VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

INFORMAČNÍ SYSTÉM SOUKROMÉHO UČITELE S MOŽNOSTÍ ZKOUŠENÍ STUDENTŮ

INFORMATION SYSTEM OF A TUTOR WITH THE POSSIBILITY OF STUDENT EXAMINATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **ONDŘEJ ZÁRUBA**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. RADEK BURGET, Ph.D.**
SUPERVISOR

BRNO 2017

Zadání bakalářské práce/19304/2016/xzarub06

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Záruba Ondřej**
Obor: Informační technologie
Téma: **Informační systém soukromého učitele s možností zkoušení studentů**
Information System of a Tutor with the Possibility of Student Examination
Kategorie: Informační systémy

Pokyny:

1. Seznamte se se základními principy tvorby webových aplikací a odpovídajícími technologiemi.
2. Seznamte se s existujícím systémem pro podporu výuky angličtiny a jeho možnostmi.
3. Analyzujte požadavky na aplikaci pro soukromého učitele angličtiny, který bude zahrnovat podporu interaktivní výuky, možnost procvičování probrané látky, zobrazení výsledků testů učitelem a rezervační systém. Požadavky konzultujte s panem Martinem Říhou.
4. Navrhněte informační systém dle zjištěných požadavků. Použijte vhodné modelovací techniky.
5. Po dohodě s vedoucím implementujte navržené řešení pomocí vhodných technologií.
6. Ověřte funkčnost systému.
7. Zhodnoťte dosažené výsledky a diskutujte možná rozšíření.

Literatura:

- Gutmans, A., Rethans, D., Bakken, S.: Mistrovství v PHP 5, Computer Press, 2012
- Žára, O.: JavaScript - Programátorské techniky a webové technologie, Computer Press, 2015

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 4.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Burget Radek, Ing., Ph.D.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Hlavním tématem této práce je modernizace stávájícího systému pro podporu výuky realizované na základě požadavků pana Martina Říhy. V rámci jednotlivých kapitol se postupně seznámíme se základními principy tvorby webových aplikací, zaměříme se na analýzu stávajícího systému a navrhneme možné metody a postupy použité při jeho modernizaci a úpravách. Detailněji se budeme věnovat především kapitolám zabývajícím se samotným návrhem nových částí aplikace a postupně projdeme celý proces od samotného návrhu přes implementaci až ke stručné kapitole zabývající se testováním aplikace. Dále se práce bude zabývat speciální částí aplikace, která umožní učiteli a studentům vzájemnou *real-time* komunikaci v prostředí podobnému fungování interaktivní tabulí, jaké známe ze škol a jejímu využití pro zatraktivnění výuky.

Abstract

The main theme of this thesis is the modernization process of the existing system for support of teaching based on the requirements of Mr. Martin Říha. Firstly within the individual chapters we will gradually become acquainted with the basic principes of web apllication developing. Next We will focus on analysis of the current application and proposed methods and procedures used in its modernization and implementation changes. In more detail we will pay special attention to the chapters dealing with the design of new parts of the application and we will go through the whole process from design to implementation and testing. Attention will also be paid to special parts of the application that will allow teachers and students to interact in *real-time* communication in an environment similar to interactive whiteboards as we know them from schools and its use to make learning more attractive.

Klíčová slova

Informační systém podporu výuky, interaktivní spolupráce, real-time komunikace, WebSockets, MySQL, PHP, JavaScript, tvorba webových aplikací, technologie client-server

Keywords

Information system for teaching, Interactive collaboration, real-time communication, WebSockets, MySQL, PHP, JavaScript, web appliactions, client-server technology

Citace

ZÁRUBA, Ondřej. *Informační systém soukromého učitele s možností zkoušení studentů*. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Burget Radek.

Informační systém soukromého učitele s možností zkoušení studentů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Radka Burgeta, Ph.D. a podle přání a požadavků zadavatele této práce pana Martina Říhy. Další informace, které se v práci vyskytují mi poskytl pan Jiří Šifalda jakožto autor původního systému. Dále prohlašuji, že veškeré literární odkazy a publikace ze kterých jsem čerpal jsou uvedeny s seznamu literatury.

.....
Ondřej Záruba
7. května 2017

Poděkování

Při této příležitosti bych rád poděkoval především vedoucímu práce panu Ing. Radku Burgetovi, Ph.D. za vedení práce a cenné rady a postřehy ze strany odborné úrovně práce. Také panu Martinu Říhovi, který jako zadavatel práce umožnil její využití pro potřeby bakalářské práce a semestrálního projektu a poskytl při vývoji dostatečný prostor pro její realizaci a v neposlední řadě také panu Jiřímu Šifaldovi, autorovi původního systému na jehož základech celá tato práce vznikla za to že dovolil, aby se původní aplikace stala součástí této práce.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Moderní technologie, především informační systémy a webové aplikace, dnes téměř všichni používáme každý den v zaměstnání, ve škole i během svého volného času, kde nám ulehčují řadu každodenních činností, zjednodušují provádění některých úkonů a odvádí za nás spoustu triviální úkolů, které bychom jinak museli dělat ručně a věnovat jim tak více času. Naší snahou by proto mělo být vytváření systémů a aplikací přesně na míru požadavkům klienta tak, aby práce s nimi maximálně ulehčila práci uživatelů sytému či aplikace a aby tyto systémy či aplikace byly přínosem a zjednodušením pro jejich každodenní činnosti. V rámci této práce se proto detailněji podíváme na postup vylepšení a modernizace již existujícího systému sloužícího k výuce anglického jazyka.

Základem této práce, je již existující, v základech funkční a používaný systém pro výuku anglického jazyka soukromého učitele angličtiny pana Martina Říhy, který si jej nechal na zakázku vytvořit, pro zefektivnění výuky a komunikace mezi ním a jeho studenty. Hlavním účelem tohoto systému tak měla být poměrně jednoduchá agenda jednotlivých studentů a jejich lekcí s přehledem slovíček ke každé vyučovací lekci, doplněná o řadu funkcí, které měly jak učiteli, tak jeho studentům usnadnit výuku a rozšířit její možnosti. Prvoplánově měl být systém zaměřen na pořízení a rozšíření slovní zásoby a základů gramatiky anglického jazyka, s možností prověření získaných vědomostí formou jednoduchých kontrolních cvičení. Postupem času si začal klient uvědomovat, jak velkým přínosem pro podporu výuky takový systém může být a začal se zamýšlet nad možností úpravy a rozšíření aktuálního systému o řadu nových funkcionalit a zdokonalení, které ho za několik let používání starého systému napadly. A právě proces úprav, rozšíření a modernizace tohoto systému je předmětem této práce. V následujících několika kapitolách konkrétně rozebereme jednotlivé kroky vedoucí k modernizaci a rozšíření původně poměrně jednoduchého systému, který byl vytvořen před zhruba pěti lety.

V rámci první kapitoly provedeme stručnou charakteristiku postupů tvorby webových aplikací a stránek. Rozebereme jednotlivé programovací, skriptovací a ostatní jazyky obvykle používané při tvorbě webových aplikací spolu se základními technologiemi a knihovnami pro usnadnění práce a zanalyzujeme výhody či nevýhody jejich praktického využití při tvorbě webové aplikace.

Ve druhé kapitole se detailněji seznámíme s existujícím systémem, postupně probereme všechny jeho části, charakterizujeme jejich provedení a případnou rozšiřitelnost pro doplnění nových funkcionalit. Rozebereme použité jazyky a technologie, zhodnotíme zejména vhodnost jejich využití v dané situaci a v případě potřeby navrhneme lepší řešení, která se poté využijí při rozšiřování systému.

Ve třetí a čtvrté kapitole dále podrobně zanalyzujeme klientovy požadavky na úpravy a rozšíření stávajícího systému. Rozebereme možnosti zakomponování nových částí do starého systému a potřebné úpravy pro dosažení kvalitních výsledků práce. Zamyslíme se nad volbou vhodných technik, jazyků a technologií, pomocí nichž navrhneme postupy pro úpravu stávajícího systému a metody pro přidání nových funkcionalit.

V páté kapitole se zaměříme na samotnou implementaci navrhovaných změn. Během implementace budou využity informace a poznatky uvedené v předchozích kapitolách, zejména pak v kapitole zabývající se návrhem implementace. V této kapitole rozebereme aplikaci konkrétních knihoven a technologií se zdůvodněním proč jsou právě tyto technologie vhodné pro daný případ.

Závěrem, v poslední kapitole budou zanalyzovány a zhodnoceny veškeré provedené úpravy a dosažené výsledky. V rámci hodnocení vyzdvihneme části, které by šlo zkonstruovat lépe nebo pomocí lepších postupů a technologií a také navrhneme postupy pro možnost budoucího rozšíření.

Kapitola 2

Principy tvorby webových aplikací

Webové aplikace jsou dnes již běžnou součástí našeho života. Díky jejich rozšířenosti a neustále stoupající poptávce po nových technologiích máme dnes k dispozici velké množství programovacích, skriptovacích a dalších jazyků, které můžeme pro tvorbu webových aplikací použít. Stejně tak i díky velkým komunitám kolem těchto jazyků vzniká spousta užitečných nástrojů, komponent a knihoven, které můžeme při vývoji webových aplikací využít pro dosažení co nejlepších výsledků.

Důvodem proč stoupá obliba webových aplikací na úkor klasických počítačových (desktopových) nebo mobilních aplikací je především jejich univerzálnost. Uživatelé nemusí instalovat žádný další software (kromě webového prohlížeče), data jsou centralizovaná, takže uživatel nemusí například řešit synchronizace mezi jednotlivými zařízeními a webové aplikace se zpravidla dají bez problémů používat jak na počítačích, tak i na mobilních zařízeních nebo tabletech, což u klasických aplikací není tak jednoduché. Stejně tak i z pohledu programátorů a vývojářů představují webové aplikace daleko snazší cestu pro vytváření univerzálních aplikací, neboť zde není potřeba řešit kompatibilitu mezi jednotlivými operačními systémy nebo architekturami, takže s trochou nadsázky můžeme říci, že webové prostředí je jako jediné skutečně multiplatformní.

V následujících několika kapitolách dojde k podrobné analýze používaných architektur a jednotlivých jazyků, podíváme se na jejich uplatnění a vyzdvihneme několik typických knihoven, které jsou v daném jazyce používány.

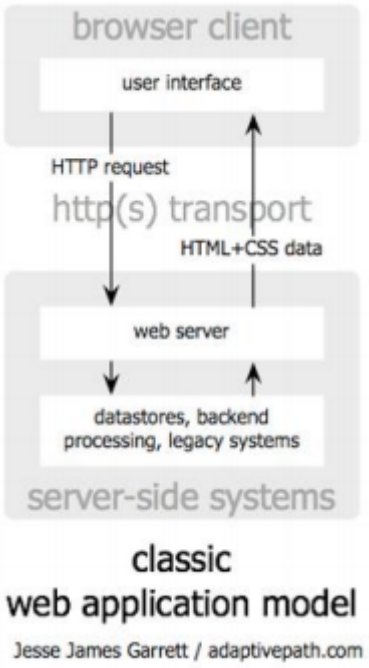
2.1 Architektury

Při vytváření webové aplikace je se nejprve potřeba zamyslet nad volbou vhodné architektury aplikace. Ačkoliv se někdy může zdát, že v prostředí webových aplikací není tento krok tak důležitý, opak je pravdou. Správná volba nám může v budoucnu ušetřit spoustu zbytečné práce. Je také ale potřeba brát ohledy na typ vytvářené aplikace a především na možnosti jejího dalšího rozšiřování a úprav. Důležité je také zbytečně nepodlehnout aktuálním trendům a opravdu zvážit možnosti a důsledky své volby.

K dispozici zde máme dvě hlavní architektury, mezi kterými si musíme vybrat a to čistě *serverové* aplikace a aplikace typu *client-server*. V rámci samotné implementace samozřejmě nemusíme použít jen jednu architekturu, pokud to aplikace skutečně vyžaduje je zpravidla bez větších problémů možné obě architektury vzájemně kombinovat tak, abychom dosáhli požadovaného výsledku.

2.1.1 Serverová architektura

Základní architekturou je klasická serverová architektura, kterou známe již z dob vzniku internetu [51]. Princip fungování je poměrně jednoduchý, uživatel prostřednictvím webového prohlížeče posílá HTTP(S) požadavek na vzdálený server. Server zpracuje uživatelův požadavek a na jeho základě vrátí připravený obsah webové stránky.



Obrázek 2.1: Schéma komunikace serverové architektury [7]

K aplikaci této architektury nepotřebujeme nějaké zvláštní prostředky, postačí nám libovolná aplikace zajišťující běh serveru, například Apache [30] nebo Nginx [23], programovací nebo skriptovací jazyk, kterým popíšeme chování webové aplikace. Po přijetí HTTP(S) požadavku server vykoná skripty uložené na serveru a základě jejich výstupů vygeneruje odpověď pro uživatelův prohlížeč.

Výhody

- Jednoduché řešení, stačí znalost jednoho serverového programovacího nebo skriptovacího jazyka.
- Jednoduší zabezpečení, díky tomu že se všechny operace provádějí na serveru, můžeme snadněji zabezpečit přístup.
- Efektivní zpracování z pohledu optimalizace pro vyhledávače (SEO), snazší strojové dolování dat.

Nevýhody

- Horší budoucí rozšiřitelnost, vzhledem k tomu, že všechny části aplikace běží na serveru, je obtížněji realizovatelné propojení například s mobilními nebo desktopovými aplikacemi.
- Synchronní zpracování požadavků. Uživatel musí čekat, než dojde ke zpracování obsahu celé stránky na serveru a až poté, je mu předložen výsledek.

2.1.2 Architektura client-server

Architektura typu *client-server* není také nic nového[40]. Běžně se s touto architekturou můžeme setkat u síťových aplikací (například DNS) nebo databázových systémů. V rámci tvorby webových aplikací, se ale tato architektura začala objevovat teprve před několika lety s příchodem technologie Ajax[39], ale velký skok udělala tato technologie teprve nedávno s příchodem *single-page aplikací*[44] a *webu 2.0*[24]. Hlavním principem této architektury je oddělení serverové a klientské části na dva odlišné celky, které mohou být implementovány za pomocí rozdílných technologie nebo jazyků. Komunikace jednotlivých částí je prováděna prostřednictvím Ajaxových dotazů na API serverové části. Proces komunikace při použití této architektury vypadá následovně. Klientská část (webový prohlížeč, mobilní nebo desktopová aplikace) generuje Ajaxové požadavky na serverovou část. Ta jednotlivé požadavky zpracuje a vrací zpět na klientskou část pouze vyžádaná data (nikoliv obsah celé stránky jako u klasické serverové technologie). Klientská část obdržená data zpracuje a zobrazí požadovaným způsobem.

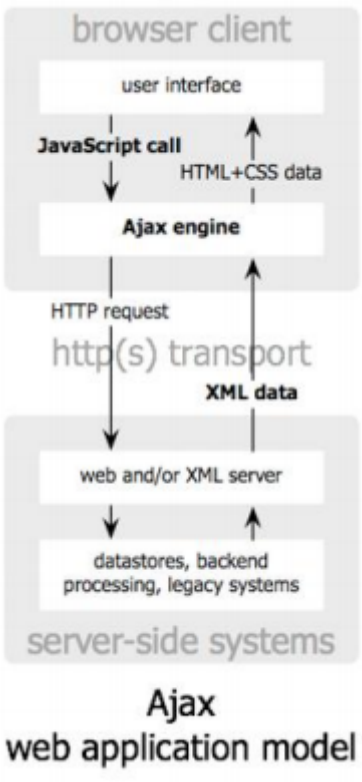
Klientskou část můžeme poté rozdělit ještě na dva další hlavní typy podle obrázku 2.3

- Tenký klient (*thin client*)
- Tlustý klient (*fat client*)

Tenký klient je označením pro klientskou část aplikace, která se nestará o žádnou složitější logiku. Úkolem tenkého klienta je odeslat požadavek na data na serverovou část a obdrženou odpověď zobrazit. Klientská aplikace tak nemá žádné náročnější požadavky na výkon nebo prostředky a právě toto zní dělá vhodného adepta na provádění operací v uživatelově prohlížeči. Na rozdíl od tenkého klienta, tlustý klient již provádí velké množství operací sám. Logiku aplikace tak nenese serverová část, ta zpravidla slouží pouze jako prostředník mezi databázovým systémem a klientskou částí, ale právě klientská část. Díky použití tlustého klienta, nemusí být serverová část vybavena výkonným hardwarem zatímco klientská část ano. Z tohoto důvodu se v prostředí klasických webových aplikací s tlustými klienty moc často nesetkáme, neboť webové prohlížeče jako primární nástroj pro provádění klientských operací není vhodným nástrojem pro složité, výkonově náročné operace. Hranice mezi tlustým a tenkým klientem však není nijak specifikována a mnohdy tak dochází ke kombinování obou přístupů podle toho, co vyžaduje aktuální situace.

Výhody

- Snadné rozšíření na další platformy (mobilní, desktopové nebo jiné aplikace)
- Možnost použití odlišných jazyků a technologií (tento bod je poněkud relativní, protože využití stejného jazyka na serverové i klientské části odstraňuje problémy s duplikací kódu)



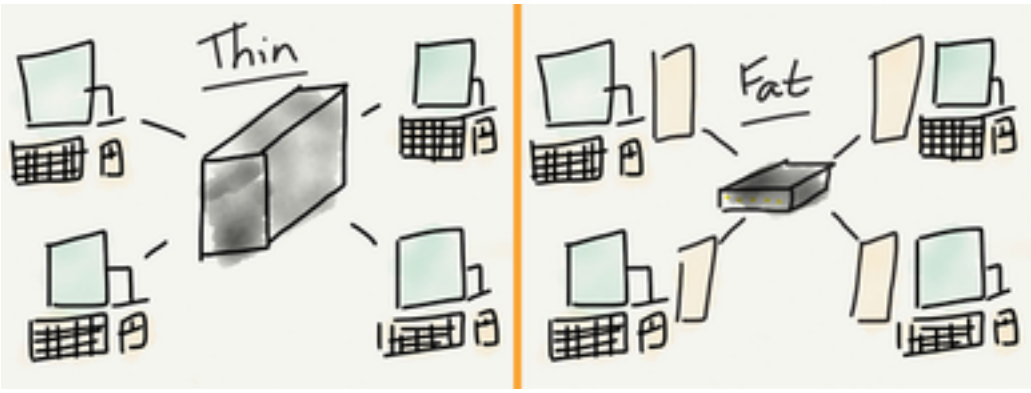
Obrázek 2.2: Schéma komunikace client-server [9]

- Rychlejší odezva webu, vzhledem k odstranění nutnosti zpracovávat celý web vždy od znova, ale můžeme aktualizovat jen potřebnou část[44].

Nevýhody

- Častá duplikace kódu v případě použití odlišných jazyků pro server a pro klientskou část. Mnohdy potřebujeme provádět na serverové a klientské části stejné operace, ale je potřeba je implementovat pomocí různých jazyků.
- Omezení daná technologií Ajax a podporou napříč webovými prohlížeči (např. upload souborů prostřednictvím technologie Ajax je možný v prohlížeči Internet Explorer až od verze 11, která byla uvedena v roce 2010[6][42])
- Potencionální problémy s optimalizací pro vyhledávače

Právě možnost oddělení logiky serverové a klientské části, je důvodem, proč v poslední době rapidně stoupá obliba této architektury. Díky možnosti zvolit různé programovací nebo skriptovací jazyky pro klientskou a serverovou část můžeme plně využít potencionál jednotlivých jazyků.



Obrázek 2.3: Tenký (*thin*) vs. tlustý (*fat*) klient [28]

2.2 Jazyky a technologie

Stejně jako architektury u webových aplikací se programovací a skriptovací jazyky používané v prostředí internetových aplikací dělí na dvě skupiny.

- *server-side* - Jazyky používané výhradně na serverové části aplikace
- *client-side* - Jazyky používané výhradně na straně klienta (prohlížeč)

Jazyků používaných pro čistě *client-side* implementaci není mnoho na rozdíl od jazyků využívaných na *server-side*. Jsou zde k dispozici tři hlavní zástupci. JavaScript, Adobe Flash a Microsoft Silverlight. Poslední dvě technologie mají pouze minoritní podíl na trhu a stále klesající trend[34] a proto se jimi již nebudeme dále zabývat na rozdíl od JavaScriptu, který bude podrobněji rozebrán v kapitole 2.2.4.

U programovacích a skriptovacích jazyků používaných na serveru je situace jiná. V dnešní době je pro serverovou část možné použít prakticky libovolný jazyk. Přesto však některé z těchto jazyků jasně dominují nad ostatními. Příkladem těchto jazyků mohou být například jazyky PHP, Python, Java, ASP.NET, Ruby, nebo Perl a nově také JavaScript (viz. kapitola 2.2.4), mezi kterým dominuje skriptovací jazyk PHP[35], kterému se budeme věnovat v kapitole 2.2.3.

Mezi další základní jazyky používané při tvorbě webových aplikací patří značkovací jazyk HTML a jazyk pro popis zobrazení elementů CSS, kterým se budeme stručně věnovat v kapitolách 2.2.1 a 2.2.2 a několik řádků textu věnujeme také nejčastěji používaným databázovým systémům v kapitole 2.3.

2.2.1 HTML

Značkovací jazyk HTML (HyperText Markup Language) je základním prvkem každé webové stránky nebo aplikace[41]. Jazyk jako takový vychází ze značkovacího jazyka SGML (Standard Generalized Markup Language)[45]. Od svého vzniku roku 1995[41] prošel jazyk HTML mnohými změnami. V prvních několika verzích jazyka HTML 2.0 (1995), HTML 3.2 (1997), HTML 4.0 (1997) a HTML 4.01 (1999) bylo HTML budováno striktně podle specifikací značkovacího jazyka SGML. Tato éra vývoje však byla v roce 2014 ukončena a jazyk

HTML přestává být závislý na jazyku SGML. Vzniká HTML5, což umožňuje opravu některých chyb, odstraňuje nepoužívané a zastaralé elementy a především přidává nové sémantické prvky[31]. Od svého vzniku HTML5 velmi rychle stoupá na oblibě a dnes ho využívá již 82% webových aplikací[38].

2.2.2 CSS

CSS (Cascading Style Sheets) je jazyk vyvinutý společností W3C společně s jazykem HTML (viz. kapitola 2.2.1) a slouží k popisu způsobů zobrazení elementů jazyka HTML[48]. V prvních verzích jazyka HTML, byly k dispozici elementy, které sloužily nejen k popisu obsahu, ale také k popisu zobrazení. Vzhledem k nedostatečné specifikaci a konkurenčnímu boji mezi webovými prohlížeči, tak vznikal velký problém s popisem vzhledu elementů, protože ho různé webové prohlížeče interpretovaly různým způsobem a proto W3C přišlo s novým jazykem, jazykem CSS, který má sloužit pouze pro popis zobrazení elementů.

