VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

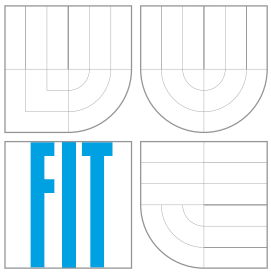
WEBOVÁ APLIKACE K ROZVOJI SLOVNÍ ZÁSOBY A KONVERZAČNÍCH SCHOPNOSTÍ VE VÍCE JAZYCÍCH

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE JAN JAKEŠ
AUTHOR

BRNO 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

WEBOVÁ APLIKACE K ROZVOJI SLOVNÍ ZÁSOBY A KONVERZAČNÍCH SCHOPNOSTÍ VE VÍCE JAZYCÍCH

WEB APPLICATION FOR IMPROVEMENT OF VOCABULARY

AND CONVERSATION SKILLS IN MULTIPLE LANGUAGES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JAN JAKEŠ
AUTHOR

VEDOUCÍ PRÁCE — Ing. IGOR SZŐKE, Ph.D.
SUPERVISOR

BRNO 2011

Abstrakt

Cílem této práce je vytvoření webové aplikace, která má uživateli usnadnit rozvoj slovní zásoby a konverzačních schopností při studiu cizích jazyků. Důležitá je orientace na studium několika jazyků současně. Základem systému je jazyk PHP a knihovna Nette Framework, dále jsou využity zejména technologie MySQL, Doctrine 2, jQuery, ExtJS a programová rozhraní aplikací Wordnik a Facebook. Tato práce popisuje kompletní vývojový cyklus aplikace od specifikace a analýzy požadavků přes návrh architektury a uživatelského rozhraní až po detaily implementace, průběh testování a jeho výsledky. Nakonec jsou zhodnoceny náměty na budoucí vývoj.

Abstract

The aim of this work is to create a web application that would help its users to improve their vocabulary and conversation skills in foreign language studies. An important point is an orientation on the study of several languages at the same time. The base of the system is PHP programming language and the Nette Framework library. Other technologies used are MySQL, Doctrine 2, jQuery, ExtJS and programming interfaces of Wordnik and Facebook applications. This text describes the complete process of the development from the specification and analysis of application demands over architecture and user interface design up to implementation details, the process of testing and its results. Finally it sums up the ideas for future development.

Klíčová slova

studium cizích jazyků, vícejazyčnost, výuka, kartičková metoda, flashcards, web, Wordnik API, Facebook API, PHP, Nette Framework, Doctrine 2, jQuery, ExtJS

Keywords

foreign language studies, multilinguality, learning, flashcards, web, Wordnik API, Facebook API, PHP, Nette Framework, Doctrine 2, jQuery, ExtJS

Citace

Jan Jakeš: Webová aplikace k rozvoji slovní zásoby a konverzačních schopností ve více jazycích, bakalářská práce, Brno, FIT VUT v Brně, 2011

Webová aplikace k rozvoji slovní zásoby a konverzačních schopností ve více jazycích

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Igora Szőkeho, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jan Jakeš
18. května 2011

Poděkování

Rád bych zde poděkoval vedoucímu Ing. Igorovi Szőkemu, Ph.D. za odborné vedení práce, za užitečné rady a za významnou pomoc při shánění testovacích dat. Za poskytnutí testovacích dat děkuji společnosti Lingea, s.r.o.

Obsah

Kapitola 1

Úvod

V několika posledních desetiletích se zejména díky rapidnímu vývoji informačních a komunikačních technologií začaly jednotlivé části světa vzájemně otevírat a studium cizích jazyků se stává pro mnoho lidí nezbytným. Troufám si tvrdit, že bez základních znalostí alespoň jednoho cizího jazyka se dnes obejde opravdu málokdo. V zemích, které mají jako mateřský jazyk některý z jazyků minoritních, se navíc stává běžným standardem studium dvou a více cizích jazyků.

Studium každého dalšího jazyka ale vyžaduje nemalé množství času a značný objem úsilí, které musíme pravidelně vynaložit, abychom své jazykové schopnosti prohlubovali anebo alespoň udržovali na určité úrovni. Studovat více jazyků zároveň od samých začátků zřejmě není nejlepší způsob učení, ale ve chvíli, kdy si student osvojí základní rysy jazyka, naučí se gramatiku a jazyk umí přirozeně používat, zůstává hlavním předmětem studia *slovní zásoba* a z ní vyplývající *konverzační schopnosti*. Tyto dovednosti jsou zásadní pro komunikaci v daném jazyce a pro pokročilého studenta by nemělo být problémem je rozvíjet a procvičovat ve více jazycích najednou a utřídit si tak slovní zásobu napříč všemi jazyky, jejichž studiu se věnuje.

Osobně nyní studuji tři cizí jazyky a z vlastních zkušeností vím, že pokud je člověk zaneprázdněn školou či prací, hledá čas a prostor na procvičování uvedených jazykových dovedností velmi obtížně. Přitom kdybychom si každý den efektivně procvičili a zopakovali několik desítek pojmů, ztratíme minimum času a našim znalostem to výrazně prospěje. Za těmito problémy jistě z části stojí přirozená lidská lenost, ale také značná dezorganizace slovní zásoby uchovávané v papírové formě. Mezi mnoha sešity člověk velice obtížně získává přehled o tom, které pojmy zvládá dobře a které by měl naopak více procvičovat.

Cílem této aplikace je přinést uživateli co nejsnazší možnost procvičování jazykových dovedností a to ve všech jím studovaných jazycích zároveň. Aplikace dále bude nabízet inteligentní výběr termínů, které je potřeba opakovat, přehled o rozsahu vlastní slovní zásoby a v neposlední řadě také motivaci ke studiu v podobě přehledu výsledků studia předchozího.

Tato práce je nejen spojením mého zájmu o webové technologie se zálibou v učení cizích jazyků, ale jsem přesvědčen o tom, že může být užitečná pro mnoho dalších uživatelů internetu.

Kapitoly 2 a 3 se postupně věnují specifikaci a analýze požadavků na vyvíjený systém. Podrobný návrh architektury a uživatelského rozhraní aplikace je popsán v kapitolách 4 a 5, kapitola 6 pak popisuje použité technologie a v kapitole 7 je popsán proces implementace. Způsobu nasazení aplikace na produkční server se věnuji v kapitole 8, kapitola 9 popisuje způsob testování s pomocí uživatelů a zhodnocuje výsledky testů a v kapitole 10 je naznačen směr budoucího vývoje aplikace.

Kapitola 2

Specifikace požadavků

V této kapitole vymezím základní cíle této aplikace a stanovím nejdůležitější body, které by měla splňovat. Bude se jednat především o neformální popis požadavků na aplikaci z pohledu uživatele. Při stanovení základních cílů vycházím především z vlastních zkušeností se studiem a výukovým software a z osobní představy, jak by měl ideální systém pro snadnou výuku více jazyků vypadat. Přesnější podobu aplikace přiblížím v kapitole 3, konkrétnímu návrhu architektury systému se pak budu věnovat v kapitole 4.

2.1 Forma aplikace

Jak název práce napovídá, aplikace má být implementována v podobě webu. Tato podoba nabízí oproti desktopové aplikaci několik výhod. Především jde o možnost centralizace uživatelských dat a statistik výuky na straně serveru a jejich přístupnost z libovolné části světa, ale zajímavá je také možnost komunikace či propojení s jinými online systémy prostřednictvím API[1].

