, který vytvořil například i C#, Turbo Pascal, Delphi a další. TypeScript používá překladač, který generuje JavaScript, tudíž může být spuštěn všude tam, kde může běžet i JavaScript. Proto musíme mít stále na mysli, že běhová typová kontrola neexistuje, protože kód běží jako JavaScriptový. [20]

Pro většinu běžně používaných JavaScriptových knihoven a frameworků existují **typové balíčky**, díky kterým se velmi rychle rozšířil ekosystém technologií, se kterými lze TypeScript používat. [20]

MySQL databáze

MySQL[1] je světově nejpopulárnější open-source **relační databáze**. [13] Ke správě a organizaci dat používá jazyk **SQL**. První vydání této oblíbené databáze bylo v roce 1995 a nyní je jedním z nejpoužívanějších databázových systémů. [8] Další výhodou je to, že je jednoduchá na instalaci a používání, také má kvalitní dokumentaci, takže je často vhodnou volbou pro začínající vývojáře. Nabízela se možnost použít i jiné alternativy, jako například PostgreSQL nebo nerelační MongoSQL, ale žádná z nich nebyla pro tuto práci výrazně lepší či horší. Protože není vyžadována žádná pokročilá funkcionalita, bylo zvoleno použití MySQL.

[1] https://www.mysql.com/

Identifikátory

Pro všechny entity v databázi, které nemají nějaký unikátní atribut, používám jako primární klíč UUID. Zkratka UUID značí **Univerzální Unikátní IDentifikátor**. Má délku 128 bitů a je navržen tak, aby byl globálně unikátní. Unikátnost není garantována nějakou centrální autoritou, jako například u IP adres. Její princip unikátnosti spočívá v tom, že 128 bitů poskytuje takové množství kombinací, že pro praktické účely reálného světa lze náhodně generované identifikátory považovat za unikátní. [15]

TypeORM

Nejprve byla databáze implementována bez objektového relačního mapování. Byla napsána v MySQL skriptu a na serveru byl repozitář naprogramovaný od základů. Jak se aplikace rozrůstala, časté chyby způsobené nedostatky v implementaci repozitáře brzdily vývoj aplikace. Proto byl zvolen nástroj, aby implementace úkonů souvisejících s databází byl efektivnější. Repozitář a inicializační skript byl z implementace odstraněn a nahrazen knihovnou pro objektově relační mapování TypeORM[2].

Zkratka ORM znamená **objektově relační mapování**. Takové mapování zjednodušuje interakci objektově orientovaných jazyků s databází. Umožňuje manipulaci s objekty nezávisle na datových zdrojích. Jedna z největších výhod tohoto přístupu je, že při programování lze zůstat v objektovém paradigmatu, které je pro mnohé vývojáře přirozenější, namísto práce v relačním paradigmatu. Byť ORM může být pomalejší než čistý SQL kód v důsledku režie, celkově však poskytuje mnoho výhod a je široce užíván. Jeho správné použití vede k rychlejšímu vývoji aplikací a snížení chybovosti. [2]

Při výběru byly ještě zvažovány knihovny Sequelize[3] a Prisma[4]. Sequelize nebylo shledáno vhodným, přestože má požadovanou funkcionalitu, dokumentace není moc kvalitní a při pokusech něco základního naprogramovat nebyla práce s ním přívětivá. Prisma má k práci s databázemi jiný přístup než TypeORM a Sequelize, ale pro tuto práci vhodná není. Bylo vhodnější ji zvolit, pokud bych využíval GraphQL, který nevyužívám. [1] Nakonec bylo vybráno TypeORM, se kterým byla práce od prvních pokusů bezproblémová. Má kvalitní srozumitelnou dokumentaci a podporu úplně pro vše, co bylo v práci potřeba. Součástí TypeORM je též dobře zpracovaný Query builder, umožňující vytvářet pokročilejší SQL dotazy, který byl brán při výběru jako plus. Nakonec však v implementaci stačily vestavěné metody, ale v případném budoucím rozšíření aplikace může být velmi užitečný.

Node.js

Při diskuzi se spolužáky často znělo špatné tvrzení, že Node.js[5] je framework. To není úplně pravda, protože Node.js je **JavaScriptový runtime**, což znamená, že je to prostředí pro běh JavaScriptového kódu mimo prohlížeč. To znamená, že Node.js umožňuje vývojářům psát serverový kód v JavaScriptu, ale neobsahuje specifické nástroje pro tvorbu webových aplikací jako framework. Takže i když Node.js sám o sobě není frameworkem, je často používán v kombinaci s frameworky jako Express nebo Nest.js. Tyto frameworky poskytují specifické nástroje a funkcionalitu pro tvorbu webových aplikací a lze je snadno použít s Node.js pro tvorbu webových aplikací a API. [5]

[2] https://typeorm.io/
[3] https://sequelize.org/
[4] https://www.prisma.io/
[5] https://nodejs.org/en

Express

Express[6] je nejoblíbenější webový **Node.js framework** a je základní knihovnou řady dalších populárních frameworků. Umožňuje například zpracování a odeslání odpovědi na HTTP požadavky, nastavit běžná nastavení webové aplikace nebo přidat do procesu zpracování HTTP požadavku tzv. middleware. Middleware je funkce, která například automaticky převádí tělo požadavku z JSON formátu nebo nastavuje HTTP hlavičky. [4]

Vue.js

Vue.js[7] je moderní JavaScriptový **frontendový framework** s podporou TypeScriptu, podobně jako React nebo Angular. Byl vyvinut v roce 2014 a získal si popularitu díky své jednoduchosti a flexibilitě. Oproti starším frameworkům, například jQuery, přináší mnohem vyšší úroveň abstrakce. Díky ní je výsledný kód kratší a přehlednější, velmi usnadňuje vývojářům práci. Jednou z klíčových funkcí Vue.js je reaktivní systém, který umožňuje propojení datového modelu s uživatelským rozhraním a automatickou aktualizaci uživatelského rozhraní při změně dat. [17]

Vue.js benefituje z architektonického vzoru SPA - **Single Page Application**.Tento vzor dodává webům prostředí bližší k desktopovým aplikacím. Na začátku načte jednu HTML stránku a dynamicky mění její komponenty podle toho, jak uživatel interaguje s aplikací. Komponenta je základní stavební jednotkou ve Vue.js. Může to být malé tlačítko, nebo celý formulář. Každá komponenta je soběstačná a komunikuje s ostatními pomocí atributů a událostí. Jedna komponenta může být využívána na mnoha místech v aplikaci a podle dat, která do ní vložíme, přizpůsobuje svůj obsah dle účelu, pro který je použita. Komponenty jsou definovány v separátních souborech a každá obsahuje tři části kódu podle účelu, viz následující ukázka. [17]

```
<template>
    <div>
        <h1> {{ WeekDaysType[day] }}</h1>
    </div>
</template>
<script setup lang="ts">
    import { WeekDaysType } from '@/interfaces/API';
    interface Props {
        day: WeekDaysType
    }
    const props = defineProps<Props>();
</script>
<style scoped>
    h1{
        text-align: center;
    }
</style>
```

První část, šablona, definuje to, z čeho se komponenta skládá. Syntakticky se jedná o XML, ve kterém jsou různé HTML elementy, elementy specifické pro Vue.js a odkazy na další

[6] https://expressjs.com/
[7] https://vuejs.org/

komponenty. Druhou částí je skript. Jedná se o JavaScriptový, popřípadě TypeScriptový kód, který načítá a upravuje data, která se v šabloně zobrazují. Třetí částí jsou kaskádové styly (CSS) které se starají o výsledný vzhled šablony.

Vue.js má ve svém ekosystému široké spektrum knihoven a pluginů. Ve této práci byly využity ty, které jsou popsány v následujících sekcích.

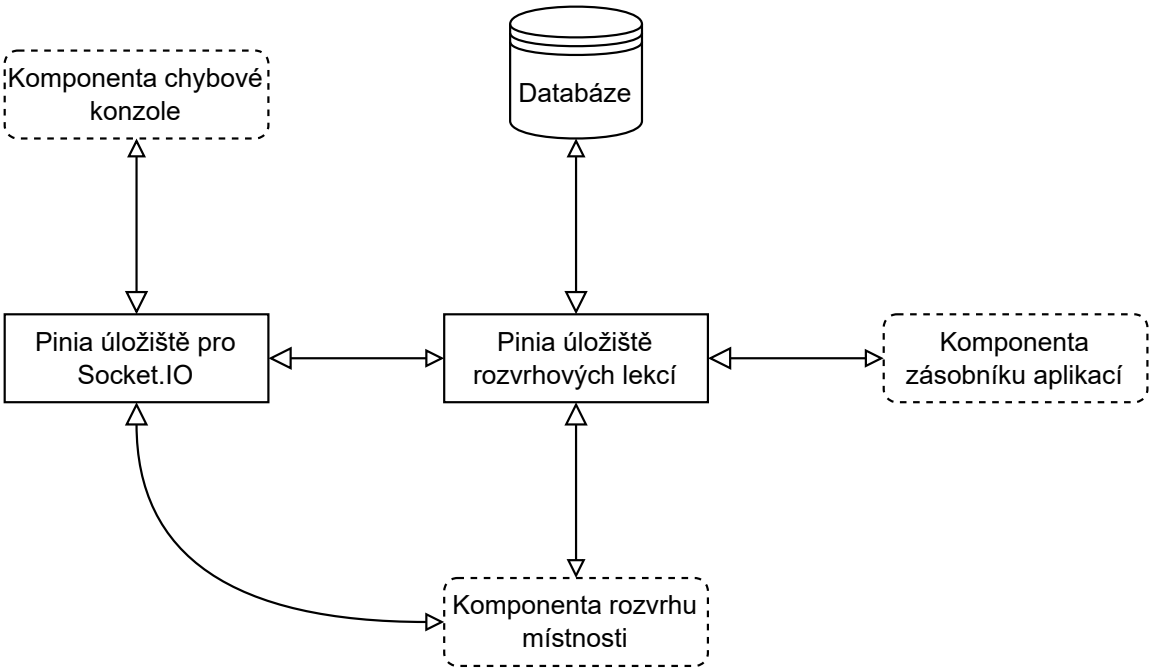
Vue-router

Vue-router[8] je oficiální knihovna pro Vue.js. Slouží k **řízení navigace** mezi jednotlivými stránkami aplikace vytvořené ve Vue.js.