Jazyk CSS prošel vývojem podobně jako jazyk HTML. Od první verze CSS1 až do aktuální verze CSS3 docházelo k postupnému definování nových selektorů, prvků a vlastností, díky kterým dnes můžeme vytvářet velmi kvalitně vypadající webové aplikace. V posledních verzích CSS2 a CSS3 přibyla také podpora takzvaných *media queries* (ve verzi CSS2 *media types*) díky kterým je možné optimalizovat webové aplikace podle velikosti zobrazovací plochy, typu zařízení, a několika dalších parametrů[32].

2.2.3 PHP

Typickým zástupcem čistě *server-side* jazyka je skriptovací jazyk PHP (PHP: Hypertext Preprocessor, který je nejčastěji používaným serverovým skriptovacím jazykem, s podílem 82% což ho značně povyšuje nad ostatní jazyky jako například ASP.NET, Javu nebo Ruby[36]. Stejně jako ostatní jazyky i PHP prošlo během let velkým vývojem. K dnešnímu datu je k dispozici 6 verzí, PHP 1.x (1995), PHP 2.x (1996), PHP 3.x, PHP 4.x (200-2005), PHP 5.x (2004-20014) a PHP 7.x (2015 až doposud). Velké změny ve vývoji PHP začínají verzí PHP 5.0 kdy se vývoj PHP začíná obracet směrem k objektově orientovanému přístupu a do jazyka se tak dostávají první konstrukce pro tvorbu objektů (tříd) a pokračují do dnes. Další významnou novinkou, která se objevuje s příchodem v PHP 7.0 je systém typové kontroly[14], který i přes dynamické typování tohoto jazyka umožňuje typovou kontrolu, což opět značně vylepšuje vlastnosti tohoto jazyka[55].

Díky velké komunitě vývojářů, kolem tohoto jazyka také vznikly a stále vznikají spousty knihoven a frameworků, které nám dnes velmi usnadňují práci. Mezi nejvýznamnější frameworky pro tvorbu webových aplikací tak můžeme zmínit například Laravel, Symfony2 nebo český Nette Framework[27].

2.2.4 JavaScript

Jedním z nejzajímavějších a v posledních letech nejdiskutovanějším jazykem je bezpochyby JavaScript. JavaScript je kombinací objektově orientovaného (prototypového) a funkcionálního jazyka. Ještě několik let zpět nebyl JavaSscript nijak populární jazyk. Jednalo se o čistě *client-side* jazyk, který se používal především pro vytváření modernějších prvků webových aplikací (vyskakovací okna, real-time validace formulářů, animace, atd...), obecně pro lepší uživatelskou přístupnost. V posledních letech však pomalu získává na oblibě.

Velkým vlivem na pozdější vývoj a oblibu tohoto jazyka byl příchod knihovny jQuery v roce 2006[43], která do značné míry odstranila nekompatibility mezi prohlížeči, zjednodušila

a unifikovala přístup k některým operacím a postupům, které byly postupem času zakomponovány (s drobnými úpravami) do standardů tohoto jazyka. Dalším velkým krokem kupředu byl příchod standardu ECMAScript 2015[12]/2016[13], který například přidává podporu tříd nebo dlouho očekávanou přímou podporu modulů, což značně zpřehledňuje výsledný kód.

Další velmi významnou událostí ve vývoji JavaScriptu byl příchod V8 engine v roce 2008[46], který otevřel možnosti pro vývoj nástrojů pro *server-side* implementaci JavaScriptu. Díky vývoji tohoto enginu v dalších letech začaly vznikat platformy pro praktické využití JavaScriptu na serveru. Zmínit zde můžeme například jednu z nejrozšířenějších platforem NodeJS[1], díky které popularita JavaScriptu začala stoupat. S příchodem NodeJS přišla kromě serverového zpracování i možnost vývoje desktopových aplikací, protože NodeJS dal možnost vzniknout dalším enginům postaveným na jeho základech zaměřujících se právě na tvorbu takovýchto aplikací použitím kombinace webových technologií (HTML, CSS, JavaScript), což opět přidává v posledních letech na atraktivnosti JavaScriptu. Příkladem těchto enginů může být například Electron[2] nebo NW.js[3].

2.3 Databáze

V poslední části této kapitoly, musí být také stručná zmínka o databázích používaných v prostředí webových aplikací. Databáze jsou nedílnou součástí každé alespoň trochu rozsáhlejší webové aplikace. Stejně jako u programovacích a skriptovacích jazyků je zde velké množství různých databázových systémů, které můžeme využít při tvorbě webových aplikací. Databáze jako takové jsou především záležitostí serverové části aplikace, ale není vyloučeno ani jejich použití v čistě *client-side* prostředí, kde se ale používají především pro zpřístupnění dat v offline verzích nebo jako dočasné úložiště před synchronizací s databází na serveru, například IndexedDB[21].

Databázové systémy můžeme rozdělit do dvou hlavních kategorií a to na SQL databáze, které využívají principu jazyka SQL, zpravidla se jedná o databáze relační, a NoSQL databáze, které data ukládají jiným způsobem než relační databáze. Mezi nejznámější a také nejpoužívanější SQL databázové systémy můžeme zařadit například MySQL (popřípadě MariaDB jako open-source odnož MySQL), PostgreSQL, Microsoft SQL Server nebo SQLite. Databáze MySQL se díky své rozšířenosti řadí mezi jeden z nejpoužívanějších databázových systémů[10]. Typickým a také nejpoužívanějším zástupcem druhé kategorie NoSQL databází je dokumentová databáze[50] MongoDB, která ukládá data jako dokumenty ve formátu podobnému formátu JSON.

[1] https://nodejs.org/en/
[2] http://electron.atom.io/
[3] https://nwjs.io/

Kapitola 3

Seznámení se s původním systémem

Jak již bylo zmíněno v úvodní kapitole (viz. kapitola 1), základem této práce je rozšíření a modernizace již existujícího a používaného systému pro podporu výuky soukromého učitele angličtiny.

V rámci této kapitoly dojde k podrobnější analýze stávajícího systému, zaměříme se na zvolené architektury, postupy, programovací a skriptovací jazyky, technologie a knihovny použité při jeho tvorbě. Pokusíme se zhodnotit kvalitu provedené práce a zaměříme se především na vhodnost volby jednotlivých postupů a technologií z dnešního pohledu tak abychom při rozšiřování tohoto systému mohli vycházet ze získaných znalostí a využít je pro maximalizaci kvality odvedené práce. Vzhledem ke stáří aplikace, která byla na zakázku vytvořena před více než třemi lety v roce 2013, a velmi rychlému a dynamickému vývoji technologií a postupů v oblasti informačních technologií, budeme místy poukazovat na problémy, nevhodná řešení či problémové implementace, které ale v době vzniku nebylo možné vyřešit tak jednoduše jako dnes.

3.1 Obecné informace

Než se začneme zabývat samotnou implementací, je potřeba se seznámit s některými základními informacemi o chodu aplikace, abychom měli představu o způsobu a četnosti jejího využití.

Aplikace byla vytvořena v roce 2013 na zakázku pro učitele angličtiny, kterému měla zjednodušit běžnou agendu spojenou s výukou. Cílem aplikace bylo umožnit učiteli interaktivní přehled všech jeho dosavadních i minulých studentů, umožnit jednoduchou správu jednotlivých lekcí s možností definovat ke každé lekci poznámky, obsahující například názvy probíraných témat, odkazy na domácí úkoly apod., a s možností ke každé lekci přiřadit nová slovíčka tak, aby se postupně rozšiřovala slovní zásoba studentů. Studenti mají možnost si také budovat vlastní slovní zásobu nahráváním slovíček do aplikace, která jim potom společně se slovíčky lekcí přidaných učitelem umožňuje jejich jednoduché procvičování spolu s automatickým hodnocením. Učitel má potom k dispozici jednoduchou statistiku s hodnocením úspěšnosti studentů a také s četností procvičování. Skrze aplikaci má také učitel možnost jednoduché komunikace se studenty formou příspěvků, které se zobrazují všem studentům přihlášeným do aplikace a také další možnosti procvičování gramatik a několik dalších užitečných funkcí usnadňujících výuku. Samotná aplikace je studenty a učitelem

používaná prakticky každý den. Vzhledem k menšímu počtu aktivních studentů (v řádech desítek) se tak nejedná o zrovna nejvytíženější aplikaci, ale je zde kladen velký důraz na intuitivní ovládání, přehledné prostředí a také podporu mobilních zařízení, která v posledních letech začínají být primárními zařízeními při využívání aplikace.

3.2 Architektura aplikace

Architektura původní aplikace je typu *client-side* (viz. kapitola 2.1.2). Jedná se tedy o dvě oddělené aplikace, každá část je implementovaná nezávisle na druhé části, pomocí rozdílných jazyků, technologií a postupů. Jedinou společnou věcí umožňující komunikací klientské a serverové části je API serverové části.

3.2.1 Klientská část

Jak již z kapitoly 2.1.2 víme, klientskou část můžeme rozdělit na dva typy a to buď na tlustý klient, nebo tenký klient, popřípadě kombinace obojího. Právě kombinace (nebo lépe, prolínání) obou přístupů je využito i v klientské části původní aplikace. Klientskou část aplikace můžeme jednoznačně označit za tenkého klienta, protože většina operací zůstává na serverové části, ale i přesto se zde najdou netriviální operace, které se provádějí na klientské části z důvodu komfortnějšího přístupu pro uživatele aplikace.

K implementaci klientské části byl využit jazyk CoffeeScript, který také někdy bývá místo jazyk označován jako preprocesor a to z důvodu nutnosti jeho následné komplikace do JavaScriptu. Ačkoliv se o tomto jazyku v této práci zatím neobjevila žádná zmínka, v době kdy byla tato aplikace vytvářena, byl CoffeeScript poměrně oblíbený jazyk. Na rozdíl od čistého JavaScriptu umožňoval vytváření tříd, zjednodušoval zápis kódu, neboť odstranil nutnost použití závorek a středníků. Také přidal několik užitečných konstrukcí pro práci s datovými poli či objekty[8]. V době vzniku této aplikace bylo použití tohoto jazyka velmi užitečnou pomůckou, která ale s příchodem JavaScriptového standartu ECMAScript 2015[12]/2016[13] přestala být takovou výhodou oproti čistému JavaScriptu a jeho obliba začala opět klesat[5]. Velkou nevýhodou tohoto jazyka byla potřeba kompilace do JavaScriptu před distribucí systému do ostrého provozu. Ke kompilaci zdrojových souborů CoffeeScriptu byl proto v původní aplikaci použit nástroj Grunt[1], který umožňuje automatizaci překladů pomocí předpřipravených úkolů (*tasks*). Tento nástroj nebyl použit jen pro překlad CoffeeScriptu na JavaScript ale také na celou řadu dalších činností jako například pro kompresi výstupních souborů, konkatenaci souborů použitých knihoven a podobně. Čím se dosáhlo úplné automatizace, která během vývoje velice ušetřila čas.

Celá klientská část byla naprogramována s využitím JavaScriptového frameworku AngularJS (první verze)[2], který je vyvíjen vývojáři společnosti Google. V době vzniku této aplikace se jednalo o poměrně zajímavou novinku, která velmi rychle expandovala a její obliba stupala. Díky základnímu paradigmatu vývoje pomocí MVC architektury[4] se velmi přiblížil frameworkům využívaným v ostatních programovacích jazycích což zapůsobilo na vývojáře a komunita kolem AngularJS se velmi rychle rozrůstala[37]. Díky základním principům frameworku AngularJS, se aplikace dá velmi dobře škálovat do samostatných modulů spojených v kompaktní celek až při kompilaci zdrojového kódu což umožnilo vytváření znovupoužitelných částí a velmi dobře posloužilo pro tvorbu původní aplikace.

[1] http://gruntjs.com/
[2] https://angularjs.org/

Po vizuální stránce není aplikace ničím výjimečná. Pro vybudování vzhledu a na záqkladní kostru aplikace byl využit CSS framework Flatstrap, který dnes již neexistuje[3] a celý design byl zpracován za použití jeho komponent. Tento framework byl postaven na frameworku Bootstrap[4], od vývojářů Twitteru, čímž se zaprvé velmi zjednodušila optimalizace pro mobilní zařízení, neboť většina používaných prvků měla již v době vývoje responsivní vlastnosti[33] a za druhé se zajistilo čisté a přehledné rozhraní, což vzhledem k tehdejším klientovým požadavkům bylo potřeba.

Závěr

Klientská část původní aplikace je velmi dobře zpracována. Díky nástrojům Grunt, který byl popsán výše a Bower[5] což byl tehdy velmi oblíbený nástroj pro správu balíčků (knihoven) a závislostí, se podařilo automatizovat celý proces vývoje, což při budoucích úpravách velmi zjednoduší práci. Zdrojový kód je velmi dobře strukturovaný a plně využívá možností, které mu dává použití frameworku AngularJS. Místy je bohužel logika poněkud překombinovaná, protože autor se snažil o univerzální přístup k řešením problémů, což je z globálního hlediska správný postup, ale v případě aplikace vyvíjené na míru klientových požadavků občas poněkud kontraproduktivní. Co se týče možností budoucího rozšiřování a úprav, kód je v dobrém stavu a návrh logiky velmi dobře usnadňuje případné přidávání nových vlastností.

3.2.2 Serverová část

Při návrhu serverové části původní aplikace, byl využit velmi kvalitní koncept modularizace. Serverová část aplikace byla rozdělena na samostatné oddělené moduly, kde každý modul měl jasně definované závislosti a byl distribuován jako samostatná knihovna (*package*) ve vlastním repozitáři, s jasně definovaným rozhraním, pomocí kterého poté docházelo k propojování a konfigurování jednotlivých modulů podle požadavků aplikace. Samotná serverová aplikace, tedy obsahovala pouze minimum aplikační logiky a jejím úkolem bylo pouze propojení jednotlivých modulů do jedné aplikace, která se navenek bude tvářit jako jeden velký celek. Rozdělením logiky do samostatných modulů ve vlastní verzovacích repozitářích systému git[6] se velmi zjednodušila práce se systémem jako s celkem, přehlednost kódů a oddělení závislostí značně přispívá k čitelnosti a rozšiřitelnosti kódu, což se v budoucích úpravách a následných rozšířeních zúročí.

Jako nástroj pro správu modulů (*packages*) a závislostí byl použit nástroj Composer[7]. Tento nástroj se stal téměř standardem při vývoji aplikací a knihoven v jazyce PHP neboť umožňuje velmi pokročilou správu závislostí, automatické načítání modulů do aplikace a instalace popřípadě aktualizace použitých knihoven se tak stala otázkou jediného příkazu v tomto nástroji, což opět značně urychluje a zjednodušuje vývoj aplikací. Do příchodu tohoto nástroje bylo potřeba všechny potřebné knihovny stahovat a registrovat do aplikace ručně, což značně komplikovalo jejich aktualizace a také velmi často způsobovalo nekompatibility mezi verzemi jednotlivých knihoven. Kombinací nástroje Composer a modularizací

[3] Původní zdrojové kódy jsou stále k dispozici na githubu (https://github.com/wilwaldon/flatstrap), ale framework již není dále vyvíjen a dokumentace k jeho starším verzím prakticky neexistuje.

[4] http://getbootstrap.com/

[5] https://bower.io/

[6] https://git-scm.com/

[7] https://getcomposer.org/

aplikační logiky tak vznikl velmi kvalitní koncept, díky kterému je serverová část aplikace velmi dobře připravena pro budoucí rozšiřování či úpravy.

Pro implementaci serverové části, jak již bylo zmíněno v předchozím odstavci byl použit scriptovací jazyk PHP ve verzi PHP 5.3, která byla v době vývoje aktuální verzí (viz kapitola 2.2.3). Pro potřeby této aplikace byl zvolen český framework Nette Framework[8] v tehdy nové verzi 2.1. Ačkoliv tento framework není v zahraniční tolik rozšířený jako například Laravel nebo Symfony2 zmiňované v kapitole 2.2.3, byla to podle mého názoru velmi dobrá volba. Jednotlivé zmiňované moduly aplikace byly implementovány jako rozšíření tohoto frameworku (*extensions*) což poté společně s použitím nástroje Composer umožnilo propojení modulů pouhým stažením a následnou jednoduchou úpravou konfiguračního souboru hlavní aplikace.

Jednou z nejdůležitějších částí serverové části aplikace je bezpochyby databázový systém a logika okolo něho. Pro potřeby této aplikace byl zvolen databázový systém MySQL, což vzhledem k poměrně malým rozdílům mezi ostatními typickými databázovými systémy, které jsou k dispozici pro využití ve spojitosti s webovými aplikacemi (viz. kapitola 2.3) a velkou oblibou této databáze působí jako dobrá volba. Pro interakci s databázovým systémem byl využit princip objektově relačního mapování (ORM), který umožňuje mapování relačních dat z databáze na objekty v objektově orientovaném jazyce[54]. K tomuto účelu byla zvolena knihovna Doctrine, která je jednou z nejpoužívanějších knihoven pro práci s ORM v PHP. Vývoj této knihovny byl ve velkém měřítku inspirován velmi oblíbenou knihovnou pro ORM v jazyce JAVA, Hibernate od níž převzala principy komunikace, které ale v posledních letech upravuje na míru jazyku PHP a tak se od původního návrhu poněkud odklání[49].

Volba tohoto přístupu a knihovny je ale poněkud sporným bodem. Objektově orientovaný přístup k tvorbě aplikací je velmi užitečná záležitost, která do značné míry usnadňuje vývojářům práci, protože nabízí lepší abstrakci nad daty a z tohoto pohledu by se tak jednalo o správně zvolený postup, ale zůstává zde otázka výkonnosti. Vzhledem k tomu, že serverová část aplikace komunikuje s okolním světem (především s klientskou částí aplikace) pouze prostřednictvím API a HTTP(S) komunikace, která probíhá v plain-textové podobě (viz. kapitola 3.2.3) a hlavní náplní serverové části je pouze zprostředkování dat mezi klientskou částí a databázovým systémem, mnohdy bez jakéhokoli dalšího zpracování na straně serveru, je potřeba zvážit výhody a nevýhody spojené s režií potřebnou pro konverzi relačních dat na objekty a potom zpět na do jednoduchých struktur potřebných pro HTTP(S) přenos. Tato otázka zde vzniká z jednoduchého důvodu. Data z relačních databází jsou v jazyce PHP získávána jako asociativní pole (`column => value`) v rámci vnitřních procesů Doctrine (samozřejmě i ostatních u ostatních knihoven pro práci s ORM) dochází k mapování tohoto pole na objekt, což sebou přináší režii se kterou je nutno počítat. V dalším kroku je potřeba takto namapovaná data předat zpět na klientskou část, což je prováděno pomocí HTTP(S) odpovědi a data tak musí být převáděna zpět na plain-textovou podobu. Zde právě dochází k onomu spornému momentu, protože je potřeba data namapovaná na objekt opět převést na základní podobu, kterou bude možné konvertovat do plain-texové podoby tedy zpět na asociativní pole. Je tedy nutné velmi dobře zvážit výhody objektově orientovaného přístupu pro práci s daty a potřebu zpětné konverze na asociativní pole, která je v mnoha případech zbytečná pokud se s daty v rámci aplikace neprovádějí žádné další operace.

[8] https://nette.org

V rámci seznamování se s aktuální podobou aplikace je také potřeba se zaměřit na samotný návrh databáze. Detailní schéma si můžete prohlédnout na obrázku A.1 v příloze.

Samotná databáze je navržena poměrně dobře. Splňuje požadavky třetí normální formy (3NF)[19] což zaručuje určitou kvalitu návrhu. Bohužel se zde vyskytuje pár případů, kdy návrh struktury databáze není úplně správný. Jak můžete na obrázku A.1 vidět, nachází se zde několik tabulek bez propojení skrze cizí klíče, přestože viditelně odkazují do jiné tabulky. V jednom takovém případě je zde totiž svázán záznam tabulky s různými tabulkami což není možné realizovat pomocí cizích klíčů. Autor původního systému proto použil systém, kdy identifikátor cizí tabulky není pevně vázaný pomocí cizího klíče, ale je uložený jako obyčejný skalární typ a k provázaní tabulek pak dochází až na úrovní aplikační logiky, kdy za pomoci zmíněného identifikátoru a informace o jméně tabulky, se kterou se bude záznam spojovat, se poskládá příslušný SQL dotaz. Tento poněkud neobvyklý návrh, který svým způsobem porušuje pravidla třetí normální formy (3NF)[19], vznikl potřebou postupného rozšiřování aplikace v době vývoje, kdy bylo potřeba bez zásahu do databáze přidat novou funkcionalitu, a toto byl jediný použitelný způsob aplikace bez provádění změn v databázi, které nebylo tehdy možné udělat. Úprava neměla údajně být trvalého rázu, ale mělo jít pouze o dočasné řešení, které ale nějakým nedopatřením v aplikaci zůstalo nevyřešeno.

HUB

Samostatnou kapitolou serverové části aplikace, o které zde bude stručná zmínka, je takzvaný HUB. V principu se jedná o samostatnou aplikaci, která se stará o autorizaci přihlášených uživatelů. Hlavním důvodem vzniku této části byl klientův požadavek na automatické přihlašování napříč všemi částmi aplikace, které jsou distribuovány na samostatných doménách a různých serverech. Původní návrh aplikace, využívá pro zapamatování přihlášení uživatele systém *localStorage*, která je ale vázána na konkrétní doménu a není tak možné zajistit zapamatování uživatelova přihlášení napříč všemi částmi aplikace[22]. V důsledku toho by byl uživatel nucen přihlašovat se do každé části aplikace zvlášť, i když prostřednictvím jediného autorizačního systému, což klientovi vadilo. Implementace této části je velice jednoduchá. V HUBU je prostřednictvím databáze udržován seznam přihlášených uživatelů s přiděleným tokenem (více v kapitole 3.2.3) a na klientské stráně aplikace je využíván princip *cross domain localStorage*[17], díky kterému můžeme zajistit bezpečné sdílení informace mezi jednotlivými částmi aplikace.

Závěr

Celkově až na drobné problémy například při návrhu databáze nebo při volbě objektového přístupu k databázi je serverová část, stejně jako část klientská, velmi profesionálně zpracována a velmi snadno rozšiřitelná. Díky rozdělení jednotlivých částí aplikace do naprosto izolovaných modulů distribuovaných ve vlastním repozitáři verzovacího systému git, je aplikace připravena pro další vývoj. Úpravy či případné rozšiřování funkcionality je díky tomu velmi jednoduché a snadno testovatelné. Toto oddělení ale má také několik málo praktických nevýhod, vzhledem k potřebě vzájemného provázání modulů občas dochází k problémům s kompatibilitou používaných knihoven třetích stran, především v případech kdy každý modul vyžaduje jinou verzi další knihovny, což mnohdy znamená nemalé zásahy do množství modulů kvůli udržení kompatibility.