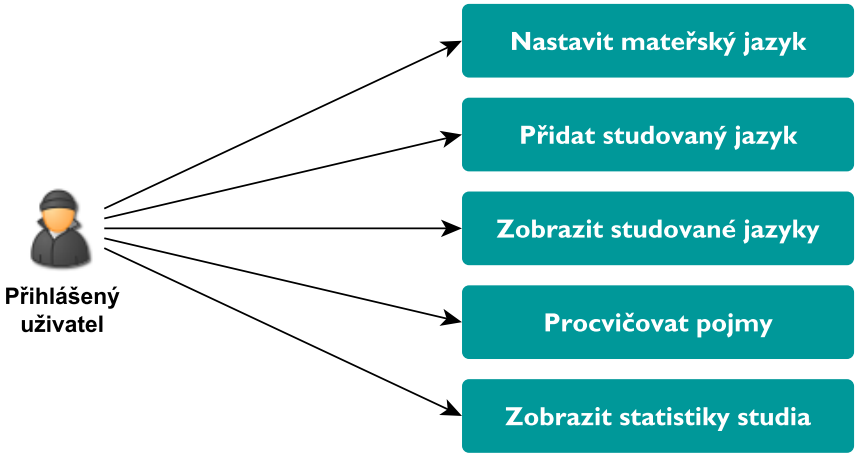
Aplikace bude mít podobu služby portálového typu, která se bude skládat ze dvou základních částí:

1. *Výuka* je nejdůležitější částí aplikace. Základní požadavky na její podobu stanovuje zadání této práce. Výuková část bude přístupná pouze přihlášeným uživatelům a bude umožňovat uživatelsky přívětivé procvičování vybraných pojmů. Pojmy bude možné trénovat buď čistě samostatně anebo v kontextu krátkých konverzačních frází. Systém bude schopen automaticky vybírat takové pojmy, které budou určitým způsobem vhodné k procvičení. Například to mohou být pojmy, ve kterých uživatel často chybuje, anebo pojmy, které procvičoval málo. Dále zde bude uživateli nabídnut výběr jazyků ke studiu a možnost prohlížení statistik studia minulého.

2. *Databáze pojmů* bude speciální multijazyčný slovník, který bude skladovat jednotlivá slova a jejich překlady v různých jazycích. Tato část bude přístupná i bez přihlášení do systému. Databáze musí být navržena tak, aby umožňovala vzájemně mapovat sobě odpovídající výrazy v libovolných jazycích. V systému bude sloužit k tomu, aby si každý nový uživatel nemusel vytvářet vlastní slovník s překlady. Více se této části věnuje sekce 2.3.

Pojítkem mezi oběma uvedenými částmi aplikace bude slovní zásoba uživatele. Každý uživatel bude mít v systému svůj vlastní slovník, do kterého bude schopen přidávat pojmy

[1] Application Programming Interface – prostředek pro programovou komunikaci mezi aplikacemi.



Obrázek 2.1: Diagram případů užití – výuková část aplikace

ze zmíněné centrální databáze pojmů. Díky tomu bude mít přehled o své aktuální slovní zásobě, což by měl být mimo jiné údaj motivující ke studiu. Slovní zásoba každého uživatele tedy bude podmnožinou centrální databáze systému.

Uživatelské rozhraní by mělo být především co nejjednodušší a mělo by umožňovat rychlý přístup ke studiu bez složitého nastavování a přípravy. Navíc by mělo být interaktivní, dynamické a zároveň použitelné ve všech dnešních majoritních prohlížečích.

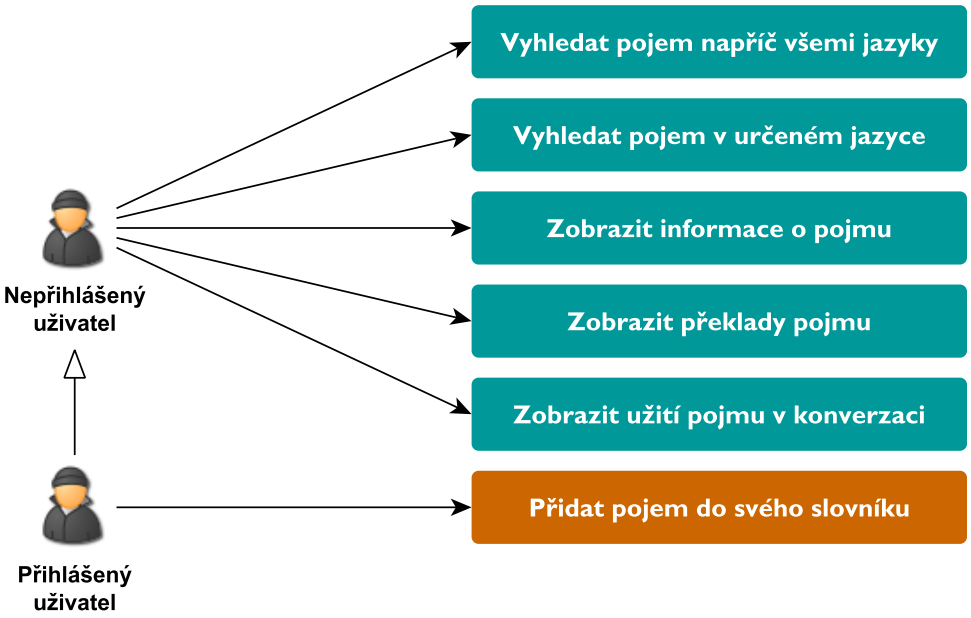
2.2 Výuka

Hlavním kritériem pro podobu výuky z pohledu studenta by měla být *jednoduchost*. Aplikace má uživateli studium maximálně usnadnit a ušetřit tak co nejvíce času. Mělo by tedy být možné přejít k výuce ihned po přihlášení do systému a také bez problémů výuku kdykoliv přerušit. Uživatel by měl být schopen procvičit si i několik desítek pojmů během krátké chvíle. Samotné procvičování pojmů bude vycházet z tzv. *kartičkové metody*, kterou podrobně popíši v části 3.2.

Základní vymezené požadavky na výukovou část aplikace shrnuje diagram případů užití na obrázku 2.1.

2.3 Databáze pojmů

Jak již bylo uvedeno v sekci 2.1, aplikace by měla obsahovat vlastní databázi pojmů a konverzačních frází. Důležité je zvážit, jak aplikace bude dané pojmy získávat, jaké informace uchovávat a jak zajistit jejich kvalitu. Od počátků vymezování požadavků na systém jsem považoval za nevhodné, aby museli uživatelé všechny pojmy do systému sami ručně zadávat. Sice to může pomoci k jejich zapamatování, ale zejména při práci s více jazyky je to pro mnoho uživatelů příliš časově náročné a může to odrazovat od studia. Databáze by navíc měla být schopna uchovávat základní statistické informace získané od uživatelů aplikace v průběhu výuky.



Obrázek 2.2: Diagram případů užití – centrální databáze pojmů

Základní požadavky na databázi pojmů z pohledu uživatele jsem opět shrnul v podobě diagramu případů užití na obrázku 2.2. Nepřihlášený uživatel si tedy bude smět libovolně procházet obsah této databáze a uživatel, který je v systému zaregistrován, si zde bude schopen přidávat jednotlivé pojmy do svého uživatelského slovníku.

Jelikož se jedná o specifikaci požadavků na systém, nezahrnuje uvedený diagram další případy užití, které později vyplynuly z podrobnější analýzy požadavků a navržené struktury databáze. Znázorněné případy užití jsou tedy požadované, ale výsledný systém jich implementuje více.

Kapitola 3

Analýza požadavků

Tato kapitola podrobněji analyzuje požadavky stanovené v kapitole 2 a slovním popisem přesněji vymezuje budoucí podobu aplikace. Nejprve se věnuji nalezení vhodné a jednoduché výukové metody, dále je pak popsáno použití této metody v aplikaci a způsob výběru pojmů k procvičování. Závěr kapitoly analyzuje podobu statistik výuky a databáze pojmů.