Vue-router umožňuje definovat cesty (routes) v aplikaci a mapovat je na odpovídající komponenty Vue.js. Tímto způsobem může uživatel přepínat mezi různými částmi aplikace bez nutnosti načítat stránky znovu. Router také umožňuje použití parametrů a dynamických cest, což znamená, že aplikace může být dynamicky vykreslena na základě dat získaných ze serveru nebo z uživatelského vstupu.

Pinia

Pinia je modernější verze Vuex, systému pro **správu stavu aplikace**. Umožňuje vytvořit store (centrální úložiště), ve kterém je uložen stav aplikace. Stav aplikace jsou data, která aktuálně v systému jsou. Toto úložiště má hlavní výhodu v tom, že může být sdíleno mezi více komponenty zároveň, takže odpadá nutnost sdílet data skrze atributy a události. Ty jsou zbytečně složité a nepřehledné zejména v situacích, kdy si předávají mezi sebou data komponenty, které jsou různě zanořené. Také je umožněno data jednoduše modifikovat pomocí metod zvaných actions (akce).



Obrázek 4.1: Příklad komunikace mezi jednotlivými komponentami s použitím Pinia úložišť

[8] https://router.vuejs.org/

Na obrázku 4.1 je příklad z implementační části této práce, jak probíhá komunikace mezi jednotlivými úložišti a komponenty. Je vidět, že díky tomu, že jsou data uložena na jednom místě, není potřeba je opakovaně získávat ze serveru pro každou komponentu. Také může být jednotlivé úložiště modifikováno jiným úložištěm na základě jeho stavu. Celkově Pinia poskytuje obrovské zjednodušení správy dat v aplikaci a její správné použití zlepšuje kvalitu kódu i efektivitu jeho tvorby.

Bootstrap

Na **úpravu vzhledu** jednotlivých **HTML elementů** byl použit frontendový framework Bootstrap 5[9]. Bootstrap byl navrhnut a vytvořen roku 2010 ve Twitteru a stal se jedním z nejpopulárnějších frameworků a open-source projektů na světě. Jeho pátá veze vyšla v roce 2021, její součástí byla implementace z jQuery na normální JavaScript, a tím došlo k odstranění závislost na frameworku jQuery. [24]

Díky tomuto frameworku je možné upravovat vzhled a pozici elementů přidáváním příslušných řetězců do hodnoty HTML atributu `class`. V této práci je tento přístup použit především k úpravě vzhledu. Ke tvorbě složitějšího rozložení editoru je však využito zejména CSS, jelikož je v něm jednodušší vytvořit na míru komplexnější rozložení elementů.

Axios

Axios je JavaScriptová knihovna, která přímo nesouvisí s frameworkem Vue.js, ale je na frontendu aplikace v této práci hojně využívána. Obsahuje metody pro **posílání HTTP požadavků** a zpracovávání odpovědí na ně. Automaticky nastavuje hlavičky a celkově je díky ní komunikace přes HTTP jednodušší, než pomocí JavaScriptové knihovní funkce `fetch()`.

WebSocket

Pro **komunikaci v reálném čase mezi serverem a klientem** byl použit protokol WebSocket. Tento komunikační protokol, který poskytuje plně duplexní (tzn. obousměrný) komunikační kanál přes jedno jediné TCP spojení. To umožňuje komunikaci mezi klientem a serverem v reálném čase. Oproti tradičním HTTP požadavkům, které klient posílá a čeká, než server odpoví, umožňuje WebSocket kontinuální obousměrný proud dat. Jeho standard je definován dokumentem RFC 6455 a je nástupcem starších řešení jako například Comet nebo long polling, které využívají prodlužování a opakovaného navazování nového spojení. Například long polling funguje tak, že klient pošle na server HTTP požadavek, který neukončí spojení, dokud nemá nějaká data, která pošle v odpovědi. Jakmile klient obdrží odpověď, okamžitě odešle na server další požadavek. Tímto způsobem vzniká řetězec HTTP požadavků, které musí server řešit. Oproti tomu WebSocket pomocí HTTP požadavku s atributem `Upgrade: websocket` změní komunikační protokol na WebSocket. [16]

Navázání spojení od klienta pomocí upgrade požadavku vypadá například následovně:

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
```

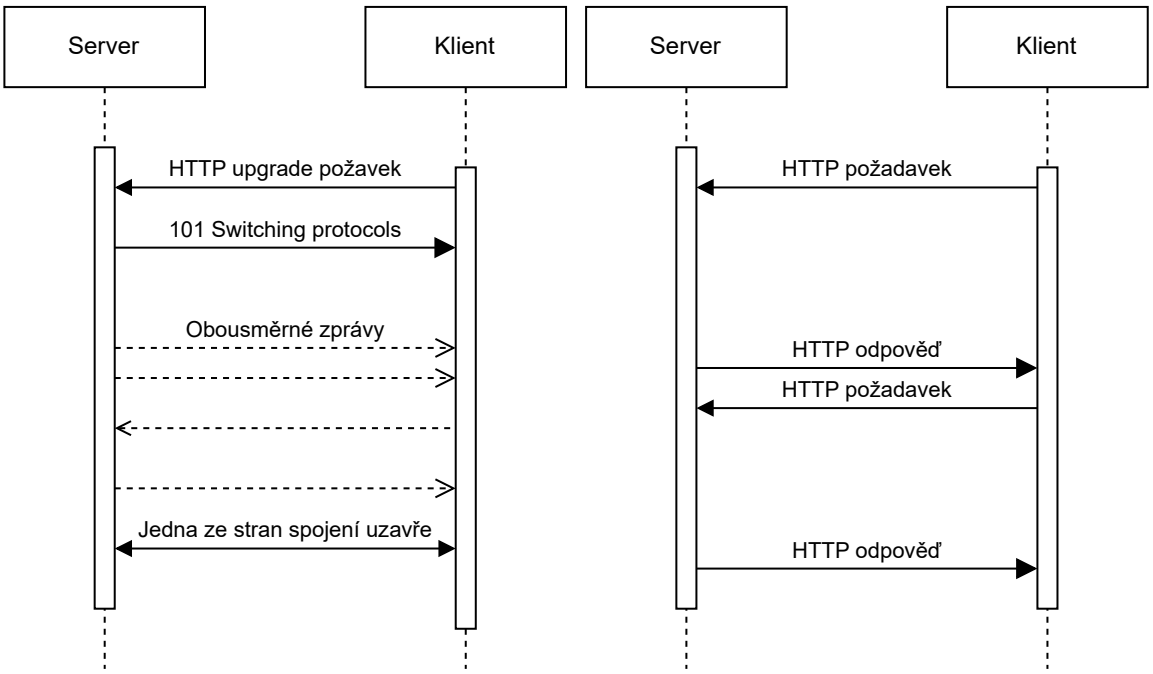
[9] https://getbootstrap.com/

```
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```

Server následně potvrdí navázání následovně:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

Tento příklad byl přejat z [18]. Jakmile je jednou spojení přes protokol WebSocket navázáno, klient a server si můžou vyměňovat data **bez** potřeby **posílání opakovaných požadavků** (viz obrázek 4.2). To zvyšuje rychlost komunikace v reálném čase, která je klíčová například pro chatovací, herní nebo kolaborativní aplikace. [16]



Obrázek 4.2: Příklady komunikace pomocí WebSocketu(vlevo) a long pollingu(vpravo)

Dříve byla velkým nedostatkem WebSocketu, v porovnání s long pollingem, podpora ve webových prohlížečích. Dnes už je podporován všemi nejpoužívanějšími, v důsledku čehož je použití WebSocketu pro aplikace pracující v reálném čase většinou vhodnější volbou. Také existují knihovny pro práci s WebSockety, například SockJS nebo Socket.IO. Ty u prohlížečů, které WebSocket z jakéhokoliv důvodu nepodporují, nahrazují nedostatek podpory tak, že mají implementováno záložní řešení, jako například výše popsaný long polling. [16]

Socket.IO

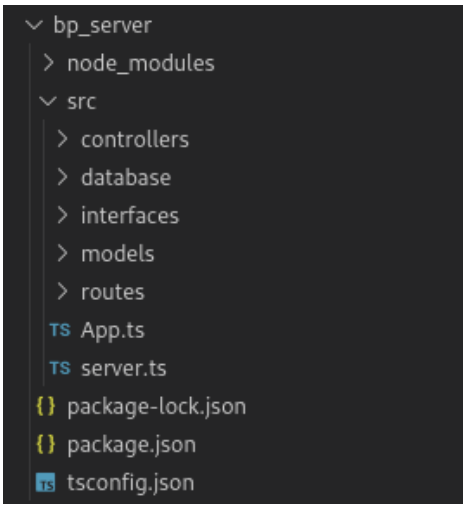
Pro implementaci byla zvolena JavaScriptová knihovna Socket.IO[10], která má podporu i pro TypeScript. Jejím použitím jsou řešeny nejen problémy s ne úplně stoprocentní podporou

[10]https://socket.io/

u prohlížečů, ale zároveň má i další výhody. Tou největší je vysokoúrovňové API, které umožňuje **jednoduchou práci s WebSockety**. Strukturuje posílaná data, aby byla přehledná a snadno zpracovatelná, řeší za uživatele automatické připojení v případě výpadku spojení a podporuje multiplexing nebo vytváření místností.