Použitím kvalitních knihoven a frameworku, bylo dosaženo čistého návrhu aplikace, která je tak velmi snadno udržovatelná a rozdělení logiky aplikace je přehledné. Vzhledem k volbě standardních knihoven je také budoucí vývoj daleko jednoduší než při použití méně

známých nebo vlastních řešení, která by bylo potřeba podrobněji studovat a to mnohdy bez jakékoliv dokumentace, což by velmi ztížilo budoucí vývoj aplikace.

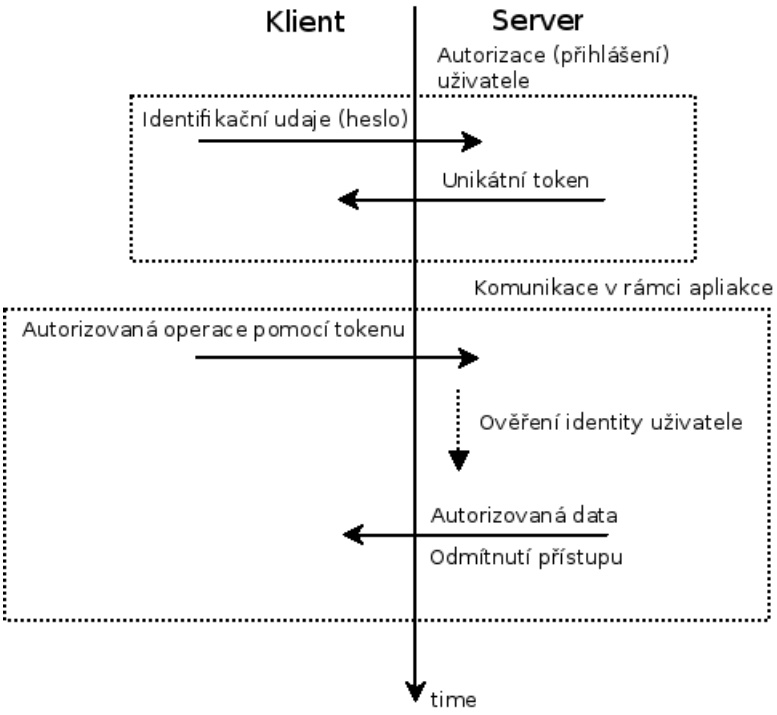
3.2.3 Komunikace mezi klientskou a serverovou částí

Jednou z nejdůležitějších částí aplikace je právě popis principu komunikace mezi klientskou a serverovou částí. Vzhledem k volbě architektury typu *client-server* a implementaci každé části pomocí jiného programovacího (skriptovacího) jazyka, bylo potřeba navrhnout způsob, jakým spolu budou tyto dvě části komunikovat.

Jak již bylo v kapitole 2.1.2 zmíněno, při volbě této architektury bývá typickým prostředkem pro zprostředkování komunikace Ajaxové volání ze strany klienta na server a stejně tak je tomu i v případě této aplikace. Serverová část pro tento účel poskytuje API (application programming interface)[47] jehož prostřednictvím jsou zpracovávány dotazy z klientské části, které je navrženo a implementováno pomocí RESTful architektury [25].

Co se samotného formátu přenášených dat týče, pro přenos dat byl zvolen formát JSON (JavaScript Object Notation). Tento formát je jedním z nejpoužívanějších formátů především díky jednoduchosti a snadnému zpracování v jazyce JavaScript. Oproti ostatním formátům pro přenos dat, jako například XML, se jedná o úspornější formát s možností přenosu strukturovaných dat[53].

Pro zabezpečení přenosu dat skrze HTTP(S) protokol se zde využívá principu `tokenů` (`token` je pseudo-náhodný řetezec znaků), jejichž princip je podrobněji vysvětlen v následujícím seznamu nebo na obrázku 3.1



Obrázek 3.1: Schéma zabezpečení komunikace

- Uživatel je vyzván k zadání autorizačních údajů

- Autorizační údaje jsou odeslány na server k autorizaci
- Server provede validaci dat a v případě kladného vyřízení generuje unikátní `token` spolu s platností tohoto `tokenu`.
- Server odpovídá na žádost klienta o autorizaci zasláním informací o uživateli a `tokenu` s časem expirace.
- Klient ukládá informace o uživateli spolu s identifikačním `tokenem` do `localStorage`.
- Při každém dalším dotazu na server, který vyžaduje autorizaci uživatele, je spolu s dotazem v HTTP hlavičce *Authorization* posílán vygenerovaný identifikační `token`, pro ověření identity uživatele.
- V případě kladného ověření, jsou zpět na klientskou část vrácena požadovaná data, v opačném případě dojde k odmítnutí požadavku.

Záver

Závěrem můžeme říci, že proces komunikace klientské a serverové části je zpracován dobře, bohužel k původní práci chybí jakákoliv dokumentace API, takže pochopení jednotlivých postupů bylo poněkud obtížnější, ale díky rozdělení aplikace na moduly, jak bylo zmíněno v předchozí kapitole, není větší problém se v tomto zorientovat.

Kapitola 4

Analýza požadavků

Jak bylo již v úvodní kapitole (viz. kapitola 1) několikrát zmíněno, obsahem této práce je rozšíření a modernizace již existujícího systému. V následujících několika kapitolách se proto zaměříme na požadavky klienta, které bude potřeba zohlednit při návrhu samotného postupu úprav, kterým se budeme podrobněji věnovat v kapitole 5. Před tím, než bude věnován prostor samotným požadavkům, je zde potřeba říci pár slov na úvod a provést stručnou sumarizaci těchto požadavků.

Požadavky na úpravu a modernizaci původního systému nevznikaly najednou, ale v průběhu několika let, kdy byl systém aktivně využíván jak učitelem, tak i jeho studenty. Velký vliv na budoucí vývoj systému tak mají především konkrétní požadavky vycházející z jeho používání. Většina hlavních požadavků, které budou podrobněji rozebrány v dalších částech této kapitoly, vzešly především z rozrůstajících se potřeb učitele, ale v nemalé míře také odrážejí potřeby a problémy studentů využívajících tuto aplikaci. Inspirací pro některé z požadavků byl také nástup modernějších nástrojů v oblasti nejen webových, ale i desktopových či mobilních aplikací, díky kterým bylo možné původní aplikaci rozšířit o funkcionality, které v době jejího vzniku nebyly známé či možné a také snaha maximálně automatizovat proces agendy spojené s výukou tak, aby všechny potřebné informace, které učitel potřebuje o svých studentech a jejich výuce znát, byly k dispozici v přehledné formě na jednom místě.

Nově vzniklé požadavky klienta mají jako hlavní cíl přenos agendy s dosavadního papírového řešení a poznámek vedených prostřednictvím kancelářských aplikací do jednotného webového systému, který by klientovi umožnil plnohodnotnější využití těchto informací. V následujícím seznamu jsou přehledně vyjmenovány stěžejní požadavky pro budoucí rozvoj aplikace. Samozřejmě se nejedná o všechny ve skutečnosti požadované a prováděné úpravy. Velké množství drobných úprav a oprav, ale není natolik zajímavé, aby se jimi tato práce zabývala.

- Aplikace umožnující interaktivní (real-time) spolupráci učitele a studentů prostřednictvím webového prohlížeče (viz. kapitola 4.1)
- Rezervační systém lekcí (viz. kapitola 4.2)
- Základní podpora plateb (viz. kapitola 4.3)
- Rozšířené možnosti lekcí (viz. kapitola 4.4)
- Rozšíření možností testování slovíček (viz. kapitola 4.5)

4.1 Interaktivní spolupráce

Jedním z prvních a zároveň i nejdůležitějších klientových požadavků bylo vytvoření uceleného systému, který by umožnil učiteli společně se studentem, či více studenty, provádět část výuky na dálku prostřednictvím počítače.

Vzhledem k neustále rostoucímu počtu studentů, klient začal poskytovat kromě klasických individuálních lekcí také možnost absolvovat některé lekce online prostřednictvím aplikace Skype. Tento způsob se postupem času stal mezi studenty velmi oblíbeným, a ačkoliv ne všechny lekce je možné absolvovat na dálku, studenti i učitel ho začali využívat čím dál více. Ačkoliv se díky videochatu prostřednictvím aplikace Skype značně zefektivnila online výuka, pořád zde chyběla možnost společné interakce s učitelovými materiály. Aplikace sice umožňuje například sdílení obrazovky, díky čemu mohl klient prezentovat svoje připravené prezentace, ale student neměl možnost do této prezentace jakkoliv zasáhnout. Zároveň s tím také stoupala učitelova potřeba využívání interaktivních nástrojů při prezenčních lekcích. Velké množství materiálů, které si klient pro své studenty připravoval, vyžadovaly nějaký druh společné interakce, což je v běžném prostředí bez nějaké dotykové tabule problém realizovat a výuka tak nebyla tak efektivní, jako kdyby prostory ve kterých výuka probíhala, byly touto tabulí vybaveny. Vzhledem kvysoké pořizovací ceně, která se pohybuje kolem 60000 Kč[1] a dalším nákladům spojeným s provozem a údržbou a také se snahou klienta být maximálně flexibilní, kdy výuka probíhá i u studentů doma, nebylo toto řešení interaktivní výuky možné. Na rozdíl od problémů s tabulí byla většina studentů vybavena vlastním počítačem, notebookem, tabletem a případně telefonem, který by se v dnešní době pro interaktivní výuku dal plně využít a mohl by tak z velké části nahradit klasické školní interaktivní tabule.

Klient tehdy proto přišel s žádostí zda by nebylo možné vybudovat systém, umožňující interaktivní výuku přímo prostřednictvím webové aplikace, kterou jeho studenti běžně využívají.

4.1.1 Specifikace požadavku

Přesná specifikace tohoto požadavku byla poměrně strohá, o to víc se budeme věnovat její podrobné analýze. Klient vznesl žádost, na vybudování aplikace, které by umožnila real-time interaktivní komunikaci mezi ním a jedním či více studenty. Aplikace samotná by měla umožnit učiteli vytvářet prezentace či cvičení, které by poté společně se studenty procházeli a pracovali s nimi. K dispozici by měli být klasické elementy jako například formulářové prvky, obrázky, textové boxy a v případě, že to bude možné, také přehrávače videí či zvuku. Aplikace by také měla umožnit přecházení mezi stránkami prezentace tak, aby učitel mohl připravit různé obrazovky s interaktivními prvky.

Důležitým bodem také byl požadavek, aby tato aplikace nebyla přímo součástí původního aplikace, ale aby byla vyčleněna bokem a přístupná komukoliv, ne jen registrovaným studentům.

4.1.2 Analýza požadavku

Vzhledem k poněkud neucelené představě, kterou klient ohledně aplikace pro interaktivní výuku má, zde bude potřeba celý požadavek analyzovat důkladněji než ostatní. Hlavním cílem zamýšlené úpravy by měla být aplikace podporující snadnou a interaktivní real-time spolupráci mezi učitelem s jedním či více studenty. Aplikace by měla sloužit jako doplněk

k videohovorům realizovaných přes aplikaci Skype a také by měla být použitelná samostatně během prezenční formy výuky.

Aplikace by měla umožnit učiteli vytvoření více stránkové prezentace, do které by mělo být možné vkládat celou řadu prvků, jako například formulářové prvky, obrázky, textová pole, videa, zvuky, a další elementy. Vzhledem k nedostatečné specifikaci klienta o množině všech využitelných prvků, musí být aplikace navržena tak, aby bylo možné kdykoliv co nejjednodušeji přidat nové interaktivní prvky podle přání klienta. Do aplikace by také mělo být možné psát pomocí pera, které by sloužilo jako zvýrazňovač a ovládalo by se pomocí myši na počítačích, případně prstem na dotykových zařízeních. Veškeré změny, které v prezentaci učitel nebo student provede, se musejí okamžitě synchronizovat s ostatními připojenými uživateli.

Učiteli by také mělo být umožněno jednotlivé prezentace ukládat a na jejich základech stavět další prezentace, nebo je využívat spolu s různými studenty. Úpravy prováděné v rámci lekce společně se studenty, by se ale neměly do takto připravených šablon nijak promítnout, aby tak bylo zajištěno, že jedna šablona bude využitelná pro více samostatných lekcí.

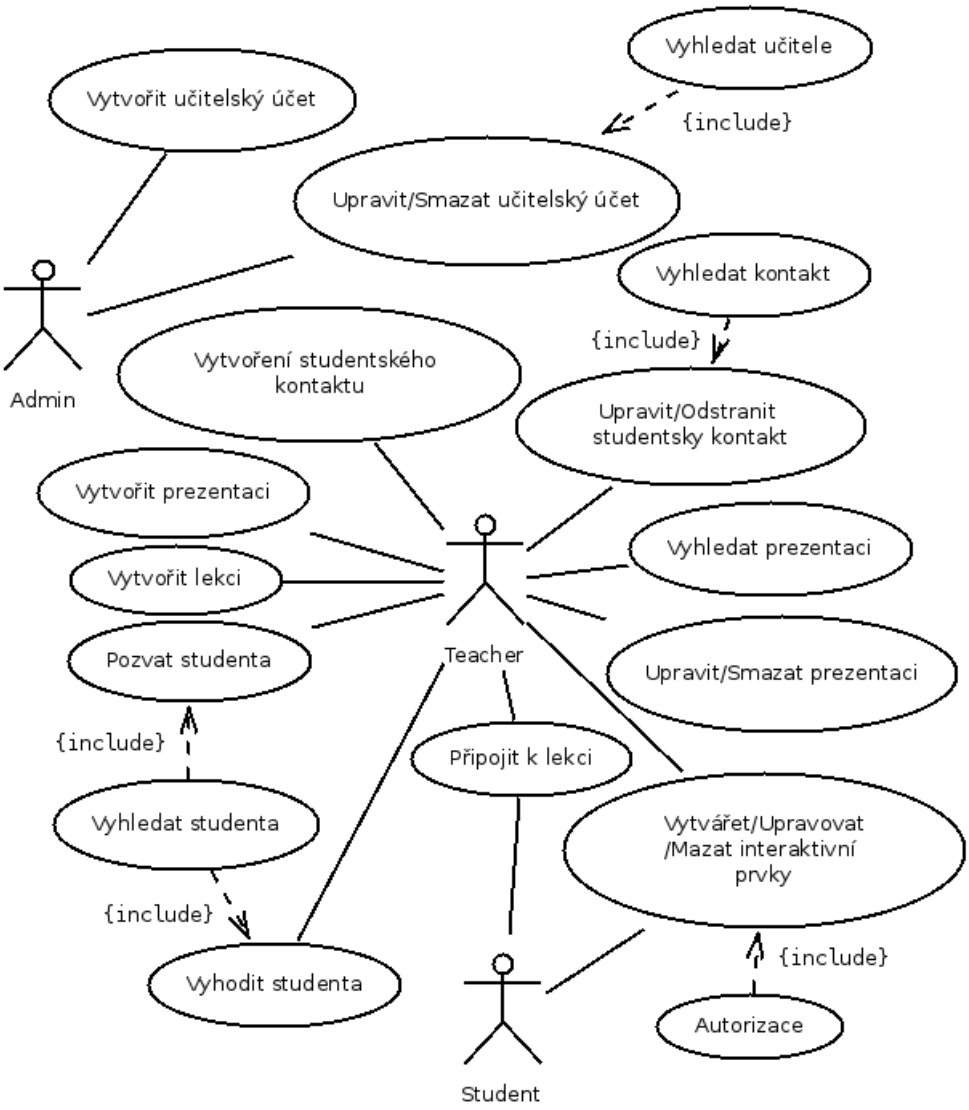
Jednotlivé lekce realizované prostřednictvím tohoto interaktivního systému musejí být zabezpečené tak, aby k nim nikdo nepovolaný neměl přístup, přestože nebude po uživatelích vyžadováno přihlášení. Právě vzhledem k požadavku na oddělení této aplikace od zbytku systému bude tato část také muset nějakým způsobem zpracovávat uživatelova data dodaná učitelem při posílání pozvánek na lekci.

Stručný výstup analýzy

Detailnější výstup z analýzy požadavku si můžete prohlédnout buď v seznamu níže, nebo na ilustračním obrázku případů užití (viz. obrázek 4.1).

- Systém zajistí automatickou a real-time synchronizaci se všemi účastníky lekce.
- Systém musí být oddělený od původní aplikace a nebude vyžadovat po uživatelích přihlášení.
- Přístup k jednotlivým lekcím musí být zabezpečen před vstupem nepovolaných osob.
- Systém musí uchovávat informace o studentech, které učitel do systému zadá.
- Učitelský přístup k aplikaci bude vyžadovat přihlášení, které ale nebude svázáno s původním systémem.
- Učitel bude mít k dispozici možnost vytvářet, mazat a upravovat prezentace.
- Každá vytvořená lekce bude přístupná přes konkrétní URL adresu kvůli možnosti rozesílání pozvánek prostřednictvím emailu.
- Systém umožní vytváření různých lekcí z jedné konkrétní prezentace.
- Změny provedené v rámci lekce nesmí ovlivnit strukturu původní prezentace.
- V rámci prezentace bude možné pracovat s množstvím interaktivních prvků jako například s formulářovými prvky, textovými boxy, obrázky, videi nebo audio soubory.
- Systém musí být snadno rozšiřitelný o další interaktivní prvky.

- Vzhledem k nevázanosti na původní systém, musí být k dispozici prostředí pro správu učitelských účtů.



Obrázek 4.1: Interaktivní aplikace - případy užití

4.2 Rezervační systém

Dalším velmi důležitým klientovým požadavkem bylo vytvoření systému, který by umožnil rezervace lekcí jednotlivými studenty.

Motivací pro vytvoření tohoto systému bylo především poměrně komplikované domlouvání mezi studenty a učitelem, vzhledem k neexistenci žádného veřejně dostupného seznamu volných termínů. Velkou komplikací pro učitele v tomto případě totiž byly dva velmi důležité body. Prvním bodem je fakt, že výuka angličtiny není klientovou hlavní pracovní náplní a je tedy potřeba přizpůsobit časy výuky jeho hlavní práci a druhým bodem je to,

že výuka probíhá oproti klasické školní (skupinové) výuce individuálně, kdy se učitel po dobu celé lekce věnuje pouze jednomu studentovi. Tyto dvě situace tak způsobovaly učiteli i studentům komplikace při domlouvání termínů jednotlivých lekcí.

4.2.1 Specifikace požadavku

Klientovým požadavkem v tomto konkrétním případě bylo vytvoření jednoduchého systému rezervací, kdy on jako učitel bude mít možnost zadat, kdy má čas a to ideálně i na několik měsíců dopředu a studenti budou poté mít možnost si z vypsaných volných časů vybrat, kdy se jim bude nejvíce výuka hodit. Zároveň s možností studentů vybrat si z volných termínů, by měl systém umožnit přímé zadání lekce učitelem například při dohodě se studentem během výuky, tedy bez nutnosti výpisu volného termínu a následné rezervace času studentem. Aplikace by měla také učiteli zajistit přehled všech nevyřízených žádostí o rezervace s možností potvrzení nebo případné zamítnutí každé žádosti, a to také prostřednictvím automatického emailu, který bude obsahovat odkaz pro potvrzení nebo odmítnutí žádosti jako nejsnazší a nejrychlejší možnost schvalování žádostí. Veškeré lekce, termíny a rezervace, by se také měly v případě zájmu studentů automaticky synchronizovat s aplikací Google Calendar, a to tak, aby data byla v každém okamžiku aktuální.

4.2.2 Analýza požadavku

Díky poměrně detailní představě a popisu požadavku od klienta si můžeme velmi přesně představit fungování výsledného systému. Přesto je však potřeba se na analýzu tohoto požadavku podívat poněkud formálněji. Před samotným návrhem je potřeba položit si několik jednoduchých otázek, které nám mohou pomoci pochopit zamýšlené chování a v neposlední řadě nám také pomůže sestavení diagramu případů užití, neboli tzv. *use-case diagram*[18].

Abychom mohli podrobněji pochopit zamýšlené chování aplikace, musíme si nejprve položit otázku: “Co by měl systém umět?”, na kterou se pokusíme odpovědět prostřednictvím seznamu pod tímto odstavcem. Dále je velmi vhodné vytvoření diagramu případů užití, který nám pomůže pochopit procesy, které budou v aplikaci vznikat (viz. obrázek 4.2).

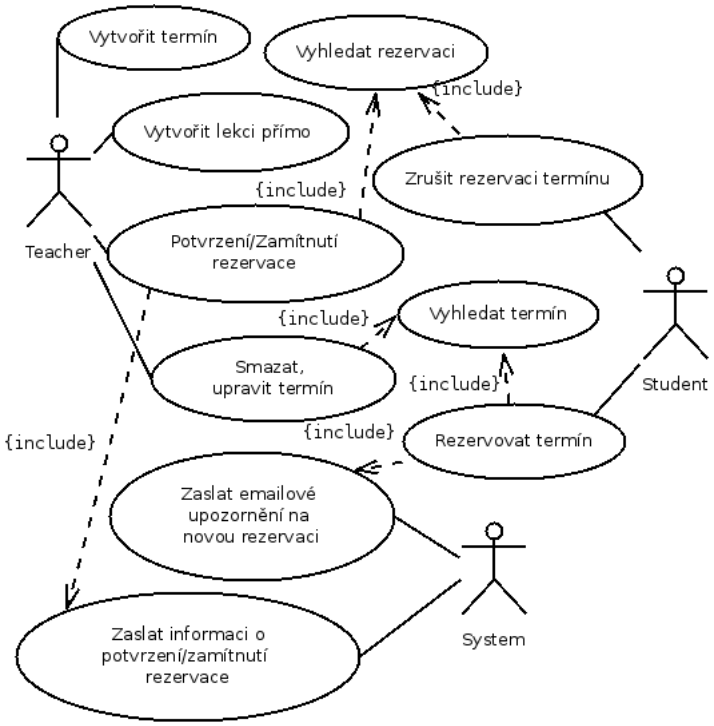
Hlavní funkcionality systému

- Systém by měl umožnit učiteli vytváření seznamu volných termínů a práci s tímto přehledem. Možnost upravovat, mazat nebo přidávat nové volné termíny by měla být jeho samozřejmou součástí. Formát volného termínu by měl být co nejjednodušší aby nekomplikoval učiteli zadávání termínů.

- Systém by měl umožnit přihlášeným studentům přehledné zobrazení seznamu volných termínů s možností podání žádosti o jeho rezervaci a to ideálně jedním kliknutím.