3.1 Metoda výuky

Jak již bylo řečeno v sekci 2.2, rozhodl jsem se výuku pojmů založit na tzv. *kartičkové metodě*[1]. Jedná se o jednoduchou metodu slovního drilu, která se používá nejen při výuce jazyků, ale i v mnoha dalších oborech. Její princip spočívá ve vytvoření malých kartiček s jednotlivými pojmy. Na jedné straně kartičky máme pojem v mateřském jazyce, na druhé straně jeho překlad v jazyce studovaném. Před otočením kartičky se student zamyslí nad uvedeným výrazem a pak zhodnotí, zda odpověděl správně. Princip této metody a její výhody stručně shrnuje článek [14].

3.1.1 Volba učební metody

Kartičkovou metodu jsem zvolil nejen proto, že ji považuji za efektivní, ale také proto, že je vhodná právě pro trénování slovní zásoby. Další metody, jejichž přehled najdeme na anglické Wikipedii [23], navíc často nejsou plně realizovatelné pomocí software. Zatímco metoda PQRST[2] vyžaduje vynaložení nemalého úsilí od studenta, tak dnes značně populární *sugestopedie* [21] a jí příbuzné metody nejsou realizovatelné bez velmi zkušeného lektora.

Uvedené metody obvykle nepracují pouze s textem či slovíčky, ale podněcují i další lidské vjemy, což zlepšuje efektivitu procesu zapamatování. Rozhodl jsem se tedy pro tento systém rozšířit zvolenou kartičkovou metodu o *možnost zobrazení fotografií a obrázků*, které souvisí s trénovanými pojmy.

3.1.2 Existující aplikace

Na zvolené metodě je postaveno několik existujících aplikací. Sám jsem v minulosti některé používal a další jsem zkoumal pro přehled a inspiraci k této práci. Seznam některých aplikací založených na metodě flashcards nalezneme na anglické Wikipedii [24]. Později v průběhu

[1] Někdy též *cedulkový systém*, v angličtině *flashcards*.
[2] Z anglických slov preview, question, read, summary, test.

implementace systému vyšel zajímavý článek [5], který porovnává tři aplikace využívající této učební metody.

Osobně mě nejvíce zaujaly a inspirovaly programy Memostation[3] a Keepinhead[4]. Oproti běžnému použití kartičkové metody se tento systém bude lišit především v tom, že uvedenou metodu rozšíří na libovolný počet jazyků. Data navíc budou ukládána do centrální databáze pojmů, aby uživatel nemusel hledat či tvořit podklady ke studiu sám.

3.1.3 Rozložení výuky v čase

Zmíněný program Memostation byl pro mě zajímavou inspirací v metodologii výuky. Autoři Memostation ve svém článku o učebních metodách [13] vysvětlují princip tzv. *rozloženého učení*[5], který vychází ze způsobu fungování lidské paměti. Tento princip říká, že studované pojmy je potřeba opakovat po určitých časových intervalech, které se s počtem opakování daného pojmu mají postupně prodlužovat.

Stejný zdroj vysvětluje i tzv. *Leitnerův způsob* učení, který definuje rozdělení kartiček s pojmy do několika různě velkých boxů a student pak kartičky přesunuje mezi boxy podle toho, jak byl úspěšný při odpovědi na tázaný pojem. Tato metoda má přirozeně odrážet potřebu opakovat jednotlivé pojmy ve stále se prodlužujících intervalech.

Zmíněné metody ale vyžadují časový plán a především jeho dodržování z pohledu uživatele. Jelikož jsem v požadavcích na systém stanovil, že uživatel má mít možnost trénovat pojmy v podstatě kdykoliv a v libovolně dlouhých časových intervalech, rozhodl jsem se navrhnout tři způsoby výběru pojmů k procvičení, které tyto myšlenky odrážejí, ale na časovém plánu nejsou závislé. Tyto metody podrobněji vysvětlím v sekci 3.3.

3.2 Způsob výuky

Při úvahách nad konkrétním zpracováním kartičkové metody v této aplikaci jsem se rozhodl, že systém bude umožňovat trénink pojmů ve třech různých formách. Dvě z nich jsou zaměřeny především na dril slovní zásoby, třetí je cílena na užití pojmů v kontextu krátkých konverzačních frází.

1. *Jednoduchá výuka (simple practise)* se nejvíce podobá běžné kartičkové metodě. Systém při výuce předloží vybraný pojem v mateřském jazyce uživatele a ten bude moci postupně zobrazit odpovědi v jednotlivých jazycích, které si zvolil ke studiu. Poté uživatel zhodnotí správnost své odpovědi a bude mít možnost přejít na další pojem.

2. *Psaná výuka (typing practise)* bude po uživateli vyžadovat odpověď zapsáním správného pojmu do textového pole. Systém pak jeho odpověď ohodnotí automaticky. Pokud má určitý pojem více překladů, systém uživateli schválí jakoukoliv správnou odpověď a poté mu ukáže i další možné varianty. Jelikož při zápisu do textového pole mohou vznikat zbytečné chyby na základě detailů a překlepů, bude systém u textového pole napovídat z kompletní databáze. Uživatel tedy s jistotou zadá pojem, který v databázi existuje.

3. *Konverzační výuka (conversation practise)* zobrazí uživateli pojem v mateřském jazyce a v cílových jazycích vybere konverzační frázi, ve které je tento pojem použit.

[3] http://www.memostation.net/
[4] http://www.keepinhead.com
[5] V angličtině *spaced repetition*.

Uživateli bude předložena konverzační fráze, ze které bude použitý pojem vypuštěn. Student pak zobrazí správnou odpověď a ohodnotí své znalosti. Tento bod mimo jiné vyžaduje, aby databáze byla schopna uchovávat informaci, kde se v dané frázi použitý pojem nachází.

3.3 Způsob výběru pojmů k procvičení

Jak jsem zmínil již v sekci 3.1, procvičované pojmy je potřeba opakovat, ideálně v prodlužujících se intervalech. Uvedená sekce popisuje metodiku rozložení výuky v čase a vysvětluje, proč jsem se rozhodl navrhnout vlastní řešení, které ze zmíněných metod vychází. Při úvahách, na základě čeho vybírat z databáze pojmy, které budou uživateli předloženy ke studiu, jsem dospěl ke třem variantám, které považuji za zajímavé:

1. Pojmy, ve kterých uživatel často dělá chyby.
2. Pojmy, které uživatel dlouho neprocvičoval.
3. Pojmy, které uživatel procvičoval málo.

Rozhodl jsem se, že naimplementuji všechna uvedená řešení a uživatel systému bude mít možnost si mezi nimi vybírat. Všechny tři varianty budou stanovenou skupinu pojmů upřednostňovat takovým způsobem, že pojem, který nejlépe splňuje dané kritérium výběru, bude mít vyšší pravděpodobnost výskytu než pojmy, které tomuto kritériu vyhovují méně.

Na rozdíl od principu rozloženého učení, uvedeného v sekci 3.1, nebude tento systém plánovat čas příštího výskytu pojmu, ale při příštím výběru z databáze ho upřednostní s určitou pravděpodobností. Ohodnocení kritéria pro pravděpodobnost výskytu rozdělí při každém výběru z databáze pojmy do několika skupin, v rámci nichž se pojem vybere náhodně. Tyto skupiny se podobají rozdělení pojmů do boxů podle v sekci 3.1 zmíněného Lietnerova způsobu učení. Podrobnostem implementace se budu věnovat v sekci 7.4.

3.4 Zpracování statistik

Při návrhu zpracování statistik výuky uživatelů systému bylo nejprve nutné stanovit, co se bude na statistikách zobrazovat. Jelikož je výuka založena na systému správných a špatných odpovědí, lze o každém pojmu uchovávat následující údaje:

- Kolikrát uživatel odpověděl správně a kolikrát špatně.
- Kdy uživatel daný pojem procvičoval.