4.2 Express.js server

Je hned několik důvodů, proč udržovat přehlednou a intuitivní **adresářovou strukturu** při vývoji aplikace. Dobrá adresářová struktura umožňuje lepší přehlednost a snažší orientaci v množství souborů s kódem aplikace. Přehledně uspořádaný kód je jednodušší udržovat a rozšiřovat. Díky tomu, že jsou funkčně související soubory s kódem shlukovány k sobě, je mnohem jednodušší hledat v aplikaci chyby nebo přidávat novou funkcionalitu.



Obrázek 4.3: Adresářová struktura serverové části aplikace

Na obrázku 4.3 je struktura serveru aplikace pro kolaborativní tvorbu rozvrhů. Kořenový adresář obsahuje **konfigurační soubory** a dva adresáře. V prvním adresáři `node_modules` jsou veškeré závislosti, které jsou potřebné pro fungování serveru, například různé knihovny a jejich konfigurační soubory. Tyto závislosti byly do projektu nainstalovány pomocí nástroje pro správu balíčku NPM (Node Packet Manager). Každá závislost v tomto adresáři má vlastní složku pojmenovanou podle jména balíčku a obsahuje soubory, které jsou potřebné pro správnou funkčnost dané závislosti.

Druhým adresářem v kořenovém adresáři serveru je `src`. V tomto adresáři a jeho podsložkách se nacházejí veškeré zdrojové soubory aplikace. V `routes` je třída `Routes.ts`, která definuje veškeré koncové body serveru a přiřazuje jim metody řadičů z adresáře `controllers`, které budou zpracovávat požadavky příchozí na tyto koncové body. Všechny definice datových struktur a výčtových datových typů, které jsou používány napříč aplikací, se nachází v `interfaces`. Hlavní logika aplikace včetně přesunů lekcí a kontroly kolizí se nachází v adresáři `models`.

Mimo adresáře se nachází `App.ts` a `server.ts`. Třída `App.ts` je hlavní třídou, ve které se navazuje spojení s databází. Jsou volány metody pro nastavení cest, je zapnuto naslouchání Socket.IO zpráv a proběhne vložení implicitních dat do databáze, například hodnoty nastavení.

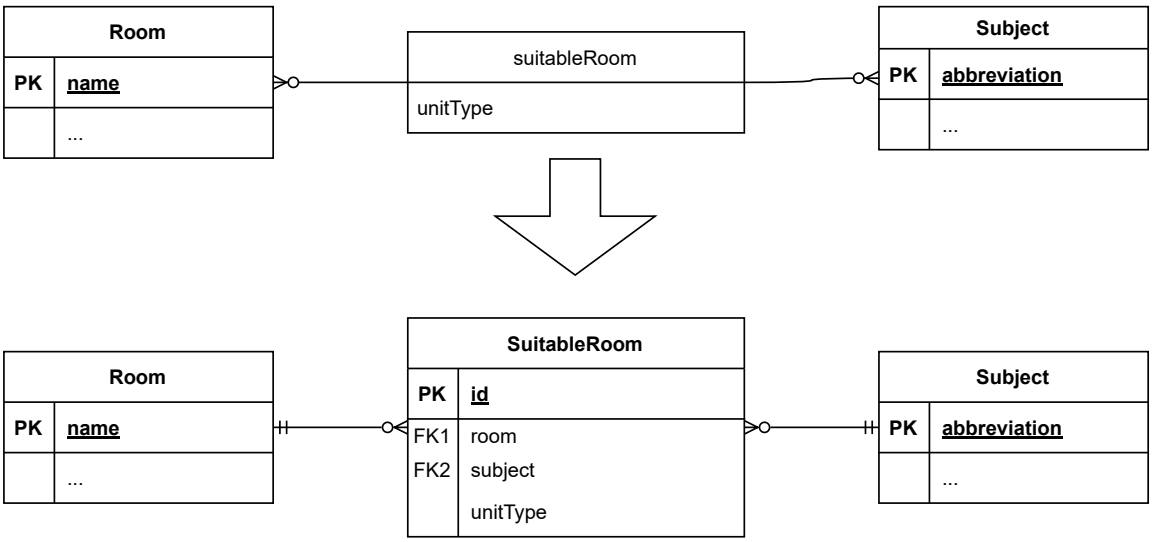
Vstupním bodem aplikace je skript **server.ts**, který spouští server a zapíná naslouchání aplikace na specifikovaném portu pro příchozí HTTP požadavky.

4.2.1 Tvorba databáze pomocí TypeORM

MySQL databáze je díky TypeORM implementována pomocí objektově orientovaného přístupu v TypeScriptu. Pro každou entitní množinu je vytvořena samostatná třída. Tyto třídy jsou potomky třídy `BaseEntity`, která je základní třídou pro tvorbu entit. Má v sobě již implementovánu funkcionalitu pro vytváření entit a vztahů mezi nimi. Též obsahuje metody pro práci s entitami, jako například `create()`, která slouží pro vytvoření nové instance entity, `save()` uloží instanci do databáze, `remove()` ji odstraní. Dále jsou v `BaseEntity` různé metody pro vyhledávání v databázi. Díky tomu, že každá třída, která definuje entitu v databázi dědí tyto metody, není potřeba je psát a testovat od základu.

Atributy entitních množin se implementují pomocí atributů třídy. K označení atributů jako sloupce tabulek se používají dekorátory. Běžný sloupec se vytváří s dekorátorem `@Column`. Pokud sloupec dekorujeme pomocí `@PrimaryGeneratedColumn("uuid")`, označíme sloupec jako primární klíč, který bude automaticky generovat identifikátory ve formátu UUID.

Dále se pomocí dekorátorů tvoří vazby. `@JoinColumn` se používá ke specifikaci toho, u které entitní množiny bude vazba uložena jako cizí klíč. Tyb vazby určují dekorátory `@ManyToOne`, `@OneToMany` apod.. Atribut vazby musí být uveden u obou entitních množin a díky tomu lze v jejich instancích přistupovat k vazbě z obou množin, ne jen z té, kde je uložena pomocí cizího klíče.



Obrázek 4.4: Převod vazby s atributem z ERD do TypeORM implementace

Pokud má však `M:N` vazba nějaký atribut, musí být v TypeORM implementována jako další entita. Na obrázku 4.4 je ukázáno, jak je implementována vazba `suitableRoom` s atributem `UnitType`.

Třídy pro veškeré TypeORM entity jsou v adresáři `database`. Nachází se zde také třída `InitialData`. Ta obsahuje statické metody pro kontrolu, zda jsou v databázi potřebná počáteční data. Pokud nejsou, pak jsou do databáze přidána s implicitními hodnotami.

Jedná se o entity třídy `Settings` a o místnost `unsorted`, na kterou se vážou všechny lekce, které nejsou rozvrhnuty do žádné z místností.

4.2.2 Zpracování HTTP požadavků

V adresáři `routes` je třída `Routes`, ve které jsou pomocí třídy `router` z frameworku `Express` definovány všechny koncové body serveru. Ke každému koncovému bodu je určena cesta, HTTP metoda, a metoda některého z řadičů, na kterou jsou delegovány požadavky příchozí na daný koncový bod. Jednotlivé definice koncových bodů jsou rozděleny do metod podle toho, k jakému typu dat se vztahují.

Řadiče z adresáře `controllers` jsou rozděleny do tříd podle typu dat, velmi podobně jako koncové body. Každá třída je implementována v jednom souboru a obsahuje metody, které zpracovávají HTTP požadavky.

V řadičích souvisejících přímo s entitami databáze jsou metody pro všechny CRUD operace. Pro čtení dat je metod zpravidla více, pro vytváření, úpravu a mazání je vždy jen jedna. Metoda pro přidání zkontroluje, zda je objekt v databázi již uložen, a vloží jej jen pokud není. Metoda pro úpravu změní objektu v databázi všechny atributy, které jsou uvedeny v těle příchozího požadavku.