- Systém by měl studentům umožnit správu jejich žádostí o rezervace termínů tak, aby v případě změny názoru, měl student možnost žádost o rezervaci volného termínu zrušit.

- Systém by měl učiteli umožnit potvrzení nebo zamítnutí žádosti o rezervaci termínu, jak prostřednictvím webové aplikace, tak i prostřednictvím automaticky odesílaného emailu s detailními informacemi o studentově žádosti s možností kliknutím na odkaz v emailu žádost přijmout nebo zamítnout.



Obrázek 4.2: Rezervační systém - případy užití

- Systém by měl automaticky odesílat učiteli emaily v případě vzniku nové žádosti o rezervaci termínu.
- Systém by měl automaticky odesílat studentovi email s informací o potvrzení či zamítnutí jeho žádosti o rezervaci termínu.
- Systém by měl umožnit učiteli zadat lekci přímo bez předchozího vytváření volného termínu následovaného studentovou rezervací.
- Učitel by měl mít k dispozici seznam nevyřízených žádostí prostřednictvím webové aplikace a měl by být upozorněn na případné nevyřízené požadavky.
- Systém by měl zajišťovat automatickou synchronizaci s aplikace Google Calendar.

4.3 Systém plateb

Původní požadavek na zavedení systému plateb byl velmi jednoduchý, ale postupem času a především s postupem testování se nároky na tento systém zvyšovaly. Původní klientovou představou bylo vytvoření obyčejného, ručně zadávaného přehledu plateb, který by si mohl klient vést bokem pro kontrolu svého primárního systému evidence plateb, ke kterému do té doby využíval běžné kancelářské aplikace. Přesto přes první nedůvěru v systém vedení plateb uvnitř systému, kdy se klient obával nepřesností či ztráty dat, začal systém evidence plateb používat čím dál více, až jej nakonec začal využívat jako primární zdroj pro přehled plateb. Společně s myšlenkovým přerodem přišly i nároky na automatické evidování cen lekcí, studentská konta či automatické označování plateb. Přes všechny tyto

úpravy původního návrhu, jedna věc zůstala striktně v režii klienta a to zadávání plateb. Ačkoliv je technicky možné, aby studenti dobíjeli svá konta prostřednictvím webové aplikace, například pomocí kreditní karty, tuto možnost klient odmítl, především z důvodů častých hotovostních plateb placených osobně během lekce.

4.3.1 Specifikace požadavku

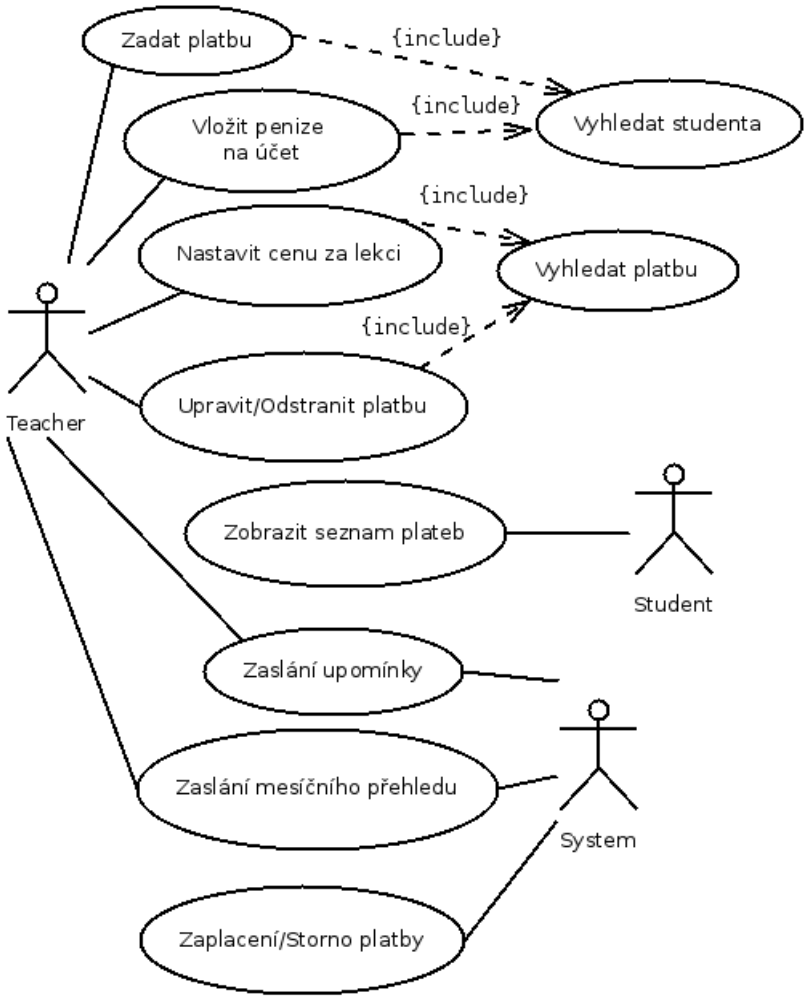
Po otestování všech možností a ujasnění všech představ tak dostáváme požadavek od klienta na vytvoření komplexnějšího platebního systému, který by měl umožňovat vedení studentských kód s možností manuálního zadání platby ze strany učitele na studentovo konto. Systém by měl učiteli umožnit definovat cenu za jednotlivé lekce, ideálně automaticky z předem nastavené ceny pro daného studenta. Platby za jednotlivé lekce by měl evidovat a být schopen zobrazit jak pro daného studenta, tak i pro všechny studenty najednou. Kromě plateb za lekce by měl systém umožnit další typy plateb jako například poplatek za pozdní zrušení lekce studentem, kompenzaci za zrušení lekce učitelem nebo další extra platby za nadstandardní služby, jak ze strany učitele, tak studenta. Systém by také měl umožňovat zasílání emailových přehledů a upomínek a to jak na vyžádání učitele, tak automaticky podle zadaných podmínek.

4.3.2 Analýza požadavku

Na rozdíl od kapitoly věnující se analýze požadavků rezervačního systému (viz. 4.2), kde bylo vycházeno především ze samotné specifikace, v této kapitole při analýze budeme muset zacházet více do hloubky.

Jak již ze samotného požadavku vyplývá, systém musí umožnit konto pro každý studentský účet v aplikaci. Stav těchto kont musí být automaticky aktualizován v případě připsání peněz na konto nebo naopak v případě platby. Každá lekce, ve chvíli kdy v systému vznikne, musí mít přiřazenu cenu, která by měla být automaticky odvozena z výchozí ceny za lekci vedené společně se studentovým kontem. Platba za tuto lekci ale nesmí být provedena (peníze nesmí být ze studentského konta odečteny) dříve než daná lekce skončí. Teprve po jejím proběhnutí musí být realizována transakce odečtení peněz ze studentova konta. Po dohodě s klientem, toto konto může nabývat i záporných hodnot, vyjadřující dluh studenta vůči učiteli. Tento dluh by ale neměl nijak omezovat možnosti studenta (například rezervace termínů). Každá platba v systému může nabývat dvou hodnot, *zaplaceno* nebo *nezaplaceno*, v závislosti na tom, jestli někdy v historii bylo na účtu dostatek prostředků pro zaplacení platby. Student sám o sobě nemá možnost s platbami nijak pracovat, může pouze sledovat historii svých plateb, zatímco učitel disponuje plnými právy a může zadávat jak nové platby, tak i příspěvky či vklady na studentovo konto. Kompletní přehled všech funkcí platebního systému si můžete prohlédnout v seznamu pod tímto odstavcem, popřípadě je k dispozici ilustrativní diagram případů užití na obrázku 4.3.

- Systém musí automaticky každému studentovi přidělit konto s počátečním stavem 0 Kč.
- Systém musí umožnit učiteli nastavení výchozí ceny lekce pro každého studenta zvlášť.
- Systém musí umožnit učiteli v případě potřeby změnit cenu lekce, jinak obvykle přenesenou z informace o výchozí ceně lekce, pro konkrétní platbu



Obrázek 4.3: Platební systém - případy užití

- Systém musí automaticky evidovat všechny proběhlé a připravené platby a na jejich základě automaticky upravovat stav studentského konta.
- Systém musí umožnit studentům zobrazit přehled všech jejich plateb za lekce, dobytí účtu a případné příspěvky od učitele.
- Každá lekce v okamžiku naplánování nese informaci i svojí ceně. Tato cena se do stavu studentského konta přenese až ve chvíli, kdy daná lekce proběhne. U plateb, které nejsou vázané na žádnou lekci, dojde k jejich přenesení okamžitě po vytvoření
- Každá platba evidovaná systémem může nabývat dvou stavů *zaplaceno* nebo *nezaplaceno*. O nastavování stavů se stará systém automaticky. V případě vzniku platby, je platba označena jako *zaplaceno*, pokud je na studentově kontě dostatek finančních prostředků, v opačném případě je platba označena jako *nezaplaceno*. V případě dobytí studentova konta se musí automaticky projít všechny nezaplacené platby od nejstarší po nejnovější a musí dojít k jejich zaplacení, pokud to finanční prostředky dovolí.

- Systém musí podporovat automatické rozesílání upomínek, přehledů a dalších informačních emailů stejně jako musí umožnit jejich manuální odeslání učitelem pro konkrétního studenta.

4.4 Rozšíření možností lekcí a emailing

Velmi obsáhlou kategorií požadavků, byly bezpochyby ty, týkající se úpravy systému lekcí, jejich rozšíření, upravení a doplnění o další návazné funkcionality. Systém lekcí, který v původním systému umožňoval jednoduchou agendu spolu se slovíčky, postupem času přestával stačit. Vzhledem k velkému množství dalších funkcí, které se na původní systém nabalovaly bylo potřeba rozšířit i možnosti této agendy lekcí tak aby více odpovídala nové podobě systému.

Zároveň s rozvojem lekcí a postupnou automatizací celého procesu výuky přišlo společně s požadavky na úpravu agendy lekcí také několik požadavků na automatické odesílání emailů prostřednictvím aplikace upozorňujících například na následující hodiny, možnost odesílání sumarizačního emailu po hodiny s obsahem probírané látky, přehledem slovíček či například seznamem domácích úkolů.

4.4.1 Specifikace požadavku

Systém by měl umožnit propojení lekcí a plateb tak, aby každá lekce měla svoji cenu s možností připsání poznámky. Pomocí rezervací termínů popsané v kapitole 4.2 by měl systém být schopen automaticky po schválení rezervace vytvořit lekci s příslušnou cenou. Každá lekce může být zrušena buď studentem nebo učitelem a za zrušení může být účtován poplatek jak studentovi, tak učiteli (v případě učitele se jedná o kompenzaci pro studenta). Učitel bude mít možnost po skončení právě probíhající lekce odeslat studentovi email s obsahem probírané látky. Stejně tak zde byla zdůrazněna potřeba uchovávání poznámek pro příští lekce aby tak učitel mohl studentovi budovat jakýsi plán výuky. Automatické upozorňování na lekce pomocí emailu by mělo být samozřejmostí.

Klient také vyjádřil požadavek, mít možnost nastavit u každého studenta standardní dobu lekce, což úzce souvisí s rezervačním systémem (viz. kapitola 4.2) a chtěl být upozorněn pokud naplánovaná či proběhlá lekce neodpovídá standardní délce aby mohl učitel případně upravit cenu za lekci. Stejně tak chtěl, aby mohl každou lekci označit speciálním štítkem, aby tak mohl označovat nadstandardní lekce.

Co se emailů a upozornění týče, byly zde vzneseny požadavky na automatické zasílání měsíčního přehledu lekcí společně s poznámkami a úkoly, a také upozornění na příští lekci aby student nezapomněl a několik dalších výše zmíněných emailů.

4.4.2 Analýza požadavku

Jak si můžeme povšimnout, požadavků na úpravy agendy lekcí zmiňovaných v kapitole 4.4.1 není málo. Úpravy oproti původnímu systému správy lekcí tak do značné míry mění chápání této agendy. Systém správy lekcí musí být nově primárně spojen s rezervačním systémem (viz. kapitola 4.2), ze kterého bude vznikat většina nových lekcí v aplikaci, což je oproti původnímu přímému zadávání velká změna. Lekce samotné se musí rozrůst o řadu drobných vlastností, jako například o informace o nestandardní délce, nebo extra štítku. Také musí dojít ke spojení lekcí s platebním systémem (viz. kapitola 4.3), pomocí kterého budou jednotlivé lekce účtovány studentům. Ekosystém kolem poznámek lekcí musí také

dostát velkých změn především kvůli změně přístupu a rozšíření o možnosti budoucího plánování výuky či zaznamenávání informací o domácích úkolech, byť jen jednoduchou formou. Vytváření lekcí musí být nově také svázáno s konkrétním studentem, aby bylo možné využít informací o standardní délce lekce nebo ceně. Také příchod emailových upozornění a souhrnů bude vyžadovat nemalé úpravy stávajícího systému, především z důvodů umožnění úprav šablon jednotlivých emailů a nastavení systému automatického odesílání. Kompletní přehled výstupu analýzy požadavků si můžete také v přehledné podobě prohlédnout v následujícím seznamu.

- Systém musí pevně spojit proces vytváření s rezervačním systémem.
- Množina informací udržovaných u každé lekce se musí rozrůst o položky pro zaznamenání informací o platbě (cena, poznámka k ceně), o indikátor nestandardní délky lekce, také musí přibýt možnost označení lekce štítkem a musí zde být k dispozici informace o stavu emailu po hodině (jestli byl souhrnný email odeslán nebo nikoliv).
- Systém musí umožnit párování plateb a lekcí.
- Systém musí zajistit pravidelné odesílání měsíčních přehledů lekcí spolu s úkoly, poznámkami a plánem na příští lekce.
- Systém musí umožnit automatické odesílání emailu s plánem příští hodiny.
- Systém musí umožnit po dokončení lekce, odeslání souhrnného emailu, který bude obsahovat informace o probírané látce, seznam slovíček přiřazených k dané lekci, plán příští výuky a seznam domácích úkolů a poznámek. Učitel musí být zajištěna možnost modifikovat šablonu tohoto emailu, před jeho odesláním.
- Systém musí umožnit učiteli upravovat šablonu libovolného emailu, který je prostřednictví aplikace automaticky odesílán.
- Musí být upraven systém poznámek tak, aby reflektoval potřeby učitele pro přidávání úkolů, poznámek a plánu k dané lekci.
- Systém musí umožnit učiteli i studentovi lekci zrušit. V případě studenta se jedná o možnost vytvoření žádosti na zrušení lekce. Společně s touto funkcionalitou musí být umožněno učiteli naúčtovat poplatek za pozdní rušení a v případě zrušení učitelem, pak kompenzaci studentovi. Systém by měl automaticky kontrolovat jestli je lekce rušena v termínu či nikoliv.

4.5 Rozšíření možností testování slovíček

Na rozdíl od předchozích kapitol, bude tato kapitola zabývající se analýzou požadavků na vylepšení systému zkoušení slovíček poměrně stručná. Klientovým přáním bylo rozšíření původního systému zkoušení slovíček, který by umožňoval jednoduchý systém zkoušení za základě samohodnocení. Na obrazovce bylo zobrazeno slovíčko a buď po stisknutí tlačítka nebo automaticky po uplynutí daného času, podle nastavení procvičování, se zobrazila správná odpověď. Student svoji odpověď nikam nezapisoval, ale po zobrazení správné odpovědi byl vyzván, aby se ohodnotil. Tento obyčejný způsob, ačkoliv v mnoha ohledech vyhovující, byl ale učitelem považován za poněkud neprůkazný, neboť předpokládal poctivé hodnocení studentů což v některých případech nebylo úplnou samozřejmostí. Z tohoto

důvodu klient přišel s nápadem na vytvoření “her”, které by umožnily průkaznější způsob hodnocení studentů.

4.5.1 Specifikace požadavku

Klient nastínil možnost vytvoření několika typů her, které by umožnily regulérní hodnocení studentů. Hlavním principem mělo být vybírání slovíček z možností, aby se předešlo zbytečné chybovosti studentů, pokud by se při vyplňování například uklepli. Původní klientovou představou bylo vytvoření dvou, hry Sloupce(Columns) a hry Střelba(Shooting).

Hra Sloupce by měla fungovat na principu spojování překladu a slovíčka. Ve dvou sloupcích vedle sebe by bylo zobrazeno x slovíček (podle nastavení hry) a jednotlivé překlady by byly proházeny. Student by poté musel v zadaném časovém intervalu klikáním správně propojit jednotlivé páry.

U hry Střelba by měla být situace podobná. Student by dostal na výběr z x možností (v závislosti na stavení hry) překladů. Po krátké časově prodlevě by se na hrací ploše náhodně zobrazilo slovíčko a student by musel velmi rychle kliknutím označit správný překlad.

Všechny hry by měly mít k dispozici celou řadu nastavení, jmenovitě by mělo být možné nastavit počet životů, čas na sestřelení/vybrání slovíčka, počet kol, zprávy pro studenta pro případ úspěšného nebo neúspěšného dokončení hry, i možnost nastavit pozadí hry, by bylo také velmi kladně hodnoceno. Všechny hry musí být přístupné přes unikátní url adresu, aby bylo možné posílat odkazy na hry pomocí emailů nebo například vkládat do poznámek z hodin.

4.5.2 Analýza požadavku

Jak již ze samotného požadavku vyplývá, sytém by měl umožnit rozšíření stávajícího systému zkoušení o nové “hry”. Ačkoliv klient používal při specifikaci svého požadavku slovo hra, vzhledem k podstatě samotného popisu se o hru ani tak nejedená, a lepším označením by pravděpodobně bylo interaktivní zkoušení slovíček.

Systém tohoto interaktivního zkoušení, by měl učiteli umožnit vytvářet hry pro konkrétní studenty (tedy z jejich slovní zásoby), které by mohl učitel odeslat studentovi skrze vygenerovaný unikátní odkaz. Zároveň by ale každý student měl mít možnost spouštět hry individuálně bez potřeby vytvoření hry učitelem. Vytváření hry, ať již učitelem nebo studentem by mělo být prováděno skrze univerzální obrazovku s nastavením, kde by bylo možné specifikovat počty životů, časové limity, obrázky na pozadí, texty pro úspěšné nebo neúspěšné dokončení hry a další specifická nastavení. Hra by se měla automaticky spustit po dokončení těchto nastavení a nastavení konkrétní hry by mělo být přístupné přes unikátní adresu tak, aby jak již bylo zmíněno, mohl učitel studentovi odeslat odkaz nebo aby se student mohl kdykoliv vrátit odehrát si svojí již dříve připravenou hru. K tomuto účelu by zde měl být také k dispozici seznam připravených her. Po dokončení hry by se měl studentovi zobrazit přehled všech odehraných slovíček společně s výsledky a chybami. Tento přehled by měl být automaticky odeslán učiteli na email a zároveň by se mělo provést automatické ohodnocení odehraných slovíček pro potřeby statistiky.

Podrobný seznam výstupu analýzy si můžete stejně jako v předchozích případech prohlédnout pod tímto odstavcem.

- Systém musí umožnit rozšíření způsobu zkoušení o další interaktivní možnosti.

- Příprava her bude probíhat pomocí univerzální obrazovky, kde student či učitel nastaví jak globální (společné pro všechny hry) tak i specifické požadavky jednotlivých her.
- Po dokončení každé hry systém zobrazí obrazovku s výsledky a hodnocením dané hry, včetně seznamu chyb.
- Systém automaticky po dokončení hry studentem odešle email učiteli s přehledem výsledků dané hry. Systém také umožní studentovi toto hodnocení odeslat i na jeho email prostřednictvím tlačítka k tomu určeného.
- Každá hra musí být přístupná přes unikátní odkaz.
- Hra Sloupce
  - Každá hra je omezená počtem životů a počtem slovíček.
  - Možnost nastavení počtu párů.
  - Možnost nastavení času jednoho kola.
  - Hráč bude pomocí myši vybírat a spojovat jednotlivé páry.
  - Každý spojený pár musí být možné zrušit.
  - Po každém kole student uvidí výsledky.
- Hra Střelba
  - Každá hra je omezená počtem životů a počtem slovíček.
  - Možnost nastavení počtu zobrazovaných možností.
  - Možnost nastavení rychlosti mizení slovíčka.
  - Slovíčko se zobrazuje náhodně v prostoru hrací plochy.
  - Možnost nastavení jestli bude docházet k sestřelování slovíčka či překladu.
  - Drobné animace pro oživení hry.
- Každá hra má k dispozici nastavení názvu, popisu, pozadí, textu pro úspěšné či neúspěšné dokončení a některé další nastavení pro vylepšení kontrastu v případě použití pozadí.
- Systém musí evidovat seznam her pro každého studenta s možností odstranění hry či změnou jejího nastavení.

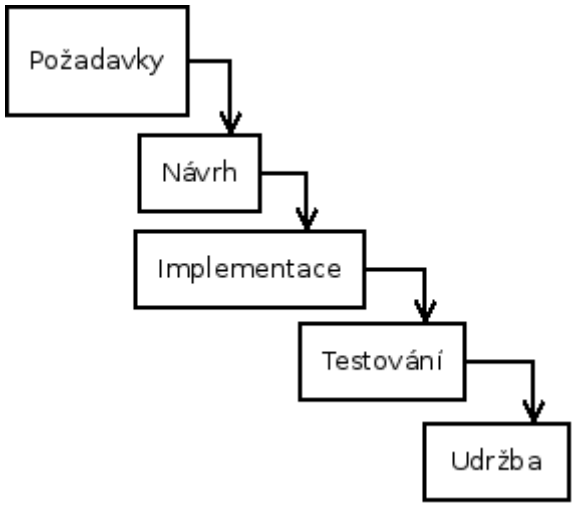
Kapitola 5

Návrh aplikace

Předchozí kapitoly byly zaměřeny především na teoretickou část a úvod do problematiky vývoje webových aplikací a stránek a také na analýzu požadavků a seznámení se se stávajícím systémem. V této kapitole se již zaměříme na praktičtější principy návrhu samotné implementace klientových požadavků a to především za použití vhodných modelovacích technik a postupů využívaných v oboru softwarového inženýrství. Podrobněji se zde také zmíníme o návrhových vzorech, které by nám mohly usnadnit přístup k řešení některých problémů, které vyplývají z analýzy požadavků v kapitole 4 a také zaměříme se na návrh databáze.

5.1 Návrh úprav a modernizace

Z oboru softwarového inženýrství známe několik modelů životních cyklů používaných při vývoji software. Z těch základních a důležitých pro samotnou práci, je potřeba zmínit především model vodopádový 5.1 a inkrementální, který rozšiřuje principy vodopádového modelu.