Z těchto údajů by měl být systém schopen zobrazovat statistiky, které by uživateli měly sdělit zejména:

- Kolik pojmů bylo procvičeno za určitou časovou jednotku.
- Kolik z procvičených pojmů bylo zodpovězeno správně a kolik špatně.
- Počet pojmů v uživatelském slovníku v závislosti na čase, který by tak měl odrážet rychlost rozšiřování slovní zásoby.

Rozhodl jsem se tuto statistiku zobrazit pomocí dvou různých grafů, z nichž jeden bude znázorňovat první dva uvedené body a druhý růst slovní zásoby. Tato dvojice grafů bude schopna podle volby uživatele zobrazovat jednotlivé statistiky v časových intervalech:

- týden,
- měsíc,
- rok,
- celková statistika.

Návrh databáze, který bude součástí kapitoly 4 tedy musí být připraven tak, aby bylo možné získávat všechny výše stanovené kombinace statistických údajů.

3.5 Obsah databáze pojmů

Základní body, které musí databáze splňovat, byly již stanoveny při specifikaci požadavků v sekci 2.3. Jelikož je systém zaměřen na výuku každého konkrétního pojmu ve více jazycích, musí být databáze stavěna tak, aby byla schopna vázat výrazy z různých jazyků podle jejich *významu*.

Data strukturovaná tímto způsobem nelze automaticky získávat tak, abychom dosáhli stoprocentně správných výsledků, protože běžné slovníky takto stavěny nejsou. Z tohoto důvodu jsem se rozhodl využít síly uživatelské komunity a částečně Wiki[6] způsobu tvorby obsahu a stanovil, že do centrální databáze budou moci přidávat pojmy přímo uživatelé systému.

3.5.1 Definice pojmů

K rozlišení významu pojmů je ovšem potřeba každý pojem definovat, což by pro uživatele bylo nelehké až nemožné. Jako základ rozlišení významu jednotlivých pojmů jsem se tedy rozhodl použít některý z volně dostupných výkladových slovníků. Rozhodoval jsem se mezi systémy Wordnet[7] a Wiktionary[8], z nichž druhý má pravděpodobně o něco aktuálnější a rozsáhlejší databázi. Při vyhledávání potřebných informací jsem objevil projekt Wordnik[9], který kumuluje data z šesti velkých anglických výkladových slovníků, včetně dvou výše uvedených, a navíc nabízí vyspělé API pro přístup ke své databázi.

Rozhodl jsem se tedy propojit tuto aplikaci se systémem Wordnik a využít definic, které pocházejí z projektu Wiktionary. Pokud uživatel nenajde požadovaný pojem v systémové databázi, bude moci vyhledat jeho anglický význam, a pokud systém zjistí, že tento pojem nezná ani v angličtině, vyhledá všechny jeho významy přes Wordnik API, přiřadí k nim anglický překlad a nabídne je uživateli. Z pohledu uživatele tedy bude nerozlišitelné, zda byly pojmy právě načteny z lokální anebo externí databáze. K nalezeným pojmům bude pak uživatel moci přiřadit překlad v libovolném jazyce.

[6] Obecné označení pro technologie, které umožňují uživatelům přidávat a měnit obsah systému.
[7] http://wordnet.princeton.edu/
[8] http://www.wiktionary.org/
[9] http://www.wordnik.com/

3.5.2 Další položky databáze pojmů

Kromě překladu bude navíc možné přiřadit ke každému pojmu jeho *příklad využití v konverzaci*. Tato data bohužel nelze automaticky získávat z uvedených výkladových slovníků, protože zde nejsou rozlišena podle významu slov. Další položkou, kterou budou moci uživatelé vkládat do databáze ke každé definici významu pojmu, je obrázek nebo fotografie, která tento význam graficky vystihuje. Obrázky pak budou uživateli zobrazovány při procvičování pojmů, jak bylo stanoveno v sekci 3.1.

Centrální databáze bude navíc ke každému pojmu uchovávat jeho *slovní druh*[10]. Tuto položku získá systém sám z databáze Wiktionary a nebudou ji tak muset zadávat uživatelé.

[10] Ve výkladových slovnících bývá slovní druh často označován také výrazem *part-of-speech*.

Kapitola 4

Návrh systému

V této kapitole se budu zabývat zejména architekturou a strukturou jednotlivých částí aplikace, návrhem databáze, návrhem podoby URL[1] a algoritmem výuky.

4.1 Architektura aplikace

Většina webových frameworků a na nich postavených aplikací se v dnešní době hlásí ke vzoru Model-View-Controller (dále jen MVC) anebo k některé z jeho modifikací. I když se dnes rozmáhají diskuse o tom, zda je architektonický vzor MVC opravdu používán tak, jak byl původně navržen [3], jeho hlavní podstatou je rozdělení aplikace do tří na sobě příliš nezávislých vrstev:

1. *Model* spravuje data aplikace a doménovou logiku[2].
2. *View* zajišťuje zobrazení a reprezentaci dat z modelu.
3. *Controller* přijímá akce od uživatele, mění stav modelu a vyvolává překreslení view.

Architektura této aplikace se odvíjí od systému Nette Framework, který je jejím základem. Nette Framework se hlásí k architektuře Model-View-Presenter (dále jen MVP), která je zmíněné architektuře MVC blízce příbuzná. Jak vysvětluje hlavní vývojář Nette Framework ve svém článku o MVC a MVP [7], hlavní rozdíl mezi těmito architekturami spočívá v tom, že u MVP uživatel komunikuje pouze s view, které jeho vstupy dále deleguje na *presenter*. Ten má podobnou roli jako controller u MVC.

Rozdíl mezi MVC a MVP znázorňuje obrázek 4.1 a podrobněji se tomuto tématu věnuje například seriál [2]. Tato aplikace je tedy postavena na architektuře MVP.

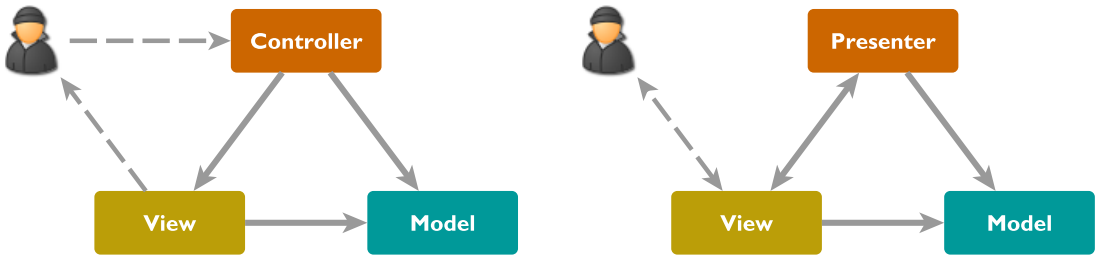
4.2 Struktura modulů

Rozsáhlejší internetové aplikace obvykle vyžadují rozdělení na více celků, které na sobě nejsou příliš závislé. V terminologii Nette Framework se k tomuto využívají *moduly*. Oficiální dokumentace frameworku popisuje modul jako logickou část aplikace, která sdružuje presentery, šablony, případně modely a komponenty do samostatného balíčku [15].

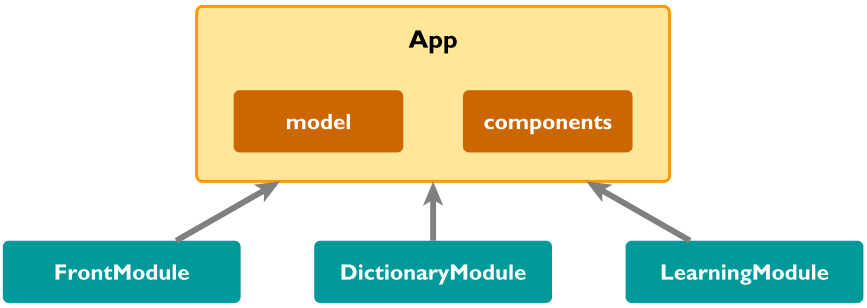
Různé moduly často potřebují přistupovat ke stejným datům a využívat stejné komponenty. K tomu Nette Framework nabízí tzv. *submoduly*, což jsou moduly, které jsou uvnitř

[1] Uniform Resource Locator

[2] Doménová nebo tzv. business logika zahrnuje základní operace nad daty, zejména na nižší úrovni.