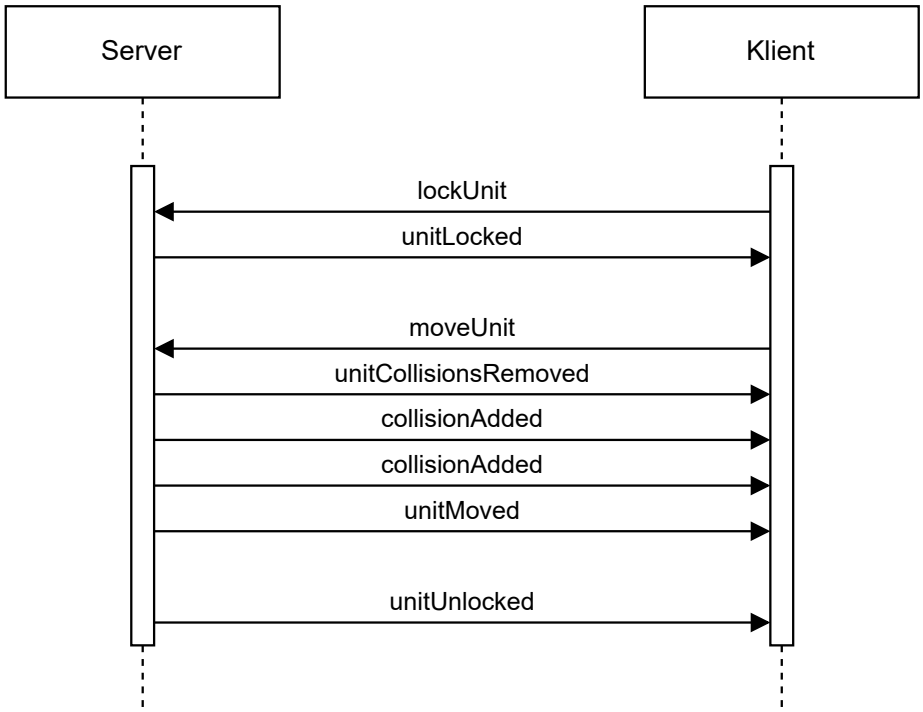
Metody pro mazání dat jsou pro různé entity jiné, jelikož jsou data na sobě různě závislá. Při mazání dat je někdy potřeba smazat nebo upravit navíc jiná data. Při špatném pořadí mazání dat může dojít k vazbám na již neexistující objekty v databázi a tím pádem k chybě. Zde jsou vypsány akce, které aplikace provádí v řadičích při procesu odstraňování dat.

- V případě mazání společných studentů se odstraní pouze kolize studentů, není potřeba odstraňovat nic dalšího. Stejný případ je při mazání preferované místnosti, kde není třeba nic jiného mazat ani upravovat.
- Pokud bude odstraněn přednášející, tak se odstraní jeho jméno ze všech lekcí, se kterými je asociován. U mazání lekcí je to podobné, je potřeba odstranit kolize související s odstraňovanou lekcí.
- Složitější to je u místností. Zde se všechny lekce, které jsou rozvrhovány do právě mazané místnosti, přesunou do implicitní místnosti `unsorted`, tudíž zpět do zásobníku lekcí. Navíc jsou smazány veškeré záznamy o vhodných místnostech, ve kterých je vhodnou místností právě smazaná místnost.
- Při odstraňování předmětů se smažou i všechny lekce, které předmět zaštiťuje. Též se smažou všechny preferované místnosti pro tento předmět.

4.2.3 Jak funguje přesun lekcí v rozvrhu na straně serveru

Na obrázku 4.5 je ukázka komunikace serveru s klientem pomocí SocketIO zpráv při přesouvání lekce v rozvrhu. Aby mohla být lekce přesunuta, musí být uzamčena uživatelem, který ji chce přesunout. V případech, kdy na základě charakteru akce je nutné uzamknout lekci dříve než bezprostředně před přesunem, je uzamčení požadováno pomocí SocketIO zprávy `lockUnit`. Při akci nahrazení lekce oznamované serveru zprávou `replaceUnit` je nahrazovaná lekce přemístěna okamžitě po uzamknutí. Proto není třeba ji uzamykat předem a uzamknutí je zahrnuto v průběhu zpracování zprávy `replaceUnit`.

Pokud se nepovede uzamknout všechny lekce potřebné k provedení akce, akce se neprovede. Zároveň jsou odemknuty všechny lekce uzamčené klientem, který se o akci pokusil.



Obrázek 4.5: Komunikace serveru s klientem pomocí SocketIO při přesunu lekce

Zamykání je organizováno třídou `EditorActionManager` a prováděno třídou `UnitLock`. Instance třídy `UnitLock` si drží záznamy o tom, které lekce jsou uzamčeny kterým klientem. V případě, že je požádáno o uzamčení lekce pomocí její metody `LockUnit()`, zkontroluje se, zda je v záznamech lekce již uzamčena. Pokud existuje záznam o uzamčení jiným klientem, je odmítnuto. Jinak je lekce uzamčena a všem klientům se odešla SocketIO zpráva `unitLocked`. Pro rozlišení klientů je používán identifikátor sezení klienta, který je automaticky vytvořen vždy při navázání spojení klienta se serverem. Při ukončení tohoto sezení jsou automaticky odemknuty všechny lekce, které jsou klientem uzamčeny.

Pokud je uzamčení lekcí pro přesun úspěšné, je proveden přesun. Po provedení přesunu se odstraní všechny kolize, které lekce na staré pozici měla a dojde ke kontrolám, které mohou najít kolize na nové pozici. Kolize jsou podrobněji rozebrány v sekci 4.2.4. Změny v kolizích jsou oznámeny všem klientům pomocí zpráv `unitCollisionsRemoved` a `collisionAdded`.

Po dokončení kontroly kolizí je klientům oznámeno zprávou `unitMoved`, že byla lekce přesunuta a zprávou `unitUnlocked`, že byla po přesunutí lekce odemknuta.

4.2.4 Algoritmus pro detekci kolizí v rozvrhu

Kolize se dají rozdělit na dvě skupiny. V první nastává konflikt dvou lekcí v případě, že se dvě lekce konají ve stejný čas. V této práci se setkáváme s následujícími:

- Konflikt přednášejícího: Dvě lekce ve stejný čas nemůže přednášet jeden přednášející.
- Konflikt místa: Dvě lekce nemůžou být ve stejnou dobu na stejném místě.
- Konflikt povinných lekcí: Dvě lekce s povinnou účastí nemůžou být ve stejnou dobu, pokud mají průnik studentů.

- Velký počet společných studentů: Lekce, které se konají ve stejný čas mají příliš vysoké procento studentů.

Ve druhé skupině dochází ke konfliktu lekce s jinou skutečností. Do druhé skupiny patří:

- Konflikt s rezervací: Místnost, ve které se lekce nachází, je v čas lekce rezervována pro něco jiného.
- Konflikt s preferencemi přednášejícího: V čas lekce přednášející nemůže učit, nebo to nepreferuje.
- Nevhodná místnost pro výuku: Přiřazená místnost není vhodná pro výuku této lekce.
- Nedostatečná kapacita učebny: Učebna nemá dostatečnou virtuální kapacitu pro konání lekce, případně reálnou kapacitu u povinných lekcí.

Způsob uložení

Kolize jsou během chodu serveru uloženy v atributu instance třídy `Collisions`. Tento atribut je pole typu `ICollision[]`. Interface `IColision`, pomocí kterého se ukládají informace o kolizi má čtyři atributy.

- `unitAId:string` a `unitBId:string` identifikují, kterých dvou lekcí se kolize týká. Pokud je to typ kolize, ve které figuruje pouze jedna lekce (tzn. kolize je ze druhé skupiny), pak oba atributy nesou hodnotu identifikátoru dané lekce.
- `CollisionType` je výčtový typ, který nabývá hodnot 0 až 8. Každá z nich určuje jeden ze sedmi typů konfliktu. Pro velký počet společných studentů existují hodnoty dvě, jelikož zde kolize není symetrická. Tento atribut slouží ke kontrole duplicit v metodě `addCollision()`.
- Posledním atributem je `explanation:string`, ve kterém je uložen řetězec s vysvětlením toho, proč daná kolize nastala.

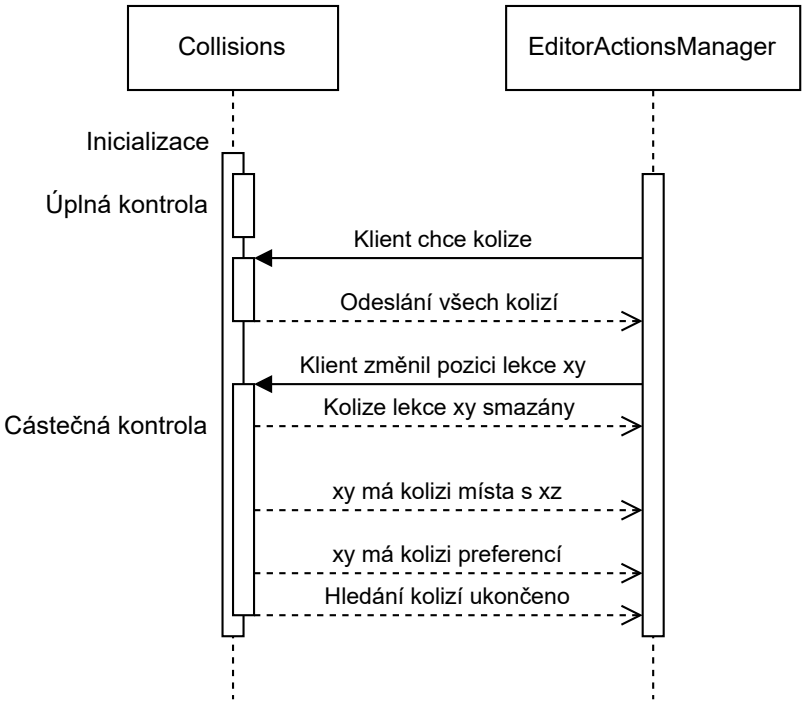
Ukládání kolizí není perzistentní, ale při spuštění serveru se dopočítávají všechny kolize na základě dat uložených v databázi. Klient tyto kolize získá pomocí HTTP požadavku a uloží si je do `CollisionStore.ts`, ve kterém jsou průběžně aktualizovány na základě příchozích Socket.IO zpráv.

Průběh kontrol

Existuje úplná a částečná kontrola. Úplná kontrola proběhne hned při spuštění serveru a poté vždy, když dojde ke změně vstupních dat.