Obrázek 5.1: Vodopádový model

V duchu těchto modelů tak musí dojít k rozvržení jednotlivých úprav na samostatné celky, které budou co nejméně záviset na okolí, aby nebyly příliš náchylné na změny klientových požadavků a jejich následné dekompozici na dílčí podproblémy, kterým je potřeba se v rámci návrhu věnovat.

Z úvodní kapitoly této práce víme, že hlavním cílem obsahu této práce je zaměřit se na metody úprav a modernizace stávajícího systému. V kapitole 4 jsme podrobně rozepsali klientovy požadavky na potřebné úpravy aplikace a v následujících kapitolách tak budeme vyházet primárně z nich. V rámci analýzy byly jednotlivé požadavky rozděleny do několika hlavních skupin, které ale samy o sobě neříkají nic o rozdělení odpovědností v duchu inkrementálního modelu vývoje. Na základě analýzy těchto požadavků a přání klienta bylo rozhodnuto, že většina změn, které bude v rámci původního systému potřeba provést, vznikne odděleně od původního systému. V důsledku čehož tak vzniknou dva úplně nové systémy, jeden se zaměřením na rezervace lekcí, který pod sebe pojme všechny požadavky na přidání systému plateb, úpravy systému lekcí, a emailing (viz. kapitola 4) a druhý s primárním zaměřením na interaktivní výuku. Právě toto oddělení do samostatných aplikací byl důvodem zvolení inkrementálního modelu na rozdíl od modelu iterativního. Pouze požadavek na přidání nových her bude proveden v rámci původního systému, který tak díky svému účelu zůstane primárně jako agenda lekcí s možností procvičování slovíček. Obě tyto samostatné aplikace však budou záviset na původním systému, přes který se bude zajišťovat autorizace uživatelů a se kterým se v případě potřeby budou data synchronizovat. Díky *client-server* architektuře původní aplikace, budou nově dodávané aplikace komunikovat s původní aplikací pomocí API a HTTP(S) požadavků.

Vzhledem k poměrně velkému množství použitých návrhových vzorů, jejichž popis by zabral několik stran této práce, vyzdvihneme pouze několik základních a důležitých, které detailněji popíšeme. Návrhové vzory typu *Facede* nebo *Abstract*, které všichni známe a které jsou velmi často použity zde tak nebudou zmiňovány.

5.1.1 Aplikace pro interaktivní výuku

Tou nejdůležitější částí systému a také částí s největší potřebou návrhu implementace je bezpochyby systém pro interaktivní real-time výuku. Vzhledem k tomu, že se nejedná o typickou součást webových aplikací, je potřeba se v této kapitolo více zamyslet nad samotným návrhem aplikace, který by měl co nejvíce odpovídat klientovým požadavkům. Vzhledem k tomu, že agenda kontaktů, účtů a emailů je prakticky totožná jako v části Rezervace v kapitole 5.1.2, zaměříme se zde především na návrh samotného real-time prostředí, aby nedocházelo k zbytečné duplikaci textu.

Technologie pro real-time komunikaci

Aplikaci jako takovou musí být možné provozovat pod webovým prohlížečem, což s sebou přináší některá omezení. V rámci webových aplikací máme k dispozici dva hlavní způsoby přenosu dat. Tím prvním způsobem je typické Ajaxové volání serverového API. Tento způsob je pro většinu běžných operací vhodný a naprosto dostačující, ale bohužel pro potřeby real-time komunikace není vhodný. Každý Ajaxový požadavek je zpracováván zvlášť a vyžaduje poměrně značnou režii spojenou se zpracováním HTTP formátu požadavku do podoby, se kterou bude umět pracovat programovací jazyk použitý na serveru (viz. obrázek 5.2). Vzhledem k tomu, že v praxi není možné otočit směr komunikace, tedy aby server odesílal data na klienta, bylo by potřeba se ze strany klienta neustále dotazovat na změny

prostředí, tzv. *pooling*, který by abnormálně zatěžovat uživatelův webový prohlížeč. A ačkoliv může být zpracování požadavku velmi rychlé, webový prohlížeč není stavěný na odesílání desítek požadavků za sekundu a takováto zátěž by vedla k pádu prohlížeče.

Obrázek 5.2: Proces zpracování Ajax požadavku

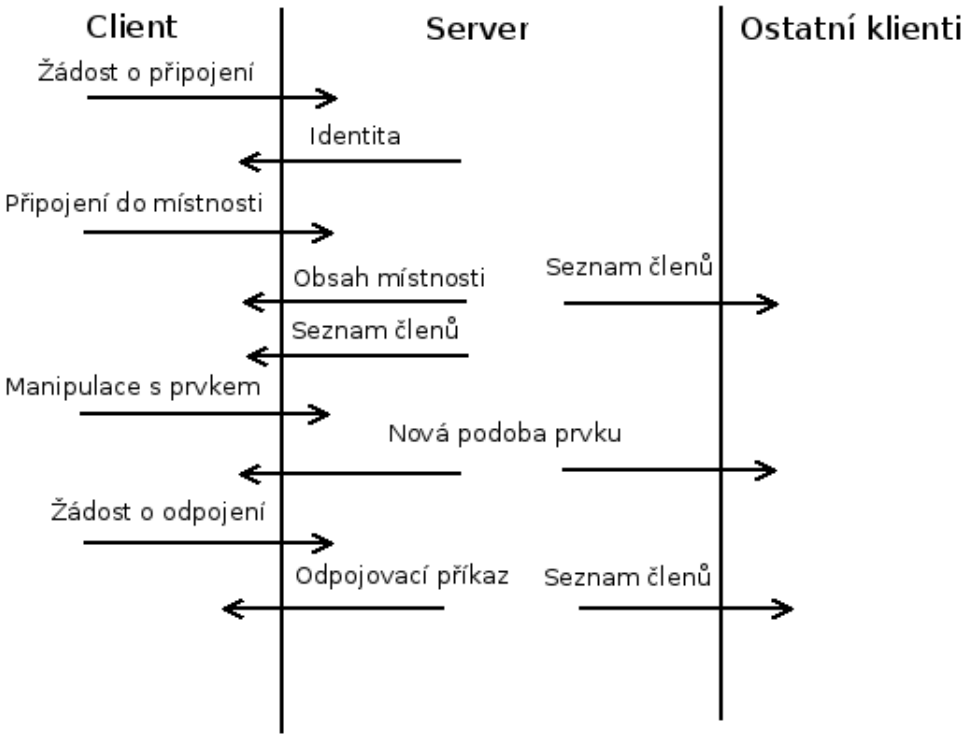
Druhou poměrně novou technologií je technologie WebSockets. Tato technologie je prakticky totožná s technologií klasických socketů, které můžeme znát z principů síťové komunikace. Principem webových socketů je vytvoření trvalého spojení mezi klientem a serverem (popřípadě peer-to-peer, tedy klientem a klientem) na konkrétním portu, které bude využito k přenášení dat. Díky trvalému spojení, které není možné prostřednictvím klasické HTTP komunikace udržovat, odpadá režie potřebná ke startování serveru, zpracování požadavku a interpretaci programu, čímž se celý proces značené urychlí[20]. Technologie webových socketů je určena právě pro asynchronní real-time komunikaci a může být využita například technologií WebRTC, která standardizuje video a audio real-time přenos[11].

Tato technologie se tak jeví jako vhodný kandidát pro implementaci interaktivního systému. Díky ní nebude problém provádět velmi častou synchronizaci mezi uživateli a dosáhneme tak požadovaného výsledku.

Formát a schéma přenosu

Vzhledem k volbě využití webových socketů pro realizaci real-time komunikace je potřeba se zamyslet nad otázkami, které by jinak u běžné HTTP komunikace nebylo nutné řešit. První otázkou, kterou je potřeba specifikovat je systém místností a princip komunikace. Aby byl dodržen klientův požadavek na možnost vytváření lekcí pro různé studenty ze stejných podkladů, budou podklady a samotné lekce odděleny. V rámci podkladů lekcí bude mít učitel možnost definovat podobu prezentace a interaktivních prvků. Lekce samotná bude reprezentována místností, podobně jak je známe z aplikací pro real-time komunikaci (chat), která bude tvořena obsahem prezentace a do níž se budou studenti a učitel připojovat. Princip celé komunikace by poté měl probíhat způsobem, který je popsán v seznamu níže (položky

označené číslem se provádějí synchronně, položky označené odrážkou asynchronně), nebo na obrázku 5.3.



Obrázek 5.3: Schéma komunikace webových socketů

1. Student žádá o připojení.
2. Server ověří identitu uživatele na základě unikátního odkazu a vrátí informace potřebné k jeho identifikaci.
3. Student se připojí do místnosti.
   - Systém uloží informaci o připojení nového člena do databáze.
   - Systém odešle obsah aktuální místnosti zpět přihlašujícímu se studentovi.
   - Systém pošle všem účastníkům informaci s aktuálním seznamem účastníků.
4. Uživatel provede změnu nějakého prvku.
   - Systém uloží informaci provedené akce.
   - Systém aktualizuje obsah místnosti.
   - Systém odešle všem účastníkům informaci o změně.
5. Uživatel požádá o odpojení.
   - Systém uloží informaci o odpojení uživatele.
   - Systém rozešle všem účastníkům aktualizovaný seznam účastníků.

- Systém odešle ukončující příkaz pro žadatele.

Přenášená data bude vhodné formátovat pomocí mechanismu, se kterým bude možné snadno pracovat jak v rámci klientské, tak serverové implementace. Prostřednictvím webových socketů je možné přenášet i binární data, což by mohlo ušetřit značné množství přenášených dat, ale vzhledem k poměrně malé množině informací, které bude potřeba přenášet, si vystačíme s klasickým formátem JSON, pro který bude velmi jednoduché je zpracovávat jak na straně klienta tak serveru.

Volba architektury a programovacího jazyka

V předchozích kapitolách jsme se zaměřili na návrh samotné implementace real-time komunikace. Z důvodů, které vyplývají z použití technologie webových socketů je vidět potřeba použití architektury typu *client-server*. Také co se implementačních jazyků týče, jsme zde poměrně hodně omezeni na využití programovacích jazyků podporujících komunikaci pomocí webových socketů.

Pro implementaci klientské části stejně jako v ostatních případech je vhodné použít JavaScript neboť je v porovnání s ostatními dostupnými technologiemi v implementaci webových socketů nejdál. Pro serverovou část bude ale nutné oproti původní aplikaci a rezervačnímu systému vybrat jiný jazyk než PHP. Ačkoliv je i PHP možné použitím různých knihoven rozšířit o podporu technologie webových socketů, jazyk jako takový není k tomuto účelu primárně určen. Aplikace postavené na bázi této technologie totiž vyžadují dlouhodobý běh bez nutnosti restartování aplikace, zatím co jazyk PHP byl navržen spíše na krátkodobý běh. Interpret jazyka PHP totiž jednotlivé zdrojové soubory zpracovává až ve chvíli, kdy obdrží požadavek a jejich interpretace se provádí vždy od začátku do konce, což sebou přináší značnou režii. Vhodným řešením tak je využití jazyka JavaScript jak na klientské tak serverové části. Pro interpretaci jazyka na serverové části je potřeba použít engine, který se postará o zpracování a interpretaci zdrojových souborů. Takovým enginem bude bezpochyby NodeJS, který je stavěn právě na provozování dlouhodobě běžících aplikací. Využití JavaScriptu na obou stranách aplikace s sebou také přináší možnost sdílení zdrojového kódu, což při správném provedení může značně zjednodušit práci.

Databáze

Výběr vhodné databáze a princip ukládání dat v případě interaktivního systému není tak úplně triviální jako v ostatních případech a proto je mu věnována krátká kapitola této práce. Vzhledem k potřebě ukládání obsahu interaktivní lekce, tedy všech elementů, jejich pozic, rozměrů a spousty dalších variabilních informací, je potřeba navrhnout způsob, jak tyto údaje uložit do databáze takovým způsobem, aby nedocházelo k porušování normálních forem[19]. Protože dopředu neznáme všechny v budoucnu používané interaktivní prvky, ani množinu informací, které ke každému prvku bude potřeba uchovávat, je potřeba navrhnout dostatečně flexibilní databázovou strukturu, která umožní přidávání nových prvků a vlastností bez nutnosti zásahu do samotné databáze.

Z předchozích kapitol víme, že pro přenos a uchovávání dat bude využit formát JSON. Stejně tak celý obsah interaktivní lekce bude uchovávaný v podobě struktury ve formátu JSON, která bude obsahovat veškeré informace o všech prvcích a jejich rozložení. JSON jakožto variabilní struktura vhodná pro uchovávání většího množství strukturovaných dat je tak horkým kandidátem pro ukládání do databáze. Jak víme z kapitoly 2.3, máme možnost využití například dokumentové databáze MongoDB, která data ukládá právě ve formátu

odpovídajícímu formátu JSON. Data jsou v této databázi ukládána jako dokumenty, které mohou mít libovolnou podobu, což by mohlo odpovídat celému konceptu interaktivního přenosu, bohužel se ale jedná o NoSQL databázi a museli bychom tak řešit spoustu komplikací, které jinak řeší SQL databáze samy (například spojení tabulek, cizí klíče a podobně). To by vzhledem k ostatní agendě spojené s interaktivní aplikací jako správa kontaktů, účtů a emailů nebylo ideální řešení. Samozřejmě by zde byla možnost použití dvou oddělených databázových systémů, klasickou SQL databázi na ukládání běžných data a NoSQL databázi pro ukládání dat v rámci interaktivní výuky. Toto řešení s sebou ale nese problémy se sdílením primárních klíčů mezi jednotlivými typy databází a v neposlední řadě také narážíme na problém s hostováním takovéto aplikace neboť většina poskytovatelů umožňuje využití pouze jednoho typu databáze.

Dalším faktorem je předpoklad, že s daty uloženými ve formátu JSON nebude potřeba nijak pracovat na úrovni databáze. Z pohledu databáze se bude jednat o atomickou hodnotu, ačkoliv ve strukturovaném formátu, a proto volba klasické SQL databáze nebude problém. Z možných SQL databází MySQL a PostgreSQL bude potřeba ale také vybrat tu nejvhodnější. Vzhledem k použití MySQL na ostatních částech aplikace, by její volba byla pravděpodobně správná, ale je potřeba ověřit, jestli by PostgreSQL přece jen nepřinesla pro daný konkrétní případ nějaké výhody. PostgreSQL totiž umožňuje ukládání dat ve formátu JSON, přímo podporuje JSON jako datový typ sloupce a definuje několik metod pro práci s ním, což by se na rozdíl od MySQL, které JSON formát nepodporuje mohlo hodit. Ale vzhledem k tomu, že z pohledu databáze budou data ukládaná v tomto formátu brána jako atomická není potřeba nějakých dalších funkcí pro práci s nimi a proto je volba MySQL databáze vhodnou volbou.

Ukázku konkrétní podoby schématu databáze týkající se interaktivní výuky si můžete prohlédnout na obrázku 5.4.

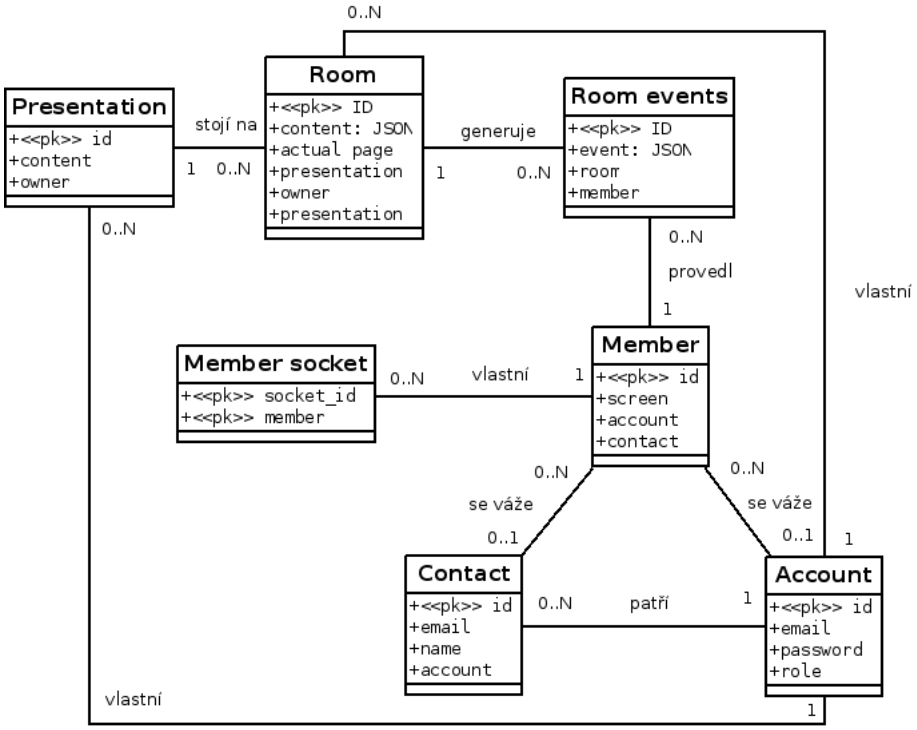
5.1.2 Rezervační systém

Jak bylo v úvodním odstavci zmíněno, pod samostatnou aplikaci rezervačního systému spadá i většina dále zmiňovaných požadavků od systému plateb až po emailing. Vzhledem k tomuto poměrně velkému množství dílčích částí, které je potřeba implementovat, se musíme zaměřit na jednotlivé prvky a zmínit klíčové body pro jejich návrh na základě zadaných požadavků.

Volba architektury a programovacího jazyka

Rezervační systém jako samotná aplikace by měl využívat architektury *client-server* popsané v kapitole 2.1.2, díky které můžeme oddělit logiku serverové a klientské části, čímž získáme možnost dalšího pozdějšího rozšiřování a udržíme tak také konzistenci s architekturou původní aplikace. Vzhledem k potřebám jednotlivých částí se jako nejvhodnější řešení zdá použití architektury MVC (Model-View-Controller), která umožní přehledně oddělit jednotlivé části aplikace od sebe[4]. V případě serveru zde budeme mluvit o modifikaci MC(Model-Controller), protože vzhledem k oddělení serverové a klientské části, které spolu budou komunikovat prostřednictvím API a HTTP(S) požadavků není na straně serveru View (pohled) potřeba.

Každou část rezervačního systému (klientská, serverová) bude vhodné implementovat pomocí různých jazyků. Pro potřeby klientské části zde přichází v úvahu pouze JavaScript a to především z důvodů výhod, které s sebou tento jazyk přináší (viz. kapitola 2.2.4) a také z důvodu udržení konzistence s původním systémem. Pro serverovou část bude nejlepší



Obrázek 5.4: ER diagram interaktivních lekcí

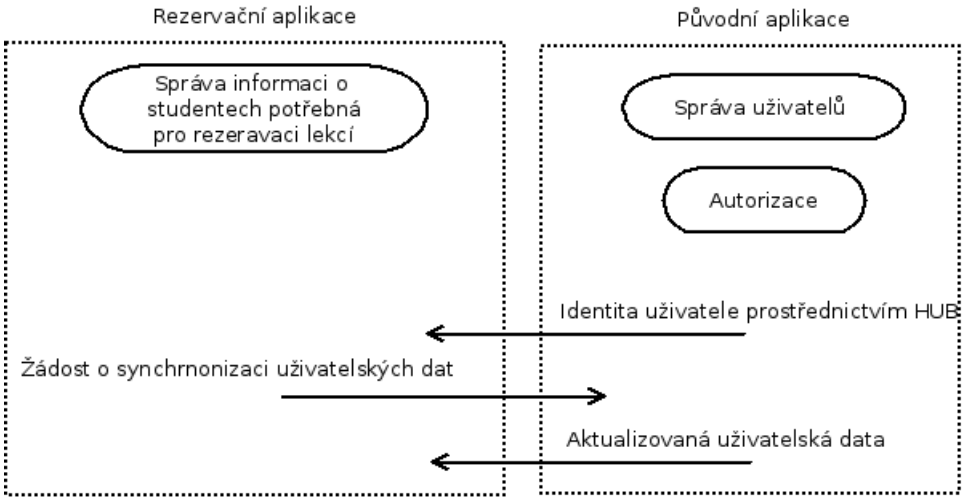
využít scriptovacího jazyka PHP, tedy také stejného jazyka jako u původní aplikace jednak z důvodu možnosti sdílení původního zdrojového kódu v nové aplikaci, ale také kvůli jeho rozšířenosti.

Rezervace

Samotný návrh systému rezervací není nijak extrémně komplikovaný a není zde ani moc bodů, které by stály za detailnější rozebrání, a proto se zaměříme především na návrh databáze. Uživatelské účty, které v rámci této části aplikace budou vedeny, budou záviset na účtech vedených v původní aplikaci, která zajistí autorizaci uživatelů a synchronizaci případných změn. V rámci databáze rezervačního systému se tak bude vést jednoduchá agenda účtů převzatých z původní aplikace (částečná kopie dat kvůli úspoře požadavků na původní aplikaci) rozšířená o nové vlastnosti jako například výchozí cena nebo standardní délka lekce (viz. obrázek 5.5). Hlavní logika správy uživatelských účtů však zůstane plně v režii původního systému.

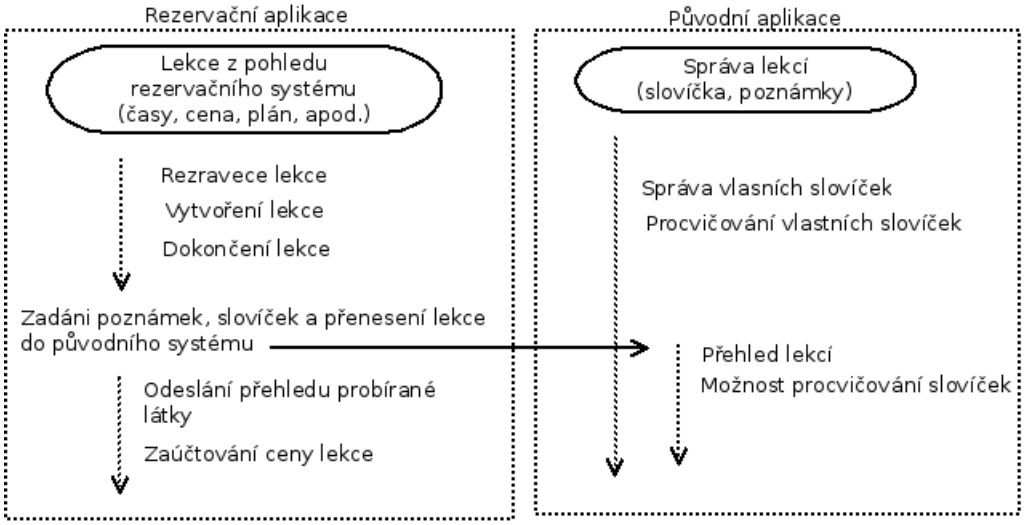
V rámci databáze bude také potřeba udržovat seznam volných termínů s vlastnostmi `start`, `konec datum` a také tabulku lekcí, která umožní rozšíření vlastností vedených o lekcích v původním systému. V tomto případě bude systém synchronizace dat fungovat podobným způsobem jako u uživatelských účtů. Lekce v rámci rezervačního systému bude mít přidělen unikátní identifikátor a s jeho pomocí ji bude možné spárovat s lekcí v originální aplikaci.