Obrázek 4.1: Rozdíl mezi architekturami MVC a MVP



Obrázek 4.2: Návrh struktury modulů aplikace

jiného, jim nadřazeného, modulu. Celou aplikaci lze pak chápat jako jeden modul a její jednotlivé části jako submoduly.

Jak jsem již uvedl v kapitole 2, tato aplikace se bude skládat ze dvou základních celků – centrální databáze pojmů a výukové části. Protože obě tyto části budou přistupovat ke stejným datům, rozhodl jsem se celý systém pojmout jako modul, který bude obsahovat datovou část aplikace a některé globální komponenty. Každý jeho submodul si pak bude spravovat vlastní presentery a šablony. Při návrhu struktury modulů jsem kromě dvou výše zmíněných částí aplikace přidal jeden modul obecnější, který bude zahrnovat vstupní stránku aplikace, případně jakékoliv další informační sekce.

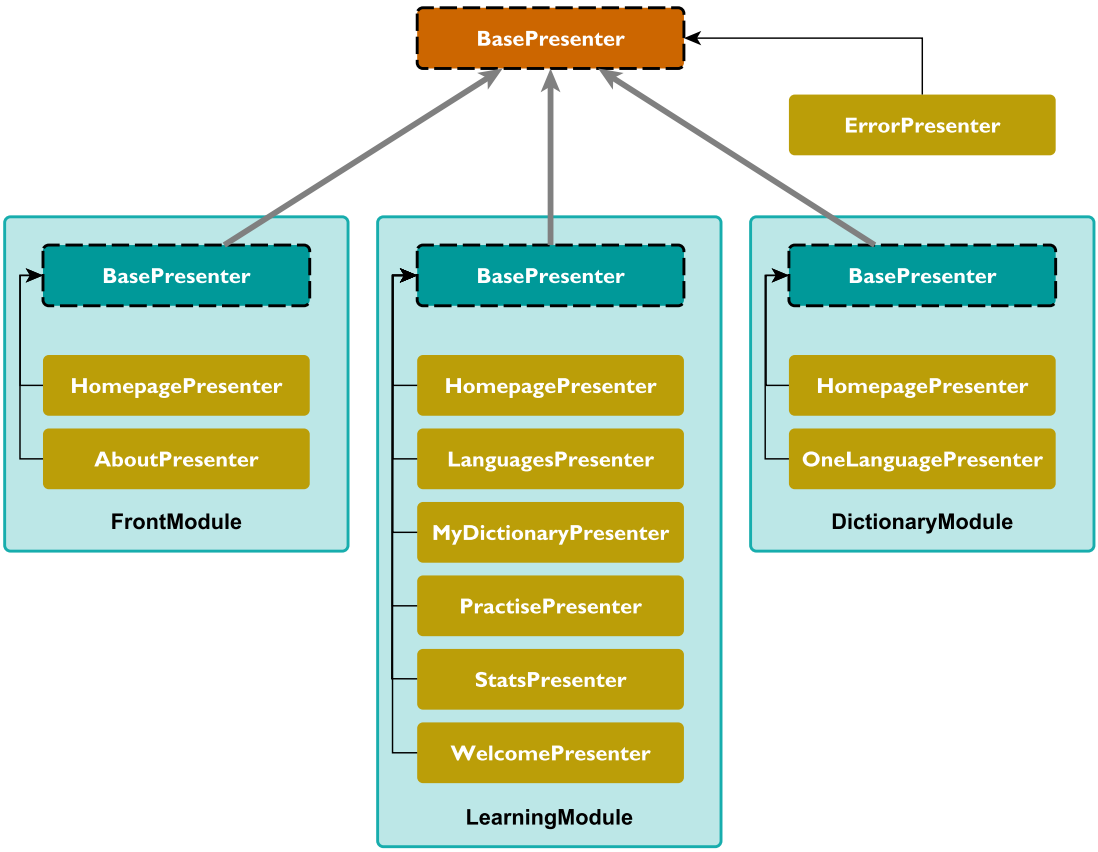
Výsledná navržená struktura modulů je znázorněna na obrázku 4.2. Názvy objektů v diagramu jsou voleny podle zvyklostí Nette Framework a při implementaci budou reflektovat názvy adresářů se zdrojovými kódy a strukturu jmenných prostorů[3]. Vztahy mezi moduly z implementačního hlediska představují vztah dědičnosti mezi třídami jednotlivých presenterů. Návrhu jejich struktury se věnuje následující sekce.

4.3 Struktura presenterů

V sekci 4.1 jsem vysvětlil pojem *presenter* a popsal jeho roli v aplikaci navržené podle vzoru Model-View-Presenter. Nyní rozeberu podrobný návrh struktury presenterů této aplikace.

Z pohledu objektově orientovaného programovacího jazyka je každý presenter představován speciální třídou, která má frameworkem definovaný životní cyklus. Podrobnému popisu životního cyklu presenteru v Nette Framework se věnuje článek [8] a oficiální dokumentace [17]. Presentery využívají třídní dědičnosti jazyka PHP a jsou uspořádány do

[3] *Jmenný prostor* neboli *namespace* je v PHP prostředek k rozdělení aplikace na logické celky a předcházení konfliktů názvů tříd z různých částí systému.



Obrázek 4.3: Návrh hierarchie presenterů

stromové struktury. Jejich hierarchie v této aplikaci je založena na struktuře modulů, která byla popsána v sekci 4.2.

Každý modul obsahuje vlastní abstraktní třídu `BasePresenter`, která je potomkem globálního abstraktního presenteru celé aplikace. Tato struktura umožňuje sdílet některé vlastnosti a metody napříč celou aplikací anebo uvnitř jednotlivých modulů. Všechna funkcionalita, kterou implementuje jakýkoliv presenter této aplikace, je tak přístupná všem jeho potomkům.

Kompletní návrh hierarchie presenterů je znázorněn na obrázku 4.3. Schéma zobrazuje zjednodušený *diagram tříd*, kde jsem pro přehlednost nezahrnul jednotlivé vlastnosti a metody presenterů a jejich vazby na modelovou vrstvu. Modrými rámečky jsou znázorněny jednotlivé moduly aplikace, přičemž každý z nich je zahrnut ve vlastním jmenném prostoru. Přerušovaným okrajem jsou vyznačeny abstraktní třídy, šipky znázorňují vztah dědičnosti a žlutě jsou vyznačeny ty presentery, které mohou přímo zobrazovat data.

`http://multi-linguist.com`	– FrontModule
`http://learning.multi-linguist.com`	– LearningModule
`http://dictionary.multi-linguist.com`	– DictionaryModule

Tabulka 4.1: Návrh podoby URL – rozdělení aplikace na subdomény

`http://dictionary.multi-linguist.com/dog`	– pojem *dog* ve všech jazycích
`http://en.dictionary.multi-linguist.com/dog`	– pojem *dog* v angličtině
`http://cs.dictionary.multi-linguist.com/pes`	– pojem *pes* v češtině

Tabulka 4.2: Návrh podoby URL – centrální databáze pojmů

4.4 Návrh podoby URL

Tvar URL[4] adres je na webu důležitý nejen z pohledu SEO[5], ale přispívá také přehlednosti webu z uživatelského hlediska. V URL by se tak měla vyskytovat především základní klíčová slova [10].