Částečná kontrola proběhne vždy po přemístění lekce v editoru. Na obrázku 4.6 je zobrazen diagram, jak probíhá komunikace zejména při částečné kontrole mezi třídami Collisions a EditorActionManager. Spouští se funkcí `refreshUnitCollisions()`. V ní se nejdříve odstraní z pole kolizí všechny kolize, ve kterých figuruje upravovaná lekce pomocí metody `removeUnitCollisions()`. Po odstranění těchto kolizí se odešle klientovi Socket.IO zpráva `'unitCollisionsRemoved'`, která oznamuje odstranění kolizí dané lekce.

Pak je spuštěna `findUnitCollisions()`, ve které jsou nejdříve vyhledány z databáze všechny ostatní lekce, které jsou zařazeny do stejného dne jako lekce, jejíž kolize hledáme. Potom je pole vyhledaných lekcí postupně procházeno a lekce z něj jsou porovnávány s lekcí,



Obrázek 4.6: Příklad komunikace s instancí třídy Collisions

jejíž kolize hledáme. Nejprve je kontrolováno, zda se lekce konají současně alespoň v jednom týdnu v semestru a zda jejich časy konání mají neprázdný průnik.

Pokud je první kontrola vyhodnocena pozitivně, tak to značí, že se lekce konají alespoň v jednom týdnu ve stejný den i čas. To samotné na kolizi nestačí, je potřeba, aby se potvrdila ještě některá z následujících kontrol pomocí jedné ze tří funkcí: `findPlaceCollision()`, `findLecturersCollisions()` nebo `findSharedStudentsCollisions()`.

- Nejjednodušší kontorolou je `findPlaceCollision()`, která porovná místnosti, do kterých jsou lekce přirazeny. Pokud jsou shodné, je přidána kolize.
- V `findLecturersCollisions()` je spočítán průnik učitelů u obou lekcí. Pokud je průnik neprázdný, je přidána kolize.
- Další kontroly probíhají ve funkci `findSharedStudentsCollisions()`. Jsou vypočítány hodnoty, kolik procent studentů z lekce A navštěvuje lekci B a obráceně. Poté je z nastavení získána hodnota, kolik maximálně studentů může navštěvovat současně konané lekce. Pokud je tato hranice překročena, je přidána kolize. U lekcí, které jsou obě označené jako povinné, stačí pro přidání kolize jediný společný student.

Nakonec se spustí kontrola druhé skupiny kolizí.

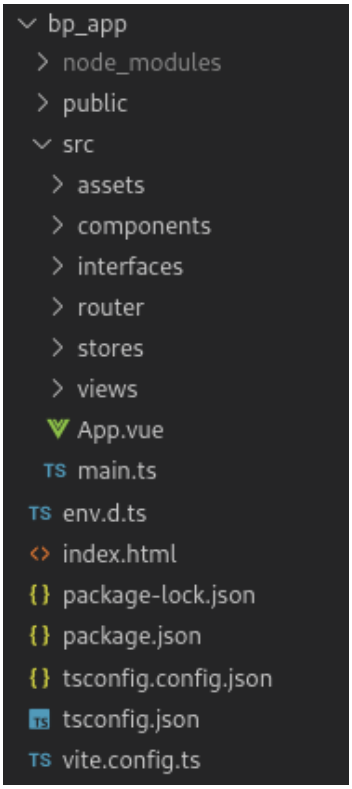
- Metoda `findSuitableRoomsCollision()` najde v databázi všechny preferované místnosti pro výuku dané lekce. Pokud nejsou k dané lekci uvedeny žádné preferované místnosti, jsou pro ni vhodné všechny. V případě, že k lekci jsou uvedeny preferované místnosti a místnost, ve které je momentálně lekce zařazena mezi nimi není, je přidána kolize.

- `findCollisionsWithLecturersPreferences()` vyhledá pro každého přednášejícího hodinu s nejhorší preferencí z hodin, do kterých lekce zasahuje. Pokud je preference stejná nebo nižší než hraniční hodnota nastavená v databázi, pak je přidána kolize.
- `findCollisionsWithReservations()` projde rezervace všech hodin, do kterých lekce zasahuje a pokud je některá z nich rezervovaná, je přidána kolize.
- `findCapacityCollisions()` v závislosti na tom, zda je lekce povinná či ne, porovnává počet studentů v lekci s reálnou kapacitou nebo s virtuální kapacitou místnosti. Pokud je kapacita nižší než počet studentů, vzniká kolize.

Kolize jsou ukládány pomocí metody `addCollision()`. Před voláním se vždy vytvoří hláška `explanation` s vysvětlením kolize, která se do metody `addCollision()` předá jako jeden z parametrů. v samotné metodě se nejdříve kontroluje, jestli mezi těmito lekcemi již není uložena kolize stejného typu jako ta, kterou chceme přidat. To proto, že při současné implementaci úplné kontroly se každá dvojice lekcí kontroluje 2x - A s B a B s A.

Úplná kontrola začíná v metodě `findAllCollisions()`, která spustí paralelně metodu `findAllCollisionsInDay()` pro všech 7 dnů v týdnu. Metoda `findAllCollisionsInDay()` vytáhne z databáze všechny jednotky, které jsou v rozvrhu přiřazeny do časů v rámci daného dne. Potom pro každou z těchto lekcí spustí metodu `findUnitCollisions()`, která je součástí průběžné kontroly a je popsána v odstavcích výše.

4.3 Vue.js klient



Obrázek 4.7: Adresářová struktura klientské části aplikace

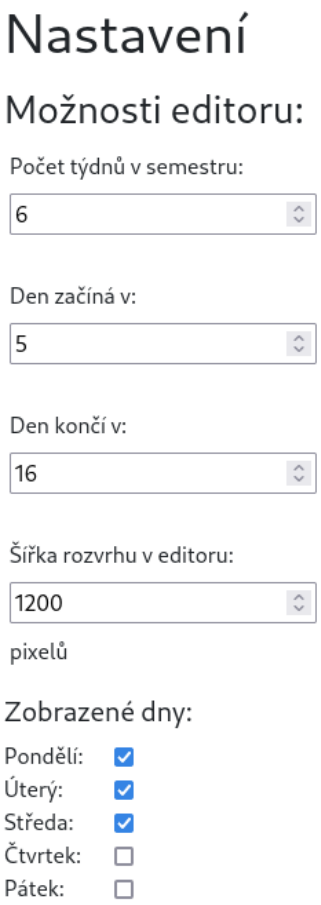
Stejně jako u serverové části aplikace, i klientská část je logicky členěna do adresářů viz obrázek 4.7. Kořenový adresář obsahuje:

- Adresář `node_modules`, ve kterém jsou veškeré závislosti, které jsou potřebné pro fungování klienta, podobně jako u serverové části.
- Adresář `public`, ve kterém je uložen jediný obrázek a to ikona aplikace.
- Adresář `src` obsahující zdrojové soubory aplikace. Je dále členěn na:
  - `main.ts` je skript, ve kterém je instanciována Vue.js aplikace a jsou registrovány knihovny `Pinia` a `Vue router`,
  - `App.vue` představuje hlavní Vue komponentu aplikace,
  - `assets` obsahuje globální soubor `main.css`, který má vliv na komponenty napříč aplikací,
  - `components` má v sobě podadresáře, ve kterých jsou roztřízeny všechny Vue komponenty podle toho, v jaké části aplikace se vyskytují,
  - `interfaces` obsahuje soubory s definicemi výčtových datových typů a rozhraní datových struktur,
  - `router` disponuje souborem `index.ts`, ve kterém jsou definice všech cest a na každou z nich mapuje pohled, který je vykreslen při vstupu na ni,
  - `stores` nese soubory s jednotlivými definicemi úložišť Pinia
  - `views` obsahuje všechny základní komponenty pohledů, které jsou přiřazeny jednotlivým cestám aplikace.
- Soubor `index.html`, který je vstupním bodem klientské části aplikace. Je v něm nastavena ikona, název aplikace a spouští se zde skript `main.ts`.
- Konfigurační soubory