V rámci snahy o modernizaci přístupu se tak původní aplikace odsunuje na pozadí a její primární účel již nebude agenda lekcí, nýbrž zaměření především na samotná slovíčka. Primární částí aplikace, tedy místem, kde se bude odehrávat většina agendy, by měl nyní



Obrázek 5.5: Schéma komunikace a závislostí uživatelských informací

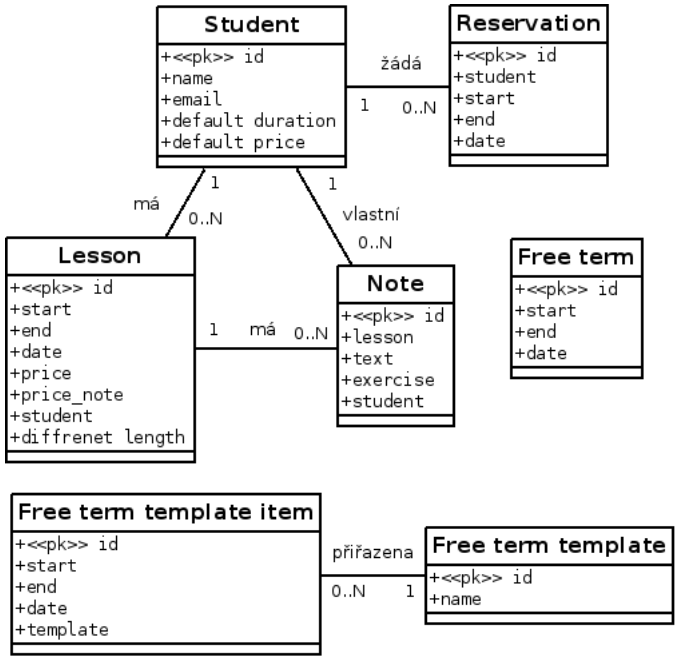
být samotný rezervační systém. Z této nové části systému se budou data jednostranně synchronizovat do původní aplikace, ale obrácená cesta, tedy synchronizace z původního systému do nového, již nebude umožněna. Ačkoliv by nebyl problém realizovat ani tento směr toku dat, klientovým přáním je postupně původní systém zbavit závislostí na lekcích a primárně ho využívat jako aplikaci pro testování slovíček. V rámci původní aplikace zůstane možnost i nadále vytvářet lekce, ale tyto lekce se již nepromítnout do rezervačního systému. Toto zachování funkcionality zde bude ponecháno především kvůli kompatibilitě a to do doby než bude plně dokončena modernizace systému. Pro ilustraci toku se můžete podívat na diagram 5.6



Obrázek 5.6: Schéma komunikace a závislostí lekcí

Stejně tak bude systém evidovat veškeré žádosti o rezervace termínů společně s nutnými vlastnostmi pro jejich identifikaci.

Ukázkový návrh ER diagramu, který se váže na část rezervací si můžete prohlédnout na obrázku 5.7



Obrázek 5.7: ER diagram rezervací

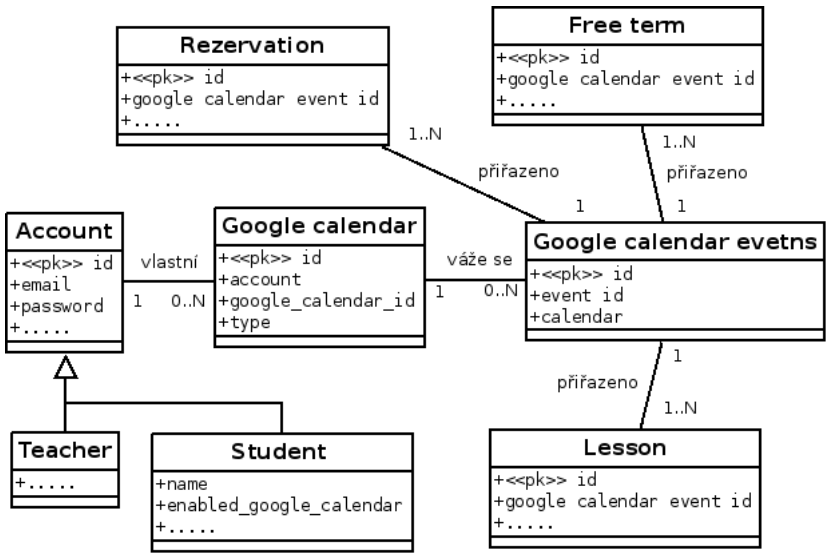
Synchronizace s aplikací Google Calendar

V rámci požadavku na synchronizaci dat s učitelovým a studentskými účty v aplikaci Google Calendar, bude také potřeba navrhnout způsob zajištění komunikace. K dispozici je zde několik možných přístupů jak synchronizaci dat zajistit. Za prvé je možné využít přihlašování prostřednictvím účtů Google pomocí protokolu OAuth, což bylo klientem odmítnuto i přes zjevné výhody tohoto řešení oproti ostatním. Díky přihlášení do aplikace pomocí Google účtu by tak synchronizace se studentovým kalendářem byla velmi jednoduchá, rychlá a přehledná, protože by všechno nastavení pod uživatelovou kontrolou. Druhým řešením jak sdílet kalendář s aplikací Google Calendar je pomocí standardu iCalendar, který popisuje princip a formát výměny kalendářových údajů mezi různými servery[2]. Jedná se o velmi dobře aplikovatelný a univerzální přístup, který umožní import do aplikace Google Calendar, bohužel v tomto případě není možné ze strany naší aplikace vynutit aktualizaci dat, kterou Google Calendar provádí jednou za několik hodin. V původní dokumentaci, kterou již bohužel není možné dohledat se vývojáři Google Calendar zmiňovali o aktualizaci jednou za 8 hodin. V nové dokumentaci bohužel zmínka o frekvenci aktualizací není, ale podle výsledků testování se provádí zhruba 3 až 4 denně, což je v případě našeho rezervačního systému velký problém. Jako jediné schůdné řešení, které zde zbývá po vyloučení prvních dvou možností, je využití takzvaných *service accounts* (servisních účtů), které nám umožní sdílení kalendáře poněkud komplikovanějším způsobem. *Service account* je speciálním typem účtu, který disponuje stejnými právy k využití služeb Google, ale je určen pouze pro programové využití. Autorizace se tak neprovádí zadáním emailové adresy a hesla, ale pomocí vygene-

rovaných RSA klíčů a veškeré ovládání aplikací probíhá skrze programové rozhraní, nikoliv pomocí webového rozhraní[3].

Pomocí tohoto přístupu jsme schopni vytvářet pro každého uživatele aplikace vlastní kalendář v aplikaci Google Calendar, který je s uživatelem na jeho žádost sdílen. Tímto způsobem se tak odstraní problémy s pomalou synchronizací standardu iCalendar a také odstraňuje nutnost využití k přihlašování do aplikace účtu Google, čímž dojde ke splnění všech klientových požadavků.

Ilustrativní ER diagram ukládání informací potřebných k identifikaci jednotlivých událostí a kalendářů najdete na obrázku 5.8.



Obrázek 5.8: ER diagram synchronizace s aplikací Google Calendar

Platby

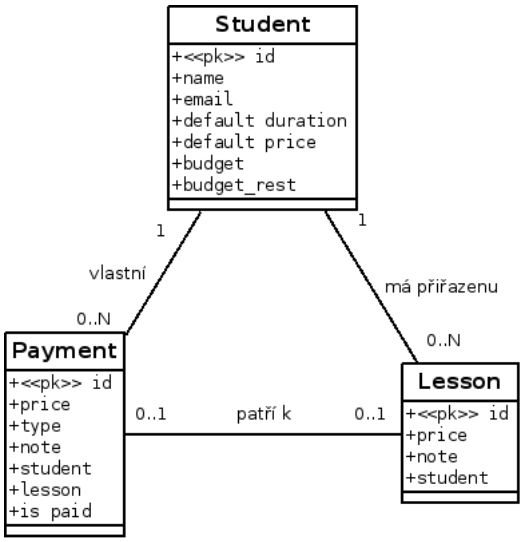
Platby jako takové jsou v systému spíše okrajová záležitost. V rámci návrhu jejich implementace, je potřeba se zaměřit především na systém transakcí a způsob ukládání dat do databáze. Z analýzy požadavků vyplývá potřeba každému studentovi zřídit virtuální konto, prostřednictvím kterého bude docházet k platbám za lekce a ostatní platby. Stav konta musí být vždy aktuální, a proto se nabízí řešení využitím agregační funkce `SUM`, která by zajistila vždy aktuální stav studentova účtu. Ačkoliv se toto řešení zdá být samozřejmostí není to úplně pravda. Využití agregační funkce `SUM`, která by při každé aktualizaci automaticky upravila stav konta v závislosti na součtu všech příjmů a výdajů sebou přináší určitá rizika, která není možné ignorovat. Jedním z těchto rizik je například ztráta dat jak způsobená chybou, tak také řízená redukce kvůli snížení paměťových nároků. V případě, že by došlo ke ztrátě dat o platbách nebo se po několika letech provozu rozhodlo o odstranění starých záznamů plateb kvůli menší datové náročnosti (nikdy nemůžeme tuto situaci dopředu vyloučit), stav studentova konta by se rapidně změnil, což není požadovaná situace. Stav studentova konta by si měl svoji hodnotu udržet i v případě, že by ze systému byly odstraněny všechny informace o platbách a proto je využití agregační funkce v tomto případě určitým rizikem.

Dalším úskalím je systém označování plateb jako *zaplaceno* nebo *nezaplaceno*. V praxi není možné nějakou platbu zaplatit napůl, platba může nabýt pouze těchto dvou stavů a je potřeba vyřešit situace, kdy student má například v systému evidovanou platbu 100 Kč, na účtu nemá žádné prostředky, tedy stav konta je −100 Kč a dobije například 50 Kč. V této chvíli je stav konta −50 Kč ale stále nedošlo k faktickému uhrazení žádné částky a informaci o skutečném množství peněz, které nepokryjí žádný výdaj tak musí být vedeno samostatně mimo informaci o stavu konta. Celý tento systém musí být propracován tak, aby v případě stornování plateb zase naopak došlo k vrácení částky a pokusu o zaplacení nějaké jiné tak, aby prostředky vedené na stínovém kontě byly vždy co nejnižší a aby v případě, že je možné nějakou platbu uhradit, byla uhrazena.

Všechny operace, které budou prováděny v rámci platebního systému, musí být prováděny jako transakce splňující čtyři základní vlastnosti známé jako *ACID*[26].

- Atomičnost (Atomicity) - operace je nedělitelný celek a buď je provedena celá nebo vůbec.
- Konzistence (Consistency) - po provedení transakce musí být systém v konzistentním stavu.
- Izolace (Isolation) - proces transakce není ovlivňován jinými transakcemi.
- Trvalost (Durability) - Provedené změny jsou trvalé.

V rámci databázových transakcí se o atomičnost, trvalost a izolaci stará systém řízení báze dat (SŘBD) a úkolem vývojáře je pouze zajištění konzistentního stavu databáze. V případě programu je ale potřeba vycházet z tohoto principu a postarat se, aby operace týkající se plateb byly zpracovány buď všechny nebo žádná aby nedošlo například k systémovému označení lekce stavem *zaplaceno* a přitom se data následkem chyby nepřenesla do databáze. Ukázkový ER diagram databáze si můžete také prohlédnout na obrázku 5.9.

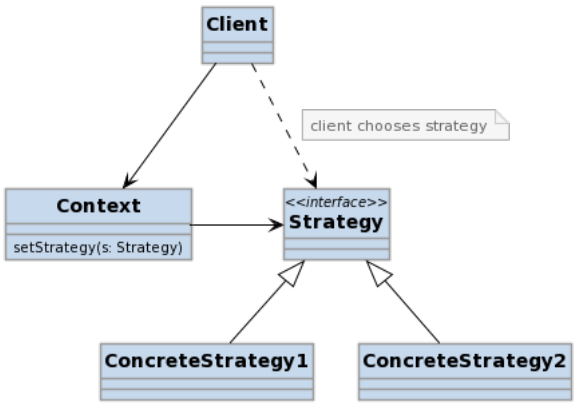


Obrázek 5.9: ER diagram plateb

Emailing

Posílání automatických emailů z webové aplikace není ničím specifickým. Vzhledem k předpokládanému poměrně malému množství emailů, které budou z aplikace odcházet, není potřeba uvažovat o využití služeb emailových serverů, ale vystačíme si s prostředky které poskytuje běžný webhosting. Požadavkem klienta byla možnost upravovat šablony jednotlivých automaticky odesílaných emailů. Z tohoto důvodu budou všechny šablony emailů vedeny prostřednictvím databáze a konkrétní informace jakými jsou například jméno studenta, čas lekce a podobně budou nahrazeny zástupnými znaky (tzv. *placeholder*), které budou za originály nahrazeny až těsně před odesláním. Právě systém nahrazování zástupných znaků originálními informacemi, je na celé části systému nejdůležitější. Vzhledem k možnosti budoucích změn je potřeba vytvořit univerzální systém, který zajistí rozšiřitelnost o libovolné zástupné znaky, jejich zpracování bude definováno nezávisle na ostatních. Z velkého množství návrhových vzorů, které by bylo vhodné pro daný případ využít, se zmíníme o dvou hlavních, na jejichž základech by měl být celý systém zpracování zástupných znaků postaven.

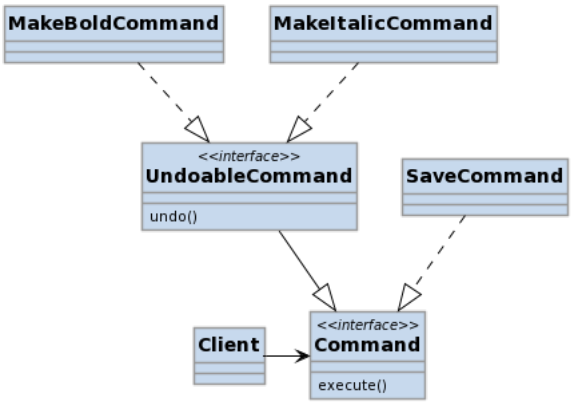
Prvním návrhovým vzorem je vzor Strategy. UML diagram tohoto návrhového vzoru si můžete podrobněji prohlédnout na obrázku 5.10. Principem tohoto návrhové vzoru je poskytnout uživateli jednotnou strategii (jednotné prostředí) která by sama v závislosti na kontextu dokázala rozhodnout o využití vhodné předem definované strategie a tu poté aplikovat.



Obrázek 5.10: Návrhový vzor strategy[16]

Druhým návrhovým vzorem je vzor Command.[15], jehož diagram si můžete prohlédnout na obrázku 5.11. Hlavním cílem tohoto návrhového vzoru bylo umožnit provádění příkazů, které byly předány hlavnímu aktérovi (hlavní třídě) pomocí instancí a hlavní třída poté zajistí jejich postupné provedení.

Právě kombinaci obou výše uvedených návrhových vzorů je vhodné použít na systém nahrazování zástupných znaků skutečnými informacemi. V rámci modulu zabývajícího se odesíláním emailů, by měla být k dispozici třída poskytující společné rozhraní pro překlad zástupného znaku na skutečnou informaci. Tato hlavní třída by měla po vzoru návrhového vzoru Strategy rozhodnout, jakou strategii pro překlad zvolí a nezávisle na uživateli zvolit vhodný způsob překladu. Jednotlivé strategie ale není vhodné napevno implementovat, je potřeba využít možnosti jednoduchého přidávání nebo odebírání strategií



Obrázek 5.11: Návrhový vzor Command [15]

po vzoru návrhového vzoru Command, kdy hlavní třída obdrží seznam instancí jednotlivých strategií.

Kapitola 6

Popis implementace

V předchozí kapitole 5 došlo k detailnímu rozebrání hlavních částí nové, modernizované aplikace z teoretického pohledu. Bylo definováno několik základních principů a pravidel jimiž je potřeba se při samotném návrhu řídit a nastíněno několik možností různých řešení problémů, které mohou nastat.

Stejně jako v předchozí kapitole bude tato kapitola rozdělena na několik hlavních kapitol podle faktických částí systému tak jak během vývoje vznikaly. V každé kapitole se stručně zaměříme na použité jazyky, technologie a návrhy jednotlivých struktur. Dojde k vyzdvihnutí důležitých částí systémů, které hrají důležitou roli v rámci celého procesu modernizace.

6.1 Původní aplikace pro správu lekcí

Jako nejvhodnější postup pro modernizaci původní aplikace bylo po konzultaci se zadavatelem práce a původním vývojářem dohodnuto, že původní aplikace zůstane z větší část v původním stavu. Aplikace by se měla zaměřit především na základní agendu lekcí a slovíček, ke které byla původně určena a všechny pokročilejší požadavky budou realizovány v samostatné aplikaci, která bude na tuto původní aplikaci napojena. Původní aplikace by se tak postupem času měla specializovat především na nástroj pro testování slovíček, protože celý systém práce se slovíčky byl v rámci aplikace velmi dobře a detaile propracován a mohl i nadále fungovat bez větších úprav či zásahů. Ostatní funkcionality jako především správa lekcí se tak z původní aplikace přesunuly do nově vzniklého systému pro správu lekcí, rezervací a plateb.

Vzhledem k tomu, že aplikace jako taková již existovala a její struktura byla detailněji rozebrána v kapitole 3 budeme se zde jen stručně věnovat úpravám, které bylo nutné provést na tomto stávajícím systému.

6.1.1 Prostředí pro synchronizaci s ostatními aplikacemi

Aby mohly nově vznikající aplikace samostatně fungovat bylo potřeba zajistit komunikaci a synchronizaci dat mezi původním a novým systémem. Pro vzájemnou komunikaci bylo potřeba vytvořit systém který umožní identifikovat oprávněné aplikace, které mohou z původní aplikace čerpat data. V tomto případě byl vytvořen jednoduchý systém postavený na komunikaci prostřednictvím *tokenů* (viz. kapitola 3.2.3), kdy v původní aplikaci dojde k vygenerování *tokenu*, který je poté použit novou aplikací pro autorizaci požadavků. Díky

použití jednoduché knihovny TinyREST[1] zajišťující veškerou komunikaci a zabezpečení prostřednictvím volání API, byla implementace tohoto bezpečnostního prvku otázkou vytvoření nového autorizačního procesu, který prostřednictvím HTTP hlavičky *Authorization* přijímá tento token a na základě jeho validace umožní aplikaci číst potřebná data.

Zároveň bylo potřeba vyřešit přihlašování do nových aplikací prostřednictvím původní aplikace, které bylo implementováno pomocí principu podobného protokolu *OAuth*, který známe například z přihlašování do různých aplikací prostřednictvím účtu u Google/Twitter nebo Facebooku. Aplikace využívá takzvaný *cross-domain local storage*[2] kdy dojde k přihlášení v původní aplikaci a skrze tento sdílený stav můžeme poté v přidružených aplikacích zjistit identitu uživatele.

6.1.2 Nové metody zkoušení slovíček

Druhou důležitou novinkou v prostředí původní aplikace bylo obohacení původního systému zkoušení slovíček o dvě interaktivnější a pro studenty zajímavější formy zkoušení slovíček. Vzhledem k tomu, že původní aplikace byla postavena na frameworku Angular 1 a serverová část na frameworku Nette Framework (viz. 3), tak při implementaci těchto novinek bylo využito již zaběhlého systému bez nějakých větších inovací.

6.2 Správa lekcí, rezervací a plateb

Na rozdíl od úprav původní aplikace, kde bylo nutné dodržovat zavedené standarty a postupy v případě systému pro správu lekcí, rezervací a plateb již nejsme tímto omezeni. Jak již v začátku této kapitoly zaznělo, celý tento systém vznikl oddělené jako samostatná aplikace z větší části nezávislá na původní aplikaci. Díky tomu byla volba programovacích jazyků, knihoven a frameworků čistě režii nové projektu.

6.2.1 Programovací jazyky a frameworky

Návrh samotné implementace, jako volba architektury nebo programovacích jazyků byla již zmíněna v kapitole zabývající se jejím návrhem 5.1.2, proto se zde zaměříme na faktické detaily samotné implementace.

Jako implementační jazyk byl na klientské části zvolen CoffeeScript a to především ze dvou důvodů. Zaprvé zde byla snaha o udržení základní kompatibility s původním systémem, který byl implementován také v CoffeeScriptu, aby v případě nějakých dalších změn bylo možné jednotlivé části systémů v případě potřeby kombinovat. Pro podporu a usnadnění vývoje byl zvolen framework *Angular 1*, který byl v době vzniku této aplikace velmi populárním a vlastně jediným kvalitním frameworkem na trhu, což spolu se snahou o udržení kompatibility s původním systémem vedlo k jeho volbě. Tento framework, který vychází z návrhového vzoru *MVC* poskytuje většinu potřebných funkcionalit pro vývoj nového systému a není tedy potřeba kombinovat jeho použití s dalšími významnými knihovnami.