Protože je aplikace rozdělená na tři moduly, jak bylo popsáno v sekci 4.2, přiřadil jsem každému z modulů jednu subdoménu. Toto rozdělení znázorňuje tabulka 4.1. V každém okamžiku by tak i z podoby URL mělo být jasné, v jaké části webu se právě nacházíme.

U modulu s centrální databází slov jsem vyhledávaný pojem umístil za základ URL a oddělil jej od domény lomítkem. Je však potřeba rozlišovat ještě situace, kdy vyhledáváme v celé databázi a kdy v jednom konkrétním jazyce. K tomu jsem použil další úroveň subdomény, která může obsahovat dvoupísmenný kód jazyka podle normy ISO 639-1, jejichž seznam nalezneme na webu [9]. Pokud je kód uveden, vyhledává se v daném jazyce, pokud ne, je prohledávána celá databáze. Uvedenou podobu URL znázorňuje tabulka 4.2.

Ostatní části webu mají za doménovou částí a lomítkem uvedeno klíčové slovo, které vychází z názvu aktuálního presenteru. Případné další parametry se přidávají za tuto část.

4.5 Návrh databáze

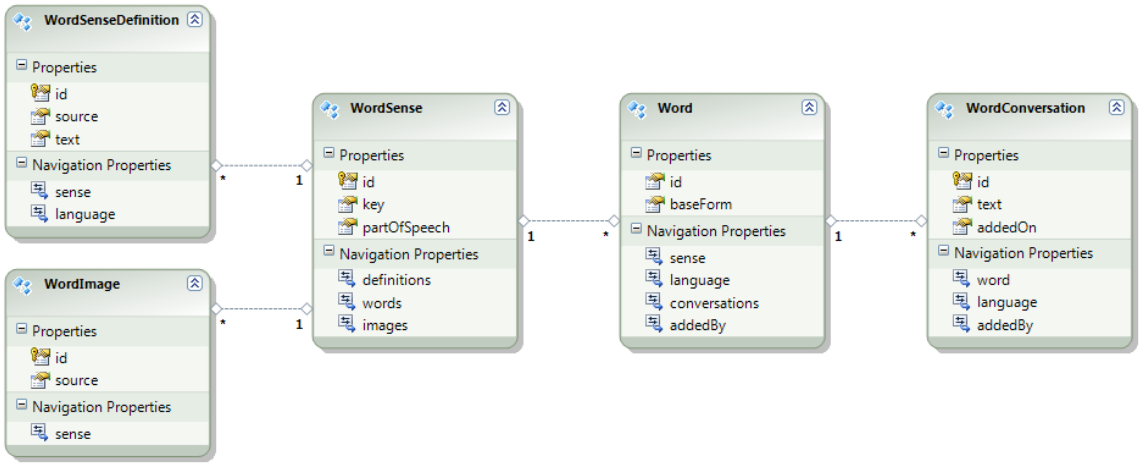
Protože je celý návrh aplikace silně objektově orientovaný, rozhodl jsem se takto pojmout i návrh databáze. V kapitole 6 pak vysvětlím, jaké technologie jsem použil pro ukládání dat a jejich mapování na objekty. Objektový návrh databáze se pomocí jazyka UML dá vyjádřit *diagramem tříd*. Diagram tříd na rozdíl od ERD[6] umožňuje vyjádřit i prvky a vztahy, které jsou běžné pro objektově orientované jazyky – zejména dědičnost, abstraktní třídy, případně rozhraní. Těchto výhod objektového návrhu jsem využil i při modelování databáze.

4.5.1 Mapování pojmů a jejich významů

V sekci 3.5 již bylo stanoveno, že pojmy uložené v databázi musí být vzájemně mapovány podle významu. To znamená, že ke každému pojmu v libovolném jazyce musí systém znát jeho význam a být schopen dohledat další pojmy, které mají význam stejný. Z hlediska databáze tyto vlastnosti vyžadují, aby *význam* a *pojem* byly dvě oddělené entity. Podstatu tohoto požadavku demonstruje obrázek 4.4.

[4] Uniform Resource Locator
[5] Search Engine Optimization
[6] Entity-Relationship Diagram

Obrázek 4.4: Mapování pojmů podle jejich významu



Obrázek 4.5: Data uchovávaná v databázi pojmů

4.5.2 Data uchovávaná v databázi pojmů

Diagram tříd na obrázku 4.5 znázorňuje návrh podoby entit, které budou uchovávat informace o jednotlivých položkách v databázi pojmů. Základem navržené struktury je třída `WordSense`, která reprezentuje význam určitého pojmu. Jejími atributy jsou unikátní identifikátor (`id`), základní tvar slova v angličtině (`key`) a slovní druh (`partOfSpeech`).

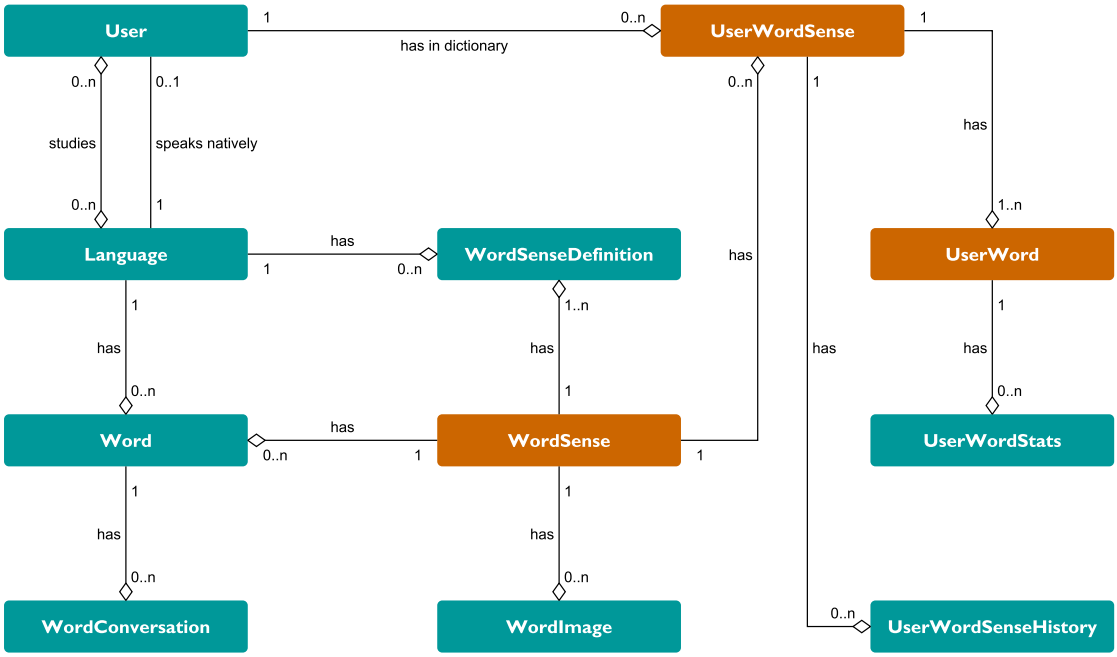
Na význam pojmu může být napojeno libovolné množství jeho překladů v různých jazycích. Tyto překlady jsou reprezentovány třídou `Word`, která uchovává zejména základní formu přeloženého slova (`baseForm`), jazyk překladu (`language`) a kým byl překlad do databáze přidán (`addedBy`).

Ke každému překladu slova dále může být přiřazeno libovolné množství užití tohoto překladu ve větě. Užití ve větě jsou reprezentována třídou `WordConversation`.

Ke každému významu slova se vážou také konkrétní znění definic tohoto významu (třída `WordSenseDefinition`) a obrázky, které daný pojem vystihují (třída `WordImage`).