4.3.1 Nastavení

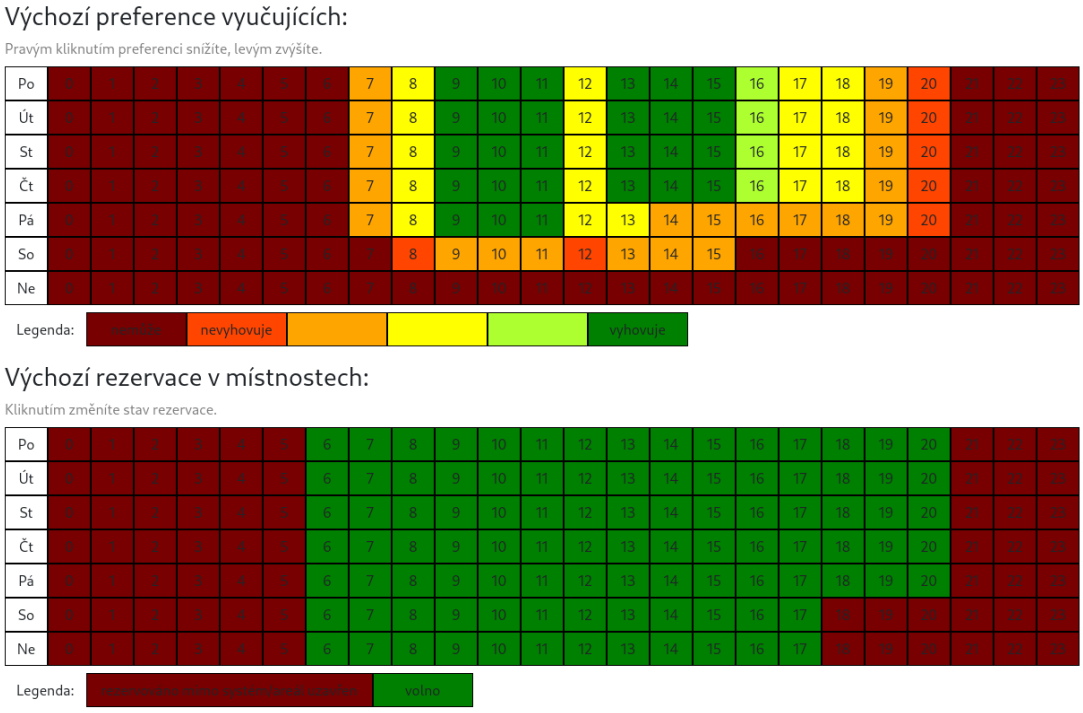
Na obrázcích 4.8 a 4.9 jsou zobrazeny ukázky stránky s nastavením. Zde lze nastavit různé hodnoty, které jsou členěny do kategorií. Tyto hodnoty jsou ukládány na server akcemi v `SettingsStore.ts`. Nejsou však synchronizovány v reálném čase, aby změny nastavení nerušily práci ostatních uživatelů. U uživatele, který nastavení upravil, se změny projeví ihned. U ostatních až při obnovení stránky. Pokud je zadána nevyhovující hodnota, je na to upozorněno okamžitě červeným rámečkem a textem s vysvětlením. V rámci nastavení je možné konfigurovat následující parametry:

- **Počet týdnů v semestru** ovlivňuje počet týdnů, které se zobrazí v editoru při týdenním režimu zobrazení.
- **Začátek a konec dne** definují časový interval, který bude zobrazen v editoru.
- **Šířka rozvrhu** v editoru udává šířku rozvrhů při týdenním režimu zobrazení v editoru.
- **Zobrazené dny** umožňují výběr dnů, které jsou v editoru zobrazeny.



Obrázek 4.8: V nastavení lze měnit konfiguraci editoru

- **Výchozí preference vyučujících** udávají výchozí vzor, který je zobrazen jako preference přednášejících při jejich tvorbě pomocí formuláře.
- **Výchozí rezervace místností** definují základní rezervované časy u nově vytvářených místností.
- **Virtuální kapacitu** vyjadřuje poměr virtuální kapacity a reálné kapacity dané místnosti, přičemž jako virtuální kapacita je zde uvedená procentuální hodnota reálné kapacity.
- Předposlední pole udává maximální **procentuální počet studentů**, kteří mohou mít dvě lekce rozvrhnuté na stejný čas, aniž by vznikla kolize.
- V posledním poli se nastavuje **krajní hodnota pro časové preference přednášejících**. Při této hodnotě je preference považována za příliš nízkou a je přidána kolize.



Obrázek 4.9: V nastavení lze měnit konfiguraci přednastavených preference přednášejících a výchozí rezervace místností

4.3.2 Zobrazování lekcí

Pro zobrazování a manipulaci s lekcemi slouží Pinia store `TimetableStore.ts`. Do něj se při připojení komponenty editoru `EditorMain.vue` pomocí HTTP GET požadavku nahrají všechna potřebná data. Tato data mají formu `IEditorTimetable[]`, tzn. pole rozvrhů všech místností. Datová struktura `IEditorTimetable` obsahuje místnost `IRoom`, které rozvrh náleží a pole `IUnit[]`, ve kterém jsou všechny lekce, které jsou zařazeny do této místnosti. Lekce, které nejsou do žádné z místností ještě zařazeny, se načtou do rozvrhu v místnosti `"unsorted"`, která jim byla implicitně přiřazena již při jejich vytvoření.

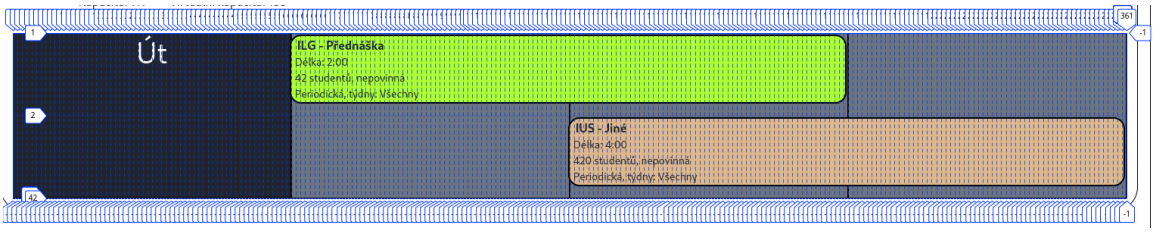
Tyto místnosti, které nejsou ještě nikam zařazeny, jsou zobrazeny jako kartičky vpravo v **zásobníku lekcí** `EditorStack.vue`. Ten nezařazené lekce nezískává pomocí HTTP požadavku, ale pomocí getteru `getUnitsByRoomName("unsorted")` z timetableStore. V zásobníku jsou kartičky zobrazeny pomocí CSS vlastnosti `display: flex`.

Již přiřazené lekce se zobrazují do rozvrhu v levé části pracovní plochy. Jsou dva režimy zobrazení pracovní plochy: celosemestrální a týdenní.

Celosemestrální režim zobrazuje pro každou místnost jeden rozvrh, kdežto **týdenní** režim zobrazuje vedle sebe rozvrh pro každý týden semestru. Tím pádem v případě, že má semestr 13 týdnů, je pro každou místnost zobrazených vedle sebe 13 rozvrhů. To usnadňuje zobrazování sudé nebo jinak méně pravidelné lekce do rozvrhu a je jedinou možností, jak vkládat do rozvrhu blokové lekce.

V celosemestrálním režimu je zobrazený pro každou místnost klasický rozvrh hodin. V záložce nastavení se tyto rozvrhy dají upravovat na míru, viz podkapitola 4.3.1. Jejich implementace je v souboru `EditorTimetable.vue`. Rozvrh má tři části. Informace o místnosti, které náleží (název, kapacitu...), časovou osu a jednotlivé komponenty dnů rozvrhu.

Každý den rozvrhu je zobrazován pomocí CSS gridu - sady funkcí, která umožňuje zobrazit rozvržení elementů do dvoudimenzionální mřížky. Tato mřížka má pro každou buňku rozvrhu 60 sloupců a proměnlivý počet řádků tak, aby se každá lekce v buňce zobrazila na svůj řádek. Počet sloupců 60 je z toho důvodu, že hodina má 60 minut a díky tomu jsme schopni zobrazovat lekce s rozlišením na jednotlivé minuty. Ukázka je na obrázku 4.10.



Obrázek 4.10: Ukázka CSS mřížky pro den rozvrhu

Hlavička s označením dne a jednotlivé kolonky pro hodiny mají fixní šířku 60 sloupců proto, aby vytvořily pravidelnou mřížku po hodinových intervalech. Počet řádků, které v mřížce toto pozadí zabírá je libovolná relativně k tomuto účelu vysoká konstanta - například 42. Dokud jsou řádky bez obsahu, tak nejsou viditelné. Pokud v konkrétním dnu není zobrazena žádná lekce, jediný obsah je hlavička dne, díky které zůstane jinak prázdné pozadí viditelné. Pokud jsou lekce vloženy do rozvrhu, výška pozadí se tak automaticky přizpůsobí podle počtu řádků, na kolika jsou lekce zobrazeny.

Lekce jsou z `TimetableStore.ts` získávány pomocí getteru `getUnitsForTimetable()`, který na základě parametrů den, týden a místnost vrátí všechny lekce, které mají být zobrazeny v tomto dnu. Než jsou však tyto lekce zobrazeny jen nutné vypočítat, do jakého řádku se v mřížce zobrazí, aby nedocházelo k překryvu. K tomu slouží datová struktura `IUnitDisplayInfo` a metoda `calculateUnitsRow()`. Tato metoda každou jednotku přetvoří ze struktury `IUnit` na `IUnitDisplayInfo` ve dvou krocích. Jeden z atributů `IUnitDisplayInfo` je `unit`, do kterého se uloží původní jednotka. Poté se v prvním kroku do atributu `numberOfCollisions` spočítá, kolik překryvů by jednotka měla, kdyby se zobrazily všechny jednotky do jednoho řádku mřížky. Tímto se vytvoří důležité kritérium, které ve druhém kroku metoda rozřazuje do řádků.

Rozřazování probíhá iteračně přiřazováním čísel řádků do atributu `row:`. Na začátku mají všechny lekce tento atribut na hodnotě 1. Při rozřazování se procházejí všechny lekce, které se v daném dnu chystáme zobrazit. Pokud narazíme na dvě, které mají časový překryv a navíc i stejný řádek, tak řádek jedné z nich inkrementujeme na základě následujících kritérií:

- Lekci, která **má více překryvů**, inkrementujeme řádek. To má ve výsledku takový efekt, že se překrývající se lekce zobrazují do obrácených pyramid, tudíž je víc lekcí nahoře a minimum dole. To výrazně přispívá ke kompaktní vizáži rozvrhu. Pokud mají obě lekce stejný počet překryvů, pak:
- Lekci **vyučované v méně týdnech semestru** se inkrementuje řádek. Toto kritérium vychází z toho, že se nejdříve rozvrhují periodické lekce, které probíhají každý týden a poté se doplňují blokovými tam, kde zbylo místo. Toto kritérium zamezí neustálému překreslování již ustálených lekcí, a bude se při práci v rozvrhu lépe orientovat.
- Pokud ani jedno ze dvou výše uvedených pravidel nedokáže určit, která lekce se zobrazí níže, pak je posunuta níže druhá z nich.