Serverová část, opět stejně jako v případě původního systému byla implementována v programovacím jazyce PHP za použití frameworku *Nette Framework* (viz. kapitole PHP 2.2.3). Na rozdíl od původního systému nebyl pro komunikaci s databází zvolen *ORM* přístup z důvodů zmíněných v kapitole zabývající se analýzou původního systému v kapitole

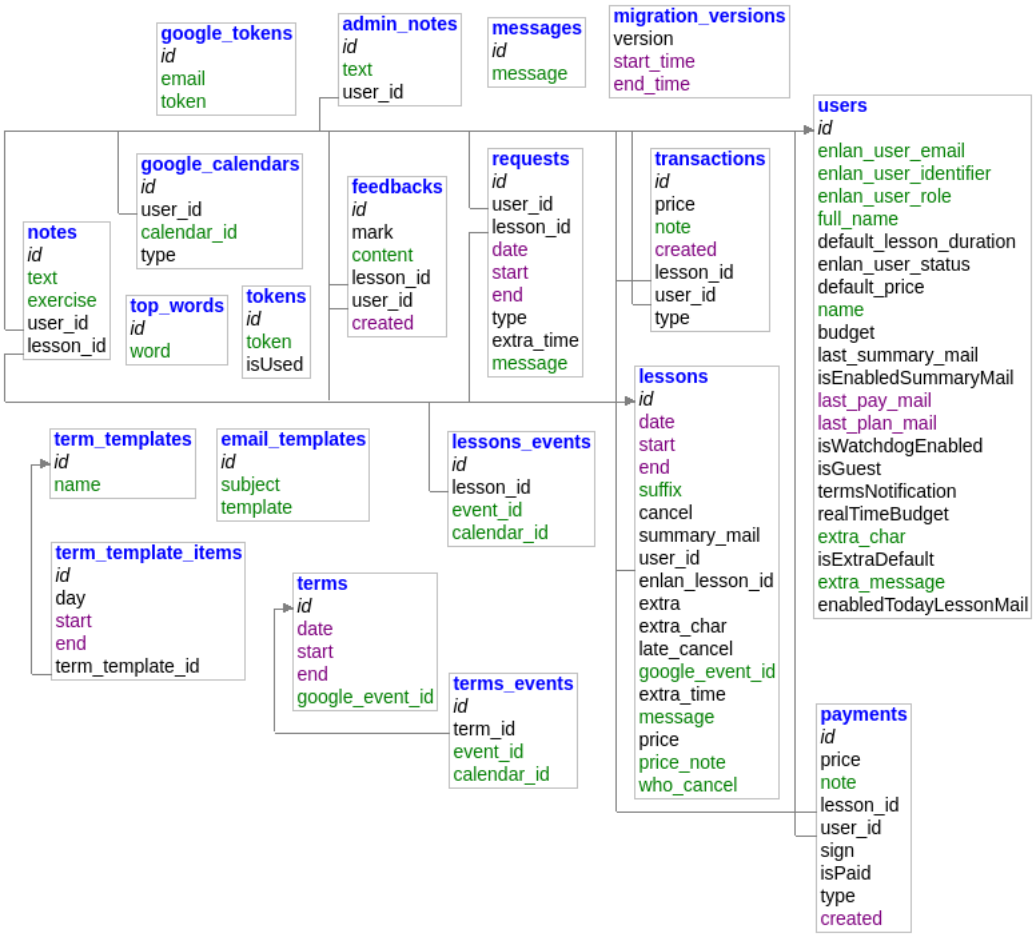
[1] https://github.com/flame-org/tinyrest
[2] https://github.com/zendesk/cross-storage

3.2.2, ale byl zvolen přímočařejší přístup k databázi pomocí lehké nadstavby v podání knihovny Dibi[3].

6.2.2 Implementační detaily

Z pohledu samotné implementace by bylo dobré vyzdvihnout pouze několik základních a důležitých bodů. Samotná komunikace mezi serverovou a klientskou částí byla implementována přesně podle návrhu v kapitole 5.1.2, tedy pomocí HTTP požadavků z klientské strany na stranu serverovou. Zabezpečení komunikace je prováděno pomocí *tokenů* s definovanou dobou platnosti a je tím zajištěna ochrana před nežádoucími situacemi.

Veškeré operace prováděné varámci správy plateb v systému se na úrovni databáze implementují pomocí transakčního zpracování tak, aby byla zajištěna maximální možná bezpečnost těchto operací. Celé schéma databáze si můžete prohlédnout na obrázku 6.1



Obrázek 6.1: Schéma databáze

Kvůli potřebě zachování synchronizovaných dat provádí aplikace automatické synchronizace s původním systémem pomocí HTTP požadavků a nástroje *cron*, který slouží k plánování úkolů v prostředí operačních systémů. Díky tomu je aplikace vždy plně sychronizována s původní aplikací a nemůže tak dojít k nekonzistentnímu stavu. Stejně tak v případě

[3]https://dibiphp.com/

správy lekcí aplikace automaticky ukládá změny do původní aplikace. Je tak možné vytvářet lekce a zadávat slovíčka prostřednictvím nové aplikace a přesto může být původní aplikace využita k jejich procvičování.

6.2.3 Zpětný pohled

Vzhledem k tomu, že vývoj tohoto systému začal již před několika lety a postupnými iteracemi se dostal až do aktuální podoby, je potřeba se trochu ohlédnout zpět a zhodnotit samotnou implementaci z odstupu. Volba programovacího jazyka CoffeeScript se zpětně může zdát jako ne úplně vhodná volba. V době, kdy aplikace vznikala a rozhodovalo se o výběru jazyka byl CoffeeScript značným favoritem především k objektovému přístupu, který v té době JavaScript postrádal, ale také kvůli dalším vymoženostem, které se sebou přinesl. Dnes by již volba jazyka s největší pravděpodobností padla na čistou implementaci pomocí JavaScriptu neboť postupem času tento jazyk získal nové konstrukce a komunita kolem něho se natolik rozrostla, že by byl jasnou volbou. Podobná situace jako s výběrem jazyka by byla i volba frameworku. V dnešní době je velkým favoritem v tvorbě JavaScritových aplikací nástroj pro budování uživatelských rozhraní s názvem ReactJS[4], který velmi rychle získal na popularitě především díky velmi preciznímu a inovativnímu přístupu k řešení problémů.

6.3 Aplikace pro interaktivní výuku

Poslední částí, která v rámci modernizace systému vznikla, je systém pro podporu interaktivní výuky realizovanoé pomocí *real-time* komunikace mezi studenty (jedním či skupinou) a učitelem prostřednictvím webového prohlížeče. Tato aplikace je v době psaní této práce vedena jako experimentální projekt (možná prototyp), aby měl klient možnost odzkoušet si tento princip výuky a v případě dalšího zájmu bude tato aplikace dále rozpracována. Z pohledu celé práce by se mohlo jednat o relativně zajímavou část a proto se v následujících kapitolách podrobněji podíváme na její samotnou implementaci.

6.3.1 Programovací jazyk a technologie

Na rozdíl od předchozích částí aplikace byl pro implementaci jak klientské tak serverové části zvolen programovací (skriptovací) jazyk JavaScript. Jak již bylo vysvětlováno v kapitole 5.1.1 důvodem pro zvolení JavaScriptu jako primárního jazyka pro implementaci obou částí byla možnost sdílení kódu mezi serverovou a klientskou částí a také kvůli podpoře technologie webových socketů zmiňované v kapitole 5.1.1.

Pro podporu samotné implementace byla na klientské části využita knihovna ReactJS, která slouží především pro tvorbu uživatelských rozhraní. Díky této knihovně je realizace celého procesu plovoucích prvků popsaných v kapitole 6.3.2 velmi jednoduchá a celý koncept je díky tomu velmi kvalitní.

Na serverové části se o interpretování JavaScriptu stará NodeJS engine, díky kterému je možné provozovat JavaScript na serverové části, což by bylo ještě v nedávné době poměrně obtížně realizováno.

[4] https://facebook.github.io/react/

6.3.2 Interaktivní spolupráce

Celý princip interaktivní výuky spočívá v určitém simulování multimediální a interaktivních tabulí jaké známe ze školního prostředí. Aplikace jako taková umožňuje uživatelům přidávat libovolné (předem definované) prvky na plochu a pak s nimi dále pracovat (pohybovat, měnit velikost, a podobně). Díky tomu má možnost učitel vkládat na obrazovku texty a obrázky a doplňovat je klasickými formulářovými prvky, které známe z jazyka HTML a z běžných webových stránek a vytvářet tak například interaktivní testy doplněné o zvukové soubory nebo další prvky.

Mezi základní prvky, které aplikace umožňuje učiteli a studentům používat patří například

- Textová pole
- Fajfka/křížek pro hodnocení studentových odpovědí
- Zaškrtávací pole
- Výběr z položek
- Textový box
- Zvukový záznam
- Obrázek

S pomocí těchto prvků a real-time synchronizace může učitel například vytvořit sadu testů, kdy všem studentům najednou zapne přehrávání audio souboru na jehož základě budou studenti odpovídat do předem připravených polí.

Implementace těchto nástrojů je díky použité knihovně ReactJS velmi snadná. Každá komponenta je reprezentována samostatnou komponentou ($React.Component()$), díky čemuž je zajištěna snadná replikace a manipulace jednotlivých prvků například posunováním po obrazovce. Veškeré tyto aktivity jsou okamžitě synchronizovány s ostatními uživateli a je tak zajištěn maximální uživatelský zážitek.

6.3.3 Real-time komunikace a protokol

Pro potřeby *real-time* komunikace jak již bylo zmíněno v předchozích kapitolách bylo využito technologie webových socketů 5.1.1. Celý proces komunikace byl implementován přesně podle návrhu, který si můžete prohlédnout v kapitole 5.1.1 na obrázku 5.3.

Kromě samotného návrhu komunikace bylo potřeba také vyřešit otázky týkající se samotného formátu (či protokolu) dat, které se budou v rámci systému přenášet. Pro potřeby přenosu informací o vytvářených nebo upravovaných, bylo využito standardního formátu *JSON*, který se pro tyto účely jeví jako vhodný. Pokud by bylo potřeba snížit datový přenos, například z důvodů velkého vytížení aplikace bylo by možné nahradit tento formát přenosem binárních dat, které je možné prostřednictvím webových socketů posílat, ale zpracování tohoto formátu by bylo výrazně složitější a prozatím to není potřeba. Struktura formátu víceméně vychází z principů knihovny ReactJS, kdy zde byla snaha o maximální kompatibilitu formátu používaného uvnitř jejich komponent a aplikací, aby bylo možné reprezentaci události ve formátu *JSON* velmi snadno převádět na existující objekt. Ukázku formátu si můžete prohlédnout na ukázkovém kódu pod tímto odstavcem.

Důležitou částí je část *event*, která definuje jednak druh události (vytváření, upravování, mazání) a element samotný. Blok *type* určuje typ elementu, pomocí kterého se na klientské straně aplikace rozhoduje jakou komponentu pro vytvoření použije a blok *attrs* definuje množinu všech argumentů které se mají komponentě předat v podobě HTML atributů. Ostatní atributy slouží k identifikaci uživatele a dalším méně důležitým věcem, které není potřeba detailněji popisovat.

```
{
    "event": {
        "type": "CREATE",
        "element": {
            "type": "TEXT_INPUT",
            "attrs": { "name": "value" },
            "prevAttrs": {}
        }
    },
    "set": 1,
    "page": 1,
    "identity": { "id": 1, "role": "teacher" }
}
```

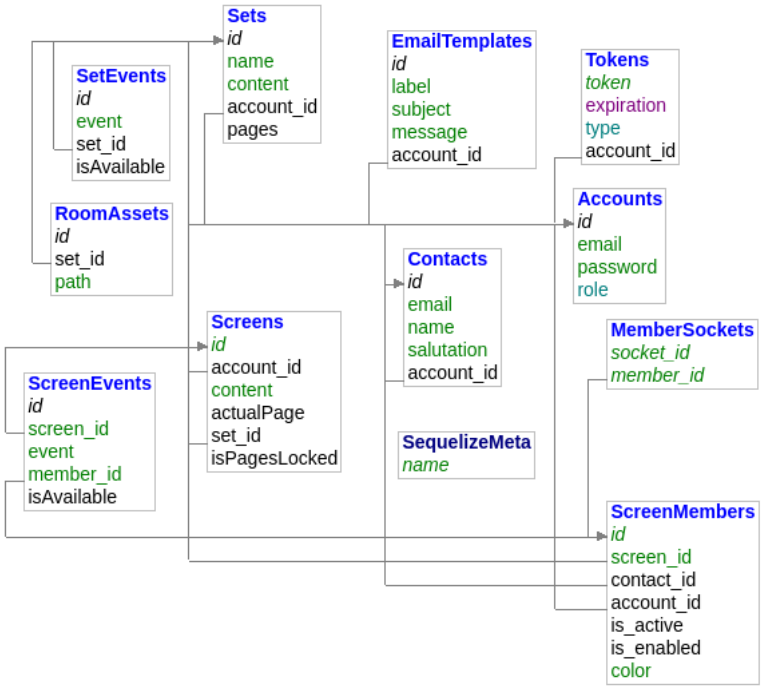
6.3.4 Databáze

Co se výběru samotné databáze týče, jak bylo již v kapitole 5.1.1 zmíněno, použita byla databáze MySQL která zajišťuje dostatečné zázemí pro provozování této aplikace. Ačkoliv by se mohlo zdát, že pro ukládání událostí ve formátu *JSON* jak popsáno v předchozí kapitole by bylo lepší využít databázový systémem, který podporuje datový typ *JSON* tak vzhledem k tomu že s daty není na úrovni databáze nijak manipulováno není tato funkce nezbytná. Kompletní schéma databáze si můžete potom prohlédnout na obrázku 6.2

6.3.5 Zpětný pohled

Stejně jako v předchozích kapitolách se i zde ohlédneme zpět za dosavadním vývojem aazhodnotíme provedené volby s odstupem času. Vzhledem k tomu, že tato část aplikace je relativně nová, není zde příliš věcí, které by bylo možné nějak vytknout. Knihovny ReactJS použitá pro tvorbu klientské části se ukázala jako velmi dobrá volba. Celý proces značně usnadnila a vzhledem k stále stoupající oblibě této knihovny nehrozí, že by v dohledné době došlo k zastavení vývoje nebo nějakým nepříznivým změnám, které by ovlivnily samotný vývoj aplikace.

Využití JavaScriptu jako primárního jazyka pro tvorbu jak klientské, tak serverové části, se ukázalo také jako dobrá volba. Opět vzhledem k stoupající popularitě a rozšiřování možností tohoto jazyka je vývoj aplikace velmi jednoduchý a výsledný produkt se zdá být dostatečně kvalitním. Ovšem narážíme zde na určité aspekty jazyka, které vývoj poněkud komplikují. Jedná se především o naprostou absenci jakékoliv typové kontroly, která by umožnila vyhnout se zbytečným problémů a nesnázím během vývoje. Vzhledem k faktu, že velké množství dat je v JavaScriptu předáváno pomocí objektů, u kterých není možné jasně definovat strukturu, je potřeba neustále validace vstupů i v případech kdy by to v případě typovaného jazyka nebylo nezbytně nutné. V dnešní době existují možnosti jak přidat



Obrázek 6.2: Schéma databáze

statickou typovou kontrolu do jazyka JavaScript což je v případě budování větších projektů rozhodně výhodou, neboť umožňují definovat struktury jednotlivých objektů včetně pokročilejší typové kontroly jakou jsou například generické datové typy. Mezi základní představitele statické typové kontroly v JavaScriptu patří projekt *flow*[5], který přidává statické typy přímo do JavaScriptu a umožňuje jejich následnou validaci, a nadstavba nad jazykem JavaScript projekt *TypeScript*[6]. Rozdíly mezi oběma nástroji nejsou veliké, přesto však můžeme především u *TypeScriptu* vyzdvihnout daleko lepší podporu napříč editory a vývojovými prostředími, která na rozdíl od *flow* mají podporu této nadstavby zabudovanou již delší dobu. Je to dáno značným rozdílem v době existence obou nástrojů. Zatím co nástroj *flow* je relativně nový, tak *TypeScript* je na trhu již velmi dlouhou dobu.

Vzhledem k těmto poznatkům získaným během vývoje a zpětným pohledem na aktuální vývoj, došlo k drobné změně implementačního jazyka oproti návrhu a pro vývoj nových částí aplikace byl místo JavaScriptu využit TypeScript jakožto jeho nadstavba. Díky tomu, že TypeScript je pouhou nadstavbou (supersetem) k JavaScriptu, je každý JavaScriptový kód zároveň i validním TypeScriptovým kódem a proto je možné oba jazyky vzájemně kombinovat. Ačkoliv to vnáší do aplikace jistý zmatek kdy některé části aplikace jsou napsány v JavaScriptu a novější pomocí TypeScriptu jedná se pouze o dočasný stav, kdy postupně postupem času dojde nahrazení původních JavaScriptových částí TypeScriptem.

[5]https://flow.org/
[6]https://www.typescriptlang.org/

Kapitola 7

Testovaní

Testování je nejen nedílnou, ale především hlavní součástí životního cyklu vývoje každé aplikace a to nejen z hlediska koncových uživatelů, ale také z pohledu samotných vývojářů softwaru.

Obecně můžeme proces testování rozdělit na několik základních metod a principů. Jedním z nejzákladnějších rozdělení testů je rozdělení podle cílové skupiny.

- **Automatické testování**, kde je kladen důraz na provádění testů za účelem ověření bezchybného chodu systému.
- **Uživatelské testování**, kde je kladen důraz na uživatelský dojem z používání aplikaci a sledování "nestandartních" operací, které nebyly zahrnuty do automatizovaného testování.

7.1 Testování aplikace pro podporu výuky

V předchozím odstavci byly nastíněny dva základní principy testování aplikací a softwaru obecně. V této kapitole se zaměříme na popis principů použitých na testování samotné aplikace pro podporu výuky, který je předmětem této práce. V následujících několika odstavcích se zaměříme na celkový koncept testování této aplikace a v dalších podkapitolách pak stručně charakterizujeme jednotlivé části aplikace a jejich přístup k testům

Protože se jedná o aplikaci, jejíž úkolem není ani tak zpracování dat nebo nějaké výpočetní operace, ale především usnadnění výuky a komunikace mezi studenty a učitelem, je zde kladen důraz především na uživatelské testování. Celý koncept testování aplikace vychází z již zaběhlého systému, kdy veškeré změny, které se v aplikaci provedou, testuje učitel spolu se studenty na speciálním testovacím serveru. Toto uživatelské testování je převážně v režii klienta (učitele), který zprostředkovává postřehy své a studentů a na jejichž základě dochází k dalším úpravám nebo opravám aplikace.

Samozřejmě se v rámci vývoje implementují také automatizované testy, které mají za úkol zajistit bezproblémové fungování aplikace a také usnadnění samotného vývoje. V rámci automatických testů je využito jak unit (jednotkových) [56] testů tak také akceptačních a integračních testů [52]. Jednotlivé typy testů nejsou na úrovni zdrojových kódů nijak odlišeny a vzájemně se kombinují a doplňují, což sebou přináší výhody v podobě snadné orientace a velmi kvalitních výstupů, ale také nevýhody v podobě velkých časových nároků spouštění testů. Bohužel vzhledem k nedostatku času a poměrně rychlému vývoji nových funkcí, není možné dostatečně pokrýt automatizovanými testy veškeré části aplikace. Prioritně je proto testování zaměřeno na kritické části systému jako práce se studentskými

konty a platbami a ostatní méně důležité funkce, které v případě chyby nemohou způsobit nějaké závažnější komplikace jsou testovány především za pomoci uživatelského testování na testovacím serveru.

7.1.1 Původní aplikace pro správu lekcí

Původní systém, který jsem přebral a na jehož základu vzniká vylepšený systém (viz. kapitola 3) postrádal jakékoliv automatické testy. Ačkoliv byl kód na celkem dobré technické úrovni, jednotlivé části systému neměli k dispozici žádné automatizované (ani jiné) testy nebo dokumentaci, kterou by se dalo při úpravách řídit. Přesto však především díky dlouholetému provozu aplikace v reálném prostředí se původnímu autorovi podařilo opravit všechny chyby a aplikace tak z pohledu uživatelů funguje bez větších problémů.

Při vývoji původní části byl kladen důraz především na uživatelské testování, kdy veškeré úpravy byly nejprve testovány učitelem a až po jejich odladění byly změny zavedeny do ostrého provozu.

Nové části, které se do původní aplikace přidávaly již z větší části obsahovaly alespoň nějakou formu automatizovaných testů, ale vzhledem k jakékoliv absenci testovací základny a mnohdy velmi obtížnému simulování některých situací, tak ani nové funkce nejsou zatím ze 100% prokryté automatizovanými testy a při testování se spoléhá především na uživatelské testování, které probíhá na odděleném serveru, kde učitel s několika vybranými studenti má možnost otestovat veškeré nové či upravené funkcionality systému tak, aby se předešlo problémům po nasazení úprav do ostrého provozu.

7.1.2 Rezervační aplikace

Nová část aplikace zaměřující se především na rezervování lekcí, platby a správu lekcí, která vzniká jako nadstavba k původnímu nevyhovujícímu systému (viz. 5) již vzniká společně s automatizovanými testy. Především kritické části aplikace, které se starají o manipulace s penězi a studentskými konty jsou téměř ze 100% pokryty automatickými testy, aby se předešlo jakýmkoliv nesrovnalostem nebo problémům s platbami, které by mohli mít negativní důsledky.

Automatickými testy je pokrytá především serverová část aplikace (*API*), které se stará o většinu operací. Naproti tomu klientská část aplikace, která obsahuje minimum aplikační logiky a zprostředkovává především uživatelské rozhraní, je testována především pomocí uživatelského testování, kde po vzoru ze zavedeného systému, kdy veškeré změny jsou aplikovány nejdříve na testovací server, kde učitel spolu se studenty testuje ovládání aplikace a dává zpět podněty k úpravám či opravám některých funkcí.

7.1.3 Systém pro interaktivní výuku

Sytém pro interaktivní výuku je nejnovější částí celého systému a je pojat spíše jako experimentální aplikace. Vzhledem k tomu že se jedná spíše o prototyp než o produkční aplikaci, jsou zde kladeny nároky především na provádění rychlých změn a z tohoto důvodu jsou automatické testy v této části aplikace spíše ojedinělé. Vzhledem k neustále se měnícím požadavkům klienta se zde zaměřujeme především na uživatelské testování, kdy osobně spolu s klientem a několika studenty testujeme ovládání aplikace a upravujeme některé prvky tak aby vyhovovaly práci na systému.

Kapitola 8

Závěr

Cílem této práce měla být kompletní modernizace a úprava stávajícího systému pro podporu výuky angličtiny. V rámci této práce jsme postupně prošli celým procesem modernizace od základních principů tvorby webových aplikací po analýzu původního systému až po návrh a samotnou realizaci úprav. Celá modernizace ve výsledku spočívala ve vytvoření dvou více méně samostatných aplikací a úprav stávajícího systému tak, aby zastřešoval jejich propojení. Důvodem pro modernizaci systému byly postupně zvyšující se nároky na původní systém, který kromě jednoduché agendy lekcí a slovíček neměl žádnou další funkcionalitu, která by výuku podpořila. Na základě klientových požadavků tak vznikl nový systém, který umožňuje vypisování volných termínů učitelem, rezervace termínů studenty, obecně správu lekcí a termínů spolu s možností automatických plateb za dané lekce či případné další služby. Přidal možnost automatického generování obsahu jednotlivých lekcí a umožnil učiteli více propojit stávající agendu lekcí s rezervačním procesem, který do té doby realizoval pouze pomocí vlastního kalendáře. Původní systém se lehce přepracoval do podoby aplikace pro vedení seznamu lekcí a slovíček, ke kterým přibyla možnost testování pomocí interaktivních testů a ostatní agenda se tak přesunula do nové části aplikace, která by v budoucnu měla původní systém zcela nahradit. V rámci úprav také vznikl jakýsi "prototyp" aplikace pro podporu interaktivní výuky prostřednictvím webového prohlížeče, kdy spolu mohou učitel a studenti "real-time" komunikovat a plnit určité úlohy, které do té doby bylo možné dělat pouze osobně.