4.5.3 Kompletní diagram tříd

V předchozích částech textu jsem vysvětlil, jak bude databáze mapovat pojmy a jejich významy, a jaká data mají být uchovávána. Po doplnění tříd potřebných pro zpracování statistik a udržování uživatelských dat dostaneme kompletní schéma databáze, které je znázorněno na obrázku 4.6. Jelikož je databáze poměrně rozsáhlá, soustředil jsem se v tomto



Obrázek 4.6: Diagram tříd databáze

schématu na vazby mezi jednotlivými třídami a úmyslně jsem neuvedl konkrétní atributy těchto tříd.

Dědičnost v tomto schématu z důvodu přehlednosti není znázorněna klasickými šipkami, ale pomocí barev. Všechny znázorněné třídy mají jako předka abstraktní třídu `Base`, která poskytuje základní funkčnost jako například správu unikátních identifikátorů jednotlivých entit a některé funkce pro pohodlnější práci s těmito třídami ve zdrojovém kódu.

Oranžově vyznačené třídy nemají třídu `Base` jako přímého předka. Tyto třídy dědí od předka `Ratable`, který zapouzdřuje schopnosti ohodnocení výuky a zpracování statistik a sám je potomkem třídy `Base`. Modrozeleně značené třídy jsou pak přímými potomky `Base`.

4.5.4 Optimalizace výkonu a denormalizace

Schéma 4.6 zachycuje vazby mezi třídami, které jsou vhodné z hlediska návrhu. Skutečná databáze bude potom v rámci optimalizace výkonu obsahovat větší množství vazeb, než je zde uvedeno. Na některých místech by totiž získávání statistických údajů spojováním několika entit nebylo dostatečně efektivní. V databázi tak budou na některých místech vznikat vazby, bez kterých by se složitým spojováním databázových tabulek dalo obejít. Tento proces se u relačních databází využívá poměrně často a označuje se pojmem *denormalizace*.

Kapitola 5

Návrh uživatelského rozhraní

Při návrhu uživatelského rozhraní jsem se snažil dodržet moderní, přehledný a jednoduchý vzhled. Inspiroval jsem se při tom oblíbenými a hojně navštěvovanými weby, se kterými se mi pohodlně pracuje a jejichž uživatelské rozhraní považuji za kvalitní. Jedná se především o Facebook[1], Github[2] a Twitter[3]. Tam, kde to bylo potřebné či užitečné, jsem s pomocí JavaScriptu vytvořil dynamické prvky a využil technologie AJAX[4], ale snažil jsem se aplikaci v tomto směru nepředimenzovat.

5.1 Layout aplikace

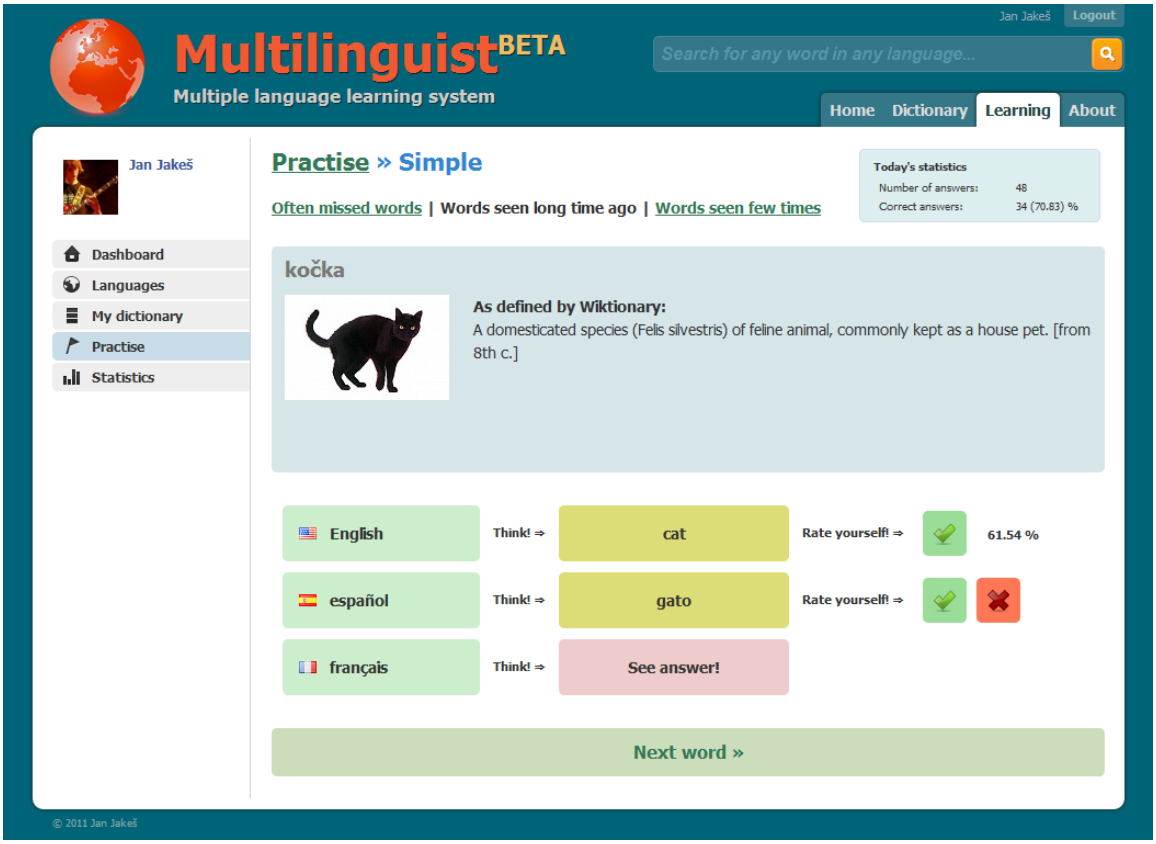
Hlavní okno aplikace má pevnou šířku 960 pixelů, takže by i s okraji a posuvníky nemělo přesáhnout 1000 pixelů. To je vzhledem k dnes běžným rozlišením monitorů nejpoužívanější rozměr webových portálů. Podrobné údaje o zastoupení jednotlivých rozlišení obrazovky mezi uživateli a maximální použitelné šířky webových dokumentů nalezneme na webu [20].

Navržený layout aplikace můžeme vidět na obrázku 5.1. Tělo dokumentu je ve všech sekcích webu rozděleno na dvě základní části:

1. *Horní panel* ve své levé polovině zobrazuje název aplikace a logo, které funguje též jako odkaz na úvodní stránku. Pravá polovina panelu shromažďuje nejdůležitější ovládací prvky webu na jednom místě. U horního okraje okna je umístěn prvek pro přihlášení resp. odhlášení uživatele. Pod ním je vyhledávací lišta, jejíž funkci podrobněji popisuje sekce 5.2. Ve spodní části tohoto panelu je umístěno menu, které slouží pro navigaci mezi hlavními částmi aplikace. Menu je opticky propojeno s obsahem stránky a viditelně zobrazuje, ve které části webu se právě nacházíme.

2. *Obsah stránky* vyplňuje celou stanovenou šířku dokumentu a je vždy uveden nadpisem nejvyšší úrovně. Pod tímto nadpisem je skrytá oblast, která po provedení některých akcí informuje uživatele o stavu systému. Tyto stavové zprávy[5] by měly být stručné a není potřeba je na stránce trvale zobrazovat, takže jsem implementoval jejich automatické skrývání pomocí JavaScriptu.

[1] http://facebook.com/
[2] http://github.com/
[3] http://twitter.com/
[4] Asynchronous JavaScript and XML
[5] V Nette Framework jsou tyto zprávy nazývané *flash messages*.