Kapitola 5

Testování

Testováním se kontroluje, zda aplikace neobsahuje žádné chyby a splňuje požadované vlastnosti. Testování může probíhat ve všech částech vývojového cyklu softwaru a lze je provádět pomocí automatizovaných nebo manuálních metod. [11]

5.1 Manuální testování

V této sekci je čerpáno z [21].
Pro testování aplikace bylo vybráno **manuální testování**. Manuální testování je vhodné zvolit v situacích, kdy je potřeba otestovat velké množství jednoduchých komponent, pro které by bylo časově velmi náročné vytvářet spousty testovacích případů při tvorbě automatizovaných testů. Je vhodné pro testování uživatelských rozhraní, jejichž kvalita je obtížně popsatelná kvantitativními metrikami. Výhody manuálního uživatelského testování:

- Během manuálního testování jsou získány osobní názory a vhled do aplikace založené na zkušenostech uživatelů, které mohou být díky jejich kreativitě různorodé a dokáží pokrýt velmi široké spektrum testovacích scénářů, včetně těch, na které vývojáři vůbec nemysleli.
- Jsou zjištěny informace ohledně použitelnosti, grafického zpracování a intuitivity aplikace, které je téměř nemožné získat automatizovanými testy.
- Uživatelé jsou schopni navrhnout změny vedoucí k nápravě nedostatků.
- Nemusí být psán žádný kód navíc.

Manuální testování má i řadu nevýhod:

- Testovací scénáře může být náročné přesně replikovat.
- Dovednosti a zkušenosti testerů mohou být na různé úrovni. Pokud nejsou znalí problematiky, výsledky testování mohou být nedostatečné nebo zavádějící. Správně složený tým testerů však může být velkou výhodou.
- Manuální testování trvá dlouho. Lidé nejsou rychlí jako počítač, a provedení testovacího scénáře zabere mnohem delší čas člověku než počítači.
- Opakované provádění testování v průběhu vývojového cyklu softwaru může být drahé, je nutné tým testerů platit za každé opakování testu. Oproti tomu automatizované testy je nutné platit pouze jednou při jejich programování.

- Testeři nejsou neomylní, a můžou dělat různé chyby.

5.2 Průběh testování

Testování se zúčastnilo 16 uživatelů, kteří by se dali rozdělit do dvou skupin.

- Studenti informačních technologií a programátoři z praxe, kteří měli zpravidla odbornější připomínky.
- Uživatelé mimo obor informačních technologií, jejichž zpětná vazba směřovala zejména k použité terminologii.

Navíc každá skupina měla svůj, ale převážně jednotný názor na srozumitelnost jednotlivých prvků aplikace. Přestože počet uživatelů, kteří aplikaci testovali, nebyl vysoký, výsledky v rámci skupin byly konzistentní. **Rozdílnost** skupin se ukázala být při tomto druhu testování klíčová, neboť díky tomu byly poznatky různorodého charakteru.

Testování probíhalo tak, že bylo uživatelům velmi stručně, asi ve dvou větách řečeno, k čemu přibližně aplikace slouží. Uživatelé tedy neměli **žádnou předchozí znalost prostředí** nebo řešené problematiky. Byl jim předán vytištěný **testovací scénář** přiložený v příloze A. Splnění tohoto testovacího scénáře a následné vyplnění zpětné vazby trvalo uživatelům přibližně 15–20 minut.

5.3 Výsledky testování

Uživatelům mimo prostředí informačních technologií (IT) zabralo delší čas se v aplikaci orientovat, uživatelům zdatným v IT zabraly více času snahy najít nějakou chybu v aplikaci. Poznatky ze zpětné vazby jsou následující:

1. Všem uživatelům se podařilo bez problémů splnit scénář daný testovacím protokolem.
2. Pro většinu uživatelů byly názvy a tlačítka intuitivní a srozumitelné.
    - Uživatelka z oboru IT uvedla, že by jí přišlo vhodné místo domovského tlačítka mít zvýrazněné vždy to tlačítko, ve které záložce se uživatel aplikace nachází.
    - Uživatelé mimo IT neuvedli žádné problémy.
3. Co se týče ovládání aplikace:
    - Třem uživatelům mimo IT připadala aplikace na první pohled složitá, ale nakonec uvedli, že bylo ovládání jednoduché.
    - Jednomu se špatně zadávaly časové preference vyučujících, kdy kvůli barvosleposti splývaly tři barvy do jedné. Dvěma vadil element pro vkládání JSON souboru roztažený přes celou šířku obrazovky.
    - Šesti uživateli byl kritizován formulář pro výběr preferencí přednášejícího. Koncept samotný jim přišel v pořádku, ale postrádali pokročilejší funkce, které by zadávání časových přednášejících zrychlily.
    - Dvěma uživatelům scházela tlačítka pro přidání dat přímo pod tabulkami s jejich přehledem.
4. Chyby aplikace se projevily pouze u jednoho ze dvou uživatelů.

- Ve formuláři pro úpravu lekce nešlo potvrdit formulář se správně zadanými týdny (a vypisovalo to, že zadány správně nejsou).
- Pod formulářem s úpravou přednášejícího chybělo tlačítko pro smazání a tlačítko pro ukládání úprav bylo špatně zbarveno.

5. Uživatelé z obou skupin taky uvedli některé věci, které se jím líbily.
   - Dvakrát zaznělo přidávání vyučujících do lekcí,
   - třikrát systém rozvrhování lekcí pomocí drag and drop akcí,
   - jednomu vlastnost vkládání dat, kdy se ze souboru JSON přidají jen ta data, která v systému ještě nejsou.

Zapracování výsledků testování do aplikace

Na základě výsledků testování bylo v aplikaci upraveno:

- Zadávání týdnů ve formuláři pro úpravu lekce bylo opraveno.
- Byla upravena barva tlačítka pro ukládání a přidáno tlačítko pro smazání přednášejícího ve formuláři pro úpravu přednášejícího.
- Byly přidány tlačítka pod tabulky s daty odkazující přímo na formuláře pro přidání daného typu dat.
- Element pro vkládání JSON souboru byl zmenšen.
- V horní nabídce je zvýrazněno vždy tlačítko aktuální záložky, nikoliv domovské.

Úprava formuláře pro zadávání priorit přednášejících a zavedení režimu, který zjednoduší práci v aplikaci barvoslepým by vyžadovaly další průzkum možných řešení a jejich implementace by byla náročnější, proto nebyly do aplikace implementovány.

5.4 Možná rozšíření do budoucna

Během procesu návrhu a vývoje aplikace vzniklo několik poznatků, které by bylo užitečné zakomponovat do konceptu nebo do finální aplikace při implementaci firmou. V tomto seznamu nejsou uvedeny všechny funkcionality, které je užitečné v aplikaci mít, například automatické generování rozvrhů nebo napojení vstupních a výstupních dat na informační systém univerzity. To proto, že tyto funkce nesouvisí plně s těmi koncepty, které má za úkol tato aplikace demonstrovat.

Po implementaci prvních dvou bodů z následujícího seznamu vhodných rozšíření by aplikace byla na úrovni MVP[1].

- Dalším vhodným rozšířením by byla možnost mít v aplikaci uložených více projektů současně, bez nutnosti ukládat rozpracovaný projekt do JSON souboru. Díky tomu by bylo možné pracovat na více projektech současně, nebo mít jednoduše k dispozici více pracovních verzí jednoho rozvrhu.

[1]Minimal Viable Product, v překladu minimální životaschopný produkt je funkčně minimální verze aplikace, která má smysl být nasazena na produkci a zpřístupněna prvním uživatelům. [14]

- Autorizační systém by umožnil, že v aplikaci může mít každý uživatel své hodnoty nastavení nezávisle na ostatních uživatelích. Navíc v kombinaci s předchozí odrážkou o možnosti mít uložených více rozvrhů v systému, by mohly v aplikaci existovat privátní projekty. Současně by mohlo aplikaci používat více institucí, protože by k daným projektům měl přístup pouze uživatel, kterému je udělen přístup.
- V aplikaci by mohlo být více základních variant pro zadávání preferencí přednášejících a rezervací místností. Mohlo by existovat několik základních variant, a další by mohly být vytvořeny a uloženy uživateli.
- Podobným způsobem by mohlo být umožněno uživatelům vytvářet nové druhy lekcí, jelikož může existovat instituce, které stávající nabídka druhů lekcí nebude dostačovat.

Aplikace je navržena tak, aby byla jednoduše rozšiřitelná. Pro implementaci těchto změn by v budoucnu nebylo nutné stavbu a principy fungování současné aplikace nijak výrazně měnit.

Kapitola 6

Závěr

Výsledkem této práce je aplikace pro kolaborativní tvorbu rozvrhů, která by měla sloužit jako důkaz konceptu pro potenciální budoucí produkt firmy IS4U.

V první části byla studována řešení aplikací, které již existují. V práci je detailně rozebrán produkt RoGeR od firmy IS4U, v jejíž spolupráci byla tato práce vytvořena. Současné řešení vytvořené touto firmou má mnoho nedostatků, a proto byl v této práci zpracován koncept aplikace, který řeší některé z nich a přidává možnost spolupráce v reálném čase.