Podle vyjádření klienta mu nová aplikace velmi usnadnila veškerou organizaci a administrativu výuky spojenou s agendou lekcí a jednotlivých rezervací studentů, kterou do té doby řešil emailem nebo po telefonu a zapisoval do kalendáře. Nová správa lekcí v rámci rezervačního systému mu umožnila vytvářet plány pro jednotlivé studenty, umožnila mu generovat souhrnné emaily se slovíčky, odkazy na zkoušení a dalšími užitečnými informacemi z každé lekce. Jednoduchá agenda plateb, umožnila nejen učiteli ale také studentům mít přehled o penězích a cenách jednotlivých lekcí, které do té doby mohli studenti získat pouze dotazem na učitele a celkově velmi zefektivnila celý proces výuky.

Druhou částí aplikace byla experimentální aplikace pro podporu interaktivní výuky, která prozatím nebyla oficiálně uvedena do provozu, ale je v době psaní této práce pouze v závěrečném testovacím režimu, kdy učitel spolu s vybranými studenty testuje jednotlivé funkce. Už teď se ale ukazuje, že aplikace umožní učiteli přesunout některé další aspekty výuky na internet, kdy budou moci studenti ať už v rámci osobního setkání nebo z pohodlí domova moci interaktivně pracovat spolu s učitelem na různých cvičeních a testech. Určitý způsob výuky byl již prostřednictvím internetu realizován a to především prostřednictvím

aplikace Skype, která ale neumožňovala pokročilou formu interakce mezi jednotlivými účastníky konverzace zatím co realizovaná aplikace ano.

Vzhledem k zadání práce, myslím můžeme říci že její cíl byl splněn. Původní aplikace prošla modernizací i když nikoliv její přímou pravou, ale cestou postupného nahrazování novou aplikací, která existuje v symbióze s aplikací původní. Veškeré klientovy požadavky na nový systém byly splněny a až na určitý technický dluh[29] jsou všechny části aplikace bez problémů použitelné a otestované skutečnými studenty v běžném provozu.

8.1 Budoucí vývoj

Celý proces modernizace stále nebyl dokončen. V rámci úprav se i nadále bude pokračovat v rozvoji interaktivního systému pro *real-time* spolupráci studentů a učitele a stejně tak i v modernizaci a postupném nahrazování původního systému novými částmi. Co se samotné implementace týče i zde je prostor pro vylepšení některých činností. Příkladem pro další rozvoj může být právě dokončení pokrytí testy tak, aby aplikace byla plně automaticky testovatelná a nemuselo se spoléhat pouze na uživatelská testování, která nejsou nikdy tak přesná jako automatické testy.

Stejně tak, jak původní aplikace, tak i některé nové části projdou renovací, protože vzhledem k velmi dlouhému vývojovému cyklu dochází k zastarávání použitých technologií a objevují se nové, které by se pro plnění konkrétních úkolů mohly hodit více a proto i nové části aplikace projdou nezbytnými úpravami.

Literatura

[1] AVMedia: *Ceník pro školy*. 2016, [Online; navštíveno 5.11.2016].
URL http://www.avmedia.cz/cs/download/nr_podzim_zima_2016.pdf

[2] B. Desruisseaux, Ed.: *Internet Calendaring and Scheduling Core Object Specification (RFC5545)*. [Online; navštíveno 7.12.2016].
URL https://tools.ietf.org/html/rfc5545

[3] Barber, I.: *Understanding Service Accounts* . 2005, [Online; navštíveno 7.12.2016].
URL http://www.riskcompletefailure.com/2015/03/understanding-service-accounts.html

[4] Bernard, B.: *Úvod do architektury MVC*. 2009, [Online; navštíveno 6.12.2016].
URL https://www.zdrojak.cz/clanky/uvod-do-architektury-mvc/

[5] BLACKDUCK: *Compare Languages*. [Online; navštíveno 3.11.2016].
URL https://www.openhub.net/languages/compare?language_name=coffeescript&measure=projects

[6] Can I Use: *XMLHttpRequest advanced features*. [Online; navštíveno 30.11.2016].
URL http://caniuse.com/#search=xmlhttprequest

[7] Center for Technology in Government: *Web environment* . [Online; navštíveno 12.9.2016].
URL http://www.thexmltoolkit.org/environment.php

[8] CoffeeScript: *Overview*. [Online; navštíveno 3.11.2016].
URL http://coffeescript.org/

[9] Curioso, A. G.: *Ajax with PHP 5* . O'Reilly Media, Inc., 2007, ISBN 9780596514037.

[10] DB-Engines: *DB-Engines Ranking*. [Online; navštíveno 3.11.2012].
URL http://db-engines.com/en/ranking

[11] Dutton, S.: *Real-time communication without plugins*. 2012, [Online; navštíveno 7.12.2016].
URL https://www.html5rocks.com/en/tutorials/webrtc/basics/

[12] ECMA International: *ECMAScript 2015*. [Online; navštíveno 2.11.2016].
URL http://www.ecma-international.org/ecma-262/6.0/

[13] ECMA International: *ECMAScript 2016*. [Online; navštíveno 2.11.2016].
URL http://www.ecma-international.org/ecma-262/7.0/

[14] Ferrara, A.: *PHP RFC: Scalar Type Declarations*. [Online; navštíveno 2.12.2016].
URL https://wiki.php.net/rfc/scalar_type_hints_v5

[15] Hordějčuk, V.: *Command (příkaz)*. [Online; navštíveno 7.12.2016].
URL http://voho.cz/wiki/command/

[16] Hordějčuk, V.: *Strategy*. [Online; navštíveno 7.12.2016].
URL http://voho.cz/wiki/strategy/

[17] Jankiewicz, J. T.: *Cross-Domain LocalStorage*. 2014, [Online; navštíveno 23.10.2016].
URL https://jcubic.wordpress.com/2014/06/20/cross-domain-localstorage/

[18] Kanisová H., Müller M.: *UML srozumitelně* . Computer Press, 2006, ISBN 8025110834.

[19] Lechocký, Z.: *Normalizace relačních databází*. 2008, [Online; navštíveno 14.11.2016].
URL http://programujte.com/clanek/2008071900-normalizace-relacnich-databazi/

[20] Malý, M.: *Web Sockets*. 2009, [Online; navštíveno 7.12.2016].
URL https://www.zdrojak.cz/clanky/web-sockets/

[21] Mozilla Developer Network and individual contributors: *IndexedDB API*. [Online; navštíveno 4.3.2016].
URL https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API

[22] Mozilla Developer Network and individual contributors: *Storage*. [Online; navštíveno 31.10.2016].
URL https://developer.mozilla.org/en-US/docs/Web/API/Storage

[23] NGINX Inc.: *Nginx*. [Online; navštíveno 11.11.2016].
URL https://www.nginx.com

[24] O'Reilly, T.: *What Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software* . MPRA, Munich Personal RePEc Archive, 2007, [Online; navštíveno 16.10.2015].
URL https://mpra.ub.uni-muenchen.de/4578/1/mpra_paper_4578.pdf

[25] Rodriguez, A.: *RESTful Web services: The basics*. 2015, [Online; navštíveno 14.11.2016].
URL https://www.ibm.com/developerworks/webservices/library/ws-restful

[26] Rouse, M.: *ACID (atomicity, consistency, isolation, and durability)*. 2006, [Online; navštíveno 25.9.2016].
URL http://searchsqlserver.techtarget.com/definition/ACID

[27] Skvorc, B.: *The Most Popular Framework of 2015*. 2015, [Online; navštíveno 5.6.2016].
URL https://www.sitepoint.com/best-php-framework-2015-sitepoint-survey-results/

[28] Stratechery LLC: *THE (ALLEGED) 13-INCH IPAD AND THE TRIUMPH OF THIN CLIENTS*. 2013, [Online; navštíveno 3.11.2016].
URL https://stratechery.com/2013/the-alleged-13-inch-ipad-and-the-triumph-of-thin-clients-finally/

[29] Tesárek, J.: *Technický dluh*. 2014, [Online; navštíveno 21.2.2017].
URL https://www.zdrojak.cz/clanky/technicky-dluh/

[30] The Apache Software Foundation: *HTTP Server Project*. [Online; navštíveno 11.11.2016].
URL https://httpd.apache.org/

[31] W3C: *HTML5*. [Online; navštíveno 1.12.2016].
URL https://www.w3.org/TR/html5/

[32] W3CSchools: *CSS3 @media Rule*. [Online; navštíveno 2.11.2016].
URL http://www.w3schools.com/cssref/css3_pr_mediaquery.asp

[33] W3Schools: *HTML Responsive Web Design*. [Online; navštíveno 3.11.2016].
URL http://www.w3schools.com/html/html_responsive.asp

[34] W3Techs: *Usage of client-side programming languages for websites*. [Online; navštíveno 1.11.2016].
URL https://w3techs.com/technologies/overview/client_side_language/all

[35] W3Techs: *Usage of server-side programming languages for websites*. [Online; navštíveno 1.11.2016].
URL https://w3techs.com/technologies/overview/programming_language/all

[36] W3Techs: *Historical trends in the usage of server-side programming languages for websites*. [Online; navštíveno 2.12.2016].
URL
https://w3techs.com/technologies/history_overview/programming_language

[37] W3Techs: *Usage statistics and market share of AngularJS for websites*. [Online; navštíveno 3.11.2016].
URL https://w3techs.com/technologies/details/js-angularjs/all/all

[38] W3Techs: *Usage statistics and market share of HTML for websites*. [Online; navštíveno 1.12.2016].
URL https://w3techs.com/technologies/details/ml-html/all/all

[39] Wikipedia: *Ajax (programming)*. [Online; navštíveno 24.10.2016].
URL https://en.wikipedia.org/wiki/Ajax_(programming)

[40] Wikipedia: *Client–server model*. [Online; navštíveno 8.7.2016].
URL https://en.wikipedia.org/wiki/Client%E2%80%93server_model

[41] Wikipedia: *HTML* . [Online; navštíveno 24.8.2016].
URL https://en.wikipedia.org/wiki/HTML

[42] Wikipedia: *Internet Explorer 11*. [Online; navštíveno 28.11.2016].
URL https://en.wikipedia.org/wiki/Internet_Explorer_11

[43] Wikipedia: *jQuery* . [Online; navštíveno 1.10.2016].
URL https://en.wikipedia.org/wiki/JQuery

[44] Wikipedia: *Single-page application*. [Online; navštíveno 24.10.2016].
URL https://en.wikipedia.org/wiki/Single-page_application

[45] Wikipedia: *Standard Generalized Markup Language*. [Online; navštíveno 24.8.2016].
URL https://en.wikipedia.org/wiki/Standard_Generalized_Markup_Language

[46] Wikipedia: *V8 (JavaScript engine)*. [Online; navštíveno 3.11.2018].
URL https://en.wikipedia.org/wiki/V8_(JavaScript_engine)

[47] Wikipedia: *Application programming interface*. [Online; navštíveno 14.11.2016].
URL https://en.wikipedia.org/wiki/Application_programming_interface

[48] Wikipedia: *Cascading Style Sheets*. [Online; navštíveno 30.10.2016].
URL https://en.wikipedia.org/wiki/Cascading_Style_Sheets

[49] Wikipedia: *Doctirne (PHP)*. [Online; navštíveno 14.11.2016].
URL https://en.wikipedia.org/wiki/Doctrine_(PHP)

[50] Wikipedia: *Document-oriented database*. [Online; navštíveno 2.10.2015].
URL https://en.wikipedia.org/wiki/Document-oriented_database

[51] Wikipedia: *History of the Internet*. [Online; navštíveno 24.11.2015].
URL https://en.wikipedia.org/wiki/History_of_the_Internet

[52] Wikipedia: *Integration testing*. [Online; navštíveno 24.03.2017].
URL https://en.wikipedia.org/wiki/Integration_testing

[53] Wikipedia: *JSON*. [Online; navštíveno 4.12.2015].
URL https://en.wikipedia.org/wiki/JSON

[54] Wikipedia: *Object-relational mapping*. [Online; navštíveno 2.10.2015].
URL https://en.wikipedia.org/wiki/Object-relational_mapping

[55] Wikipedia: *PHP*. [Online; navštíveno 1.11.2016].
URL https://en.wikipedia.org/wiki/PHP#Early_history

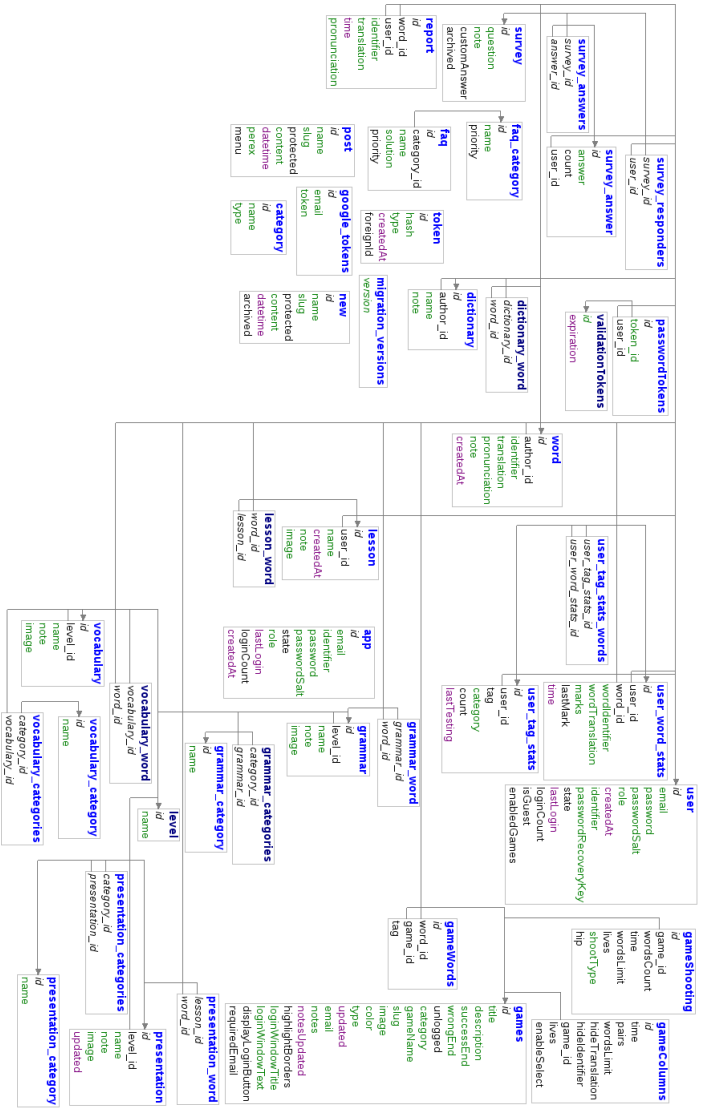
[56] Wikipedia: *Unit testing*. [Online; navštíveno 6.12.2016].
URL https://en.wikipedia.org/wiki/Unit_testing

Přílohy

Seznam příloh

Příloha A

Schéma původní databáze



Obrázek A.1: Schéma původní databáze

Příloha B

Návod na instalaci

Veškeré potřebné části, knihovny a scripty pro spuštění a otestování aplikace jsou k dispozici na přiloženém DVD (viz. příloha C). Vzhledem k velkému množství potřebných nastavení a nástrojů, které jsou nutné pro přeložení, zkompilování a spuštění všech částí aplikace je připraveno několik Docker images[1], které umožní snadné spouštění včetně konfigurací a nástrojů prostřednictvím několika málo příkazů za pomocí nástroje Docker Compose[2]. Pro bezproblémové spuštění všech částí aplikace je zapotřebí mít nainstalované aplikace Docker 17.03.1-ce a Docker Compose 1.9.0, build 2585387. Je silně doporučeno pro spuštění využít operační systém Linux neboť na operačním systému Windows nebylo sestavení pomocí Dockeru testováno.

B.1 Požadavky

- Aplikace Docker 17.03.1-ce
- Aplicace Docker Compose 1.9.0
- Operační systém Linux (testováno na Fedora 25)
- Internetové připojení
- Ncat: Version 7.40 (https://nmap.org/ncat)

B.2 Postup instalace

Toto je doporučený postup pro spuštění všech součástí aplikace pomocí předpřipravené konfigurace webového serveru a všech potřených nástrojů a knihoven. Jednotlivé aplikace je samozřejmě možné znovu přeložit pomocí instalačních skritpů uložených ve složkách `pacakges/<name>/scripts` ale vzhledem k velkému množství nástrojů (Bower, Gulp, Grunt, NodeJS, NPM, yarn, Composer), které je potřeba spustit to není doporučeno.

Upozornění: Balíček `packages/api.booking.enlan.eu` a `packages/api.enlan.eu` není možné dodatečně aktualizovat pomocí nástroje Composer, vzhledem k potřebě stahování privátních repozitářů, ke kterým v rámci této práce nemůže být umožněn přístup. Všechny potřebné knihovny, které jsou součástí této práce jsou dopředu nainstalovány a k dispozici ve složkách `vendor` a `libs` tak, aby aplikace bylo možné spustit.

[1] https://www.docker.com/
[2] https://docs.docker.com/compose/

1. Zkopírování složky `source_code` obsahující zdrojové soubory na lokální disk počítače (Aplikace potřebují zapisovat na disk a to není na DVD možné).
2. Vytvoření a nastavení *Servise account* účtu na Google API Console (potřeba pro synchronizaci s aplikací Google Calendar, bez tohoto kroku nebudou aplikace fungovat) podle návodu B.2.1
3. Zkopírování získaného certifikátu na umístění `packages/api.booking.enlan.eu/app/config/certs/google-api-key.local.p12`
4. Upravení informací `google.clientId` a `google.email` v souboru `packages/api.booking.enlan.eu/app/config/config.local.neon` tak aby odpovídali údajům z vytvořeného *Service account*
5. Nastavení `pwd` do složky `source_code`. Příkaz `cd /some/path/to/source_code`
6. Překompilovat Dockeriles pomocí příkazu `sh ./scripts/build.sh`
7. Spuštění všech částí aplikace pomocí příkazu `sh ./scripts/start.sh`
8. Zkopírování vygenerovaného seznamu IP adres a doménových jmen do lokálního souboru `/etc/hosts` aby systém mohl správně načítat URL adresy.
9. Po přihlášení do aplikace na adrese `booking.enlan.dev` je nutné spustit synchronizaci pomocí modrého tlačítka s obrázkem symbolizujícím aktualizaci vedle nápisu `Booking` viz. B.4 (přihlašovací údaje viz. B.3). **Je nutné počkat na dokončení aktualizace, pokud zmizí informace o načítání stejně vyčkejte na automatické přenačtení stránky a zobrazení zelené zprávy informující o dokončení.**
10. Spuštěné aplikace se doporučuje vypnout pomocí připraveného scriptu `sh ./scripts/stop.sh`
11. (Volitelné) Po ukončení aplikace doporučuji vymazat vytvořené soubory pomocí příkazu `docker rm $(docker ps -aq)`

B.2.1 Vytvoření účtu Google API Console

Návod je platný ke dni 5.5.2017

1. (Volitelný) Vytvoření klasického Google (email) účtu
2. Přihlášení do Google pomocí vašeho účtu
3. Otevřít stránku `https://console.developers.google.com/cloud-resource-manager`
4. Kliknout na `CREATE PROJECT`
5. Vyplnit požadované údaje a **počkejte** na úspěšné vytvoření projektu.
6. Přejděte na stránku `https://console.developers.google.com/apis/`
7. Ověřte si že máte vybraný projekt vytvořený v předchozím kroku a vyhledejte službu `Google calendar API` a klikněte na tlačítko `Enable`

8. V menu na levé straně vybrat položku `Credentials` a v otevřené stránce kliknout na `Create redentials` a vybrat položku `Service account key`
9. Vyplnit požadované údaje (bez role) a vybrat `Key type = P12`.
10. Uložit vygenerovaný klíč do aplikace (viz. Postup instalace B.2)
11. Na stránce `Credentials` kliknout na tlačítko `Manage service accounts` nad seznamem vytvořených účtů a zkopírovat `Service account id` jako `google.email` a `Key ID` jako `google.clinetId` do aplikace (opět viz. B.2.

B.3 Přihlašovací údaje

Jednotlivé aplikace mají vytvořeny základní demonstrační účty. Informace o přihlašovacích jménech jsou k dispozici na DVD v souboru `source_code/README.md`.

B.4 Důležité informace

- Synchronizace aplikací `booking.enlan.dev` a `enlan.dev` (tedy, nové a staré aplikace) se provádí v rozhraní aplikace `booking.enlan.dev` pomocí modrého tlačítka s obrázkem symbolizujícím aktualizaci vedle nápisu `Booking`. Tato ruční synchronizace je nutné vzhledem k absenci automatického `cornu`, které není v rámci testovací aplikace možné použít.
- Jednotlivé aplikace během svého běhu odesílají uživatelům emaily. Vzhledem k tomu, že se jedná o testovací prostředí nejsou emaily fakticky nikam odesílány, ale jsou automaticky zachytávány a mohou být zobrazeny pomocí fiktivního emailového klienta, který je po spuštění aplikace dostupný na adrese `maildev.enlan.dev`

Příloha C

Přiložené DVD

C.1 Struktura DVD

V bodech popsána struktura DVD

- Složka `source_code` obsahuje všechny zdrojové soubory, které jsou potřebné ke spuštění aplikace (viz. B)
- Složka `docs` obsahuje zdrojové soubory a podklady pro přeložení textu této práce do PDF souboru.

C.2 Organizace zdrojových souborů a autorství

Vzhledem k tomu, že tato práce staví na původní již existující aplikaci, které nejsem autorem, ale je její přítomnost je nutná pro spuštění a otestování v rámci této práce, přiložené DVD obsahuje i původní aplikaci. Veškeré soubory, kterých jsem autorem výhradně já, jsou opatřeny komentářem na začátku každého souboru obsahující řetězec `(c) Ondřej Záruba <email>`.

Zdrojové kódy všech částí aplikace jsou k dispozici ve složce `source_code/packages`, kdy každá složka znamená samostatnou aplikaci. Jedinou výjimkou jsou moduly do původní aplikace, které se nacházejí ve složce
`source_code/packages/api.enlan.eu/libs/enlan/<module>`.
Většina modulů je dílem původního autora, ale některé moduly obsahují úpravy nebo nové prvky. Tyto soubory jsou také opatřeny identifikujícím komentářem. Ostatní soubory ve složce `source_code` na přiloženém DVD jsou pro snadné spuštění a testování v rámci této práce.