Byly analyzovány výhody a nevýhody staré aplikace. S konzultantem firmy IS4U byly dohodnuty parametry konceptu nové aplikace, na základě kterých byl vytvořen návrh pomocí ER diagramu a wireframů. Během dalších konzultací byl návrh upraven. Následně byly vybrány a popsány technologie vhodné pro tuto práci a došlo k implementaci webové aplikace. Použita byla MySQL databáze, implementována pomocí knihovny pro objektově relační mapování TypeORM. Serverová i klientská strana aplikace byly napsány v jazyce TypeScript, na serverové straně s využitím běhového prostředí Node.js a frameworku Express.js. Klient byl implementován pomocí frameworku Vue.js. Komunikace mezi klientem je řešena mimo klasických HTTP požadavků i pomocí protokolu WebSocket, který umožnil uživatelům spolupráci v reálném čase. Pro implementaci komunikace skrz protokol WebSocket byla použita knihovna Socket.IO.

Nakonec byla webová aplikace pro kolaborativní tvorbu rozvrhů testována uživateli a jejich zpětná vazba byla následně zpracována. Ukázalo se, že je důležité mít testery s rozdílnými úrovněmi zkušeností, neboť poté jejich provedení testovacího scénáře a následná zpětná vazba bývají různorodé a vzájemně se doplňují.

Nejdůležitějšími vylepšeními oproti současné aplikaci jsou:

- Přechod z desktopové na **webovou aplikaci**,
- podpora práce s **blokovými** a nepravidelně se konajícími lekcemi,
- možnost **spolupráce** V editoru s více uživateli **v reálném čase**.

V průběhu vypracování této práce došlo k ukončení spolupráce mezi fakultou a firmou IS4U. V důsledku čehož přišla práce v pozdější fázi vývoje o cenné konzultace a o finální zpětnou vazbu. I přesto má však tato práce potenciál pro další využití. Může být v budoucnu inspirací pro jakoukoliv jinou firmu či instituci, která vyvíjí software pro řešení problematiky tvorby rozvrhů.

I přes to se v práci podařilo implementovat vše, co firma IS4U požadovala. Existují však další možnosti, jak tento koncept ještě dále rozšířit. Přidání autorizačního systému a mož-

nost existence více projektů současně v aplikaci by mohla aplikaci připravit na nasazení nezávisle na firmě IS4U.

Literatura

[1] Awad, B. *Should you use Sequelize, TypeORM, or Prisma 1?* [online]. May 2019 [cit. 2023-05-02]. Dostupné z: https://www.youtube.com/watch?v=mpbWQbkl8_g#t=20m15s.

[2] Awati, R. *What is object-relational mapping (ORM)? – TechTarget definition* [online]. TheServerSide.com, Mar 2023 [cit. 2023-05-02]. Dostupné z: https://www.theserverside.com/definition/object-relational-mapping-ORM.

[3] Bawa, A. *Types of client server communication* [online]. OpenGenus IQ: Computing Expertise amp; Legacy, Dec 2021 [cit. 2023-05-06]. Dostupné z: https://iq.opengenus.org/types-of-client-server-communication/.

[4] contributors, M. *Express/node introduction - learn web development: MDN* [online]. MozDevNet, Feb 2023 [cit. 2023-05-06]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction.

[5] Davidson, T. *Node.js vs next.js: Runtime environment VS framework* [online]. Clean Commit, Feb 2023 [cit. 2023-05-06]. Dostupné z: https://cleancommit.io/blog/node-js-vs-next-js-runtime-environment-vs-framework/.

[6] Day Richter, J. *What's different about the new google docs: Working together, even apart* [online]. Google, Sep 2010 [cit. 2023-05-04]. Dostupné z: https://drive.googleblog.com/2010/09/whats-different-about-new-google-docs_21.html.

[7] Dybka, P. *Crow's foot notation* [online]. Vertabelo, Apr 2016 [cit. 2023-05-04]. Dostupné z: https://vertabelo.com/blog/crow-s-foot-notation/.

[8] Education, I. C. *MySQL vs. mongodb: What's the difference?* [online]. IBM Cloud Education, Nov 2021 [cit. 2023-05-02]. Dostupné z: https://www.ibm.com/cloud/blog/mysql-vs-mongodb.

[9] Gala, M. *The Enigma of collaborative editing* [online]. Medium, Sep 2019 [cit. 2023-05-04]. Dostupné z: https://medium.com/@mehulgala77/concurrent-collaborative-editing-d10192e55d2e.

[10] García Gallardo, E. *What is MVVM architecture?* [online]. Built In, Jan 2023 [cit. 2023-05-06]. Dostupné z: https://builtin.com/software-engineering-perspectives/mvvm-architecture.

[11] Hamilton, T. *What is software testing? definition* [online]. 2023 [cit. 2023-05-07]. Dostupné z: https://www.guru99.com/software-testing-introduction-importance.html.

[12] IBM. *Optimistic and pessimistic record locking* [online]. IBM Corporation, Mar 2021 [cit. 2023-05-05]. Dostupné z: https://www.ibm.com/docs/en/rational-clearquest/7.1.0?topic=clearquest-optimistic-pessimistic-record-locking.

[13] JetBrains. *Databases survey 2022* [online]. JetBrains, 2022 [cit. 2023-05-02]. Dostupné z: https://www.jetbrains.com/lp/devecosystem-2022/databases/.

[14] Koďousková, B. *Co JE MVP a proč Ho Použít K tvorbě digitálního produktu* [online]. Rascasone, Jul 2021 [cit. 2023-05-02]. Dostupné z: https://www.rascasone.com/cs/blog/mvp-digitalni-projekt-aplikace.

[15] Leach, P. J., Salz, R. a Mealling, M. H. *A Universally Unique IDentifier (UUID) URN Namespace* [RFC 4122]. RFC Editor, červenec 2005 [cit. 2023-03-04]. DOI: 10.17487/RFC4122. Dostupné z: https://www.rfc-editor.org/info/rfc4122.

[16] Lombardi, A. *WebSocket*. First edition. Sebastopol: O´Reilly, 2015. ISBN 978-1-449-36927-9.

[17] Macrae, C. *Vue.js: up and running: building accessible and performant web apps*. Sebastopol: O'Reilly, 2018. ISBN 978-1-491-99724-6.

[18] Melnikov, A. a Fette, I. *The WebSocket Protocol* [RFC 6455]. RFC Editor, prosinec 2011 [cit. 2023-03-04]. DOI: 10.17487/RFC6455. Dostupné z: https://www.rfc-editor.org/info/rfc6455.

[19] Procházková, P. *Informační systém řešící rozvrhování*. Praha, CZ, 2011. Bakalářská práce. Vysoká škola ekonomická, Fakulta informatiky a statistiky. Dostupné z: https://vskp.vse.cz/28244_informacni-system-resici-rozvrhovani??page=777.

[20] Rozentals, N. *Mastering typescript*. Second edition. Birmingham: Packt Publishing Limited, 2017 [cit. 2023-05-03]. ISBN 978-1-78646-871-0.

[21] Singureanu, C. *Manuální testování - typy, Proces, Nástroje a další!* [online]. ZAPTEST, Apr 2023 [cit. 2023-05-04]. Dostupné z: https://www.zaptest.com/cs/manualni-testovani-co-to-je-typy-postupy-pristupy-nastroje-a-dalsi.

[22] Support, M. *Document collaboration and co-authoring* [online]. [cit. 2023-05-04]. Dostupné z: https://support.microsoft.com/en-us/office/document-collaboration-and-co-authoring-ee1509b4-1f6e-401e-b04a-782d26f564a4.

[23] Svirca, Z. *Everything you need to know about MVC architecture* [online]. Towards Data Science, May 2020 [cit. 2023-05-02]. Dostupné z: https://towardsdatascience.com/everything-you-need-to-know-about-mvc-architecture-3c827930b4c1.

[24] Thornton, J. a Otto, M. *About Bootstrap* [online]. [cit. 2023-05-04]. Dostupné z: https://getbootstrap.com/docs/5.3/about/overview/.

Příloha A

Testovací protokol

Přečtěte si následující otázky a v průběhu nebo po splnění scénáře je vyplňte, své odpovědi v případě potřeby zdůvodněte. Čím podrobněji budou odpovědi vyplněny, tím bude zpětná vazba pro autora práce cennější.

1. Podařilo se vám splnit všechny zadané úkoly?
2. Byly všechny názvy a tlačítka intuitivní a srozumitelné?
3. Bylo ovládání aplikace jednoduché?
4. Narazili jste na nějakou chybu nebo něco nefungovalo tak, jak se od toho očekává?
5. Máte ještě něco dalšího, co chcete uvést k aplikaci v rámci zpětné vazby?

Postupujte dle definovaného scénáře:

1. Vložte do systému data poskytnutá v souboru `testovaciData.json`.
2. Zkontrolujte, zda byla data úspěšně přidána.
3. V nastavení úpravte základní priority přednášejících tak, že o víkendu učit nemohou, naopak v pondělí odpoledne by možnost učit preferovali.
4. Vlož do systému přednášejícího pomocí formuláře.
5. Dále do systému pomocí formuláře vlož místnost. K přednastaveným rezervacím u ní přidej rezervaci na celé středeční dopoledne.
6. Přidej jeden předmět. Do tohoto předmětu vlož přednášku, kterou navštěvuje 100 studentů, a která se koná každý týden. Dále dvě cvičení, jedno v liché a další v sudé týdny. Nakonec přidej zkoušku, která bude bloková.
7. Přidejte záznam o tom, že pro přednášky vámi přidaného předmětu je vhodná místnost D105
8. Všechny lekce rozřaď do místností tak, aby nevznikly žádné kolize.
9. Výsledný rozvrh stáhni ve formátu JSON.