Obrázek 5.1: Základní layout aplikace

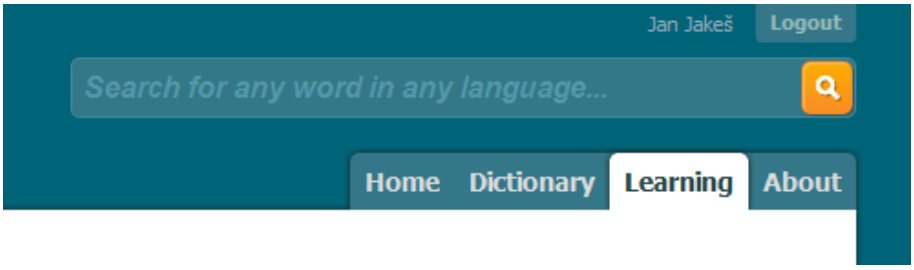
Ve výukovém modulu aplikace je do levé části obsahu stránky přidáno ještě další menu, které slouží pro navigaci mezi jednotlivými sekcemi tohoto modulu. Nad tímto menu je zobrazeno jméno uživatele, případně jeho profilový obrázek.

Při návrhu rozložení jednotlivých prvků na stránce jsem se snažil držet způsobů, na které jsou uživatelé webu zvyklí. Obecné zásady tvorby uživatelského rozhraní jsem čerpal ze starší knihy [18], která podrobně analyzuje design 50 v té době hojně navštěvovaných webů jako například Amazon, eBay či BBC Online. Podle jejích autorů je logo v 84 % případů umístěno vlevo nahoře, vyhledávací prvek najdeme na 35 % webů vpravo nahoře a u dalších 30 % pak vlevo nahoře.

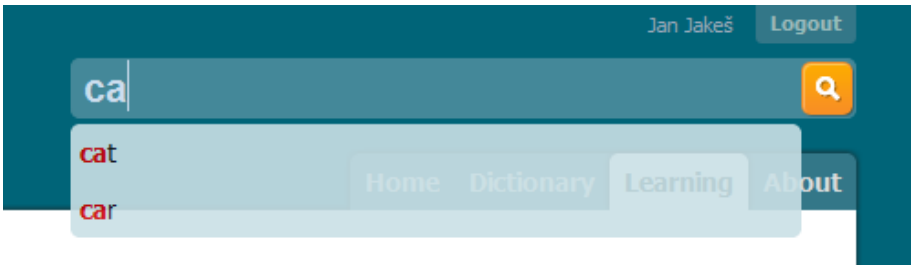
Jako hlavní navigaci jsem použil záložky, které podle uvedené knihy používá 34 % webů. Navigaci umístěnou vlevo najdeme na 30 % webových stránek, takže jsem ji použil jako menu druhého stupně. Na základě uvedené knihy jsem dále pro textovou oblast zvolil černý text na bílém pozadí, čímž je zajištěn vysoký kontrast a dobrá čitelnost. U odkazů jsem se držel klasického podtržení, protože tomu tak je na 80 % webových stránek.

5.2 Vyhledávací lišta

Vyhledávací lišta je součástí horního panelu, takže je přístupná z jakékoliv sekce webu. Tato lišta slouží k vyhledávání v databázi pojmů napříč všemi jazyky. Její důležitou součástí je AJAXový našeptávač, tedy prvek, který při psaní automaticky napovídá, které pojmy existují v databázi.



Obrázek 5.2: Základní podoba vyhledávací lišty



Obrázek 5.3: Vyhledávací lišta s našeptáváním

Tato vyhledávací lišta je jedním z nejdůležitějších prvků uživatelského rozhraní, takže jsem ji navrhl dostatečně velkou vzhledem k ostatním prvkům. Její základní podobu znázorňuje obrázek 5.2. Pokud není zrovna vyhledáván žádný pojem, zobrazuje lišta v decentních nerušivých barvách tzv. *placeholder*. Jedná se o text, který uživateli vysvětluje funkci této lišty a případně ho pobízí k jejímu použití. Tento prvek je součástí návrhu standardu HTML5, ale jeho podpora v prohlížečích je zatím mizivá, takže jsem ho implementoval vlastním způsobem pomocí JavaScriptu.

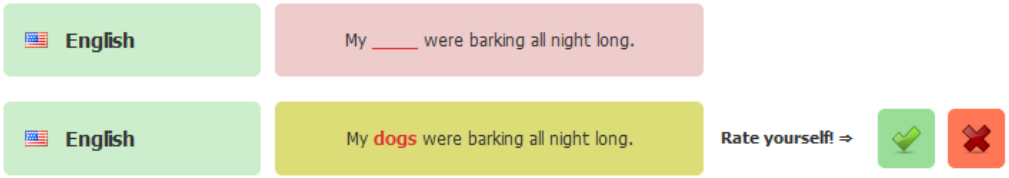
Podobu automatického našeptávače můžeme vidět na obrázku 5.3. Tento prvek vyhledává v databázi napříč všemi jazyky a napovídá uživateli podle části již zadaného textu. Po provedení akce vyhledávání je uživatel přesměrován do univerzálního slovníku a je mu sděleno, ve kterých jazycích tento pojem existuje. Odsud uživatel může přejít k pojmu v konkrétním jazyce.

Pro pokročilejší uživatele jsem dále navrhl vyhledávání pouze v jednom jazyce pomocí jednoduchého příkazu. Pokud uživatel zadá dvoupísmenný kód jazyka zakončený dvojtečkou, je vše, co za dvojtečkou následuje, vyhledáváno přímo ve slovníku konkrétního jazyka. Tento kód je navíc respektován i systémem našeptávání, takže jsou uživateli rovnou předkládány pojmy z žádaného jazyka. Konkrétní ukázku použití tohoto příkazu lze vidět na obrázku 5.4.

5.3 Výuka

Pohodlná výuka z pohledu uživatele je jedním ze základních požadavků na tuto aplikaci, takže bylo třeba promyslet návrh jednoduchého a přehledného výukového prostředí. Všechny typy výuky mají společné základní prvky. Jak je vidět z obrázku 5.1, do horní části výukové sekce jsem umístil odkazy, které kdykoliv v průběhu výuky umožňují změnit způsob výběru zkoušených pojmů. Pro přehled o počtu procvičených pojmů jsou zde umístěny také denní statistiky.

Obrázek 5.4: Vyhledávací lišta s našeptáváním v jednom jazyce



Obrázek 5.5: Ukázka výuky konverzace

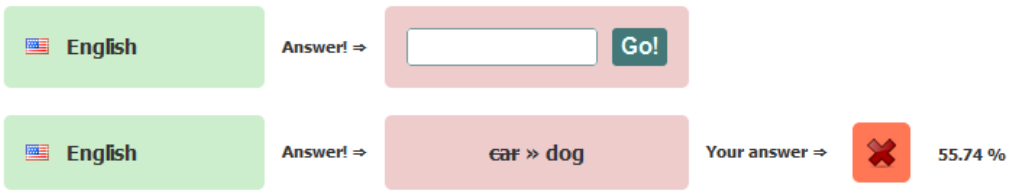
V horní polovině výukového okna je umístěn panel, který uživateli předloží pojem k procvičení. Ten je zde zobrazen v rodném jazyce uživatele, v podobě obrázku a pomocí definice. Pod tímto panelem je část, ve které je uživatel zkoušen postupně v jednotlivých jazycích. Tato část se liší podle typu výuky.

Na obrázku 5.1 je vidět podoba *jednoduché výuky*. První řádek zobrazuje pojem, který uživatel již zodpověděl, ve druhém řádku je uživateli právě nabídnuto sebeohodnocení a ve třetím řádku uživatel správnou odpověď ještě nezobrazil.

Obrázek 5.5 ukazuje podobu výuky konverzace před a po zobrazení správné odpovědi. Na obrázku 5.6 pak můžeme vidět podobu psané výuky. První řádek zde ukazuje vstupní pole pro zadání odpovědi. Na druhém řádku je zobrazen příklad odpovědi, kterou systém vyhodnotil jako špatnou, a následně zobrazil uživateli odpověď správnou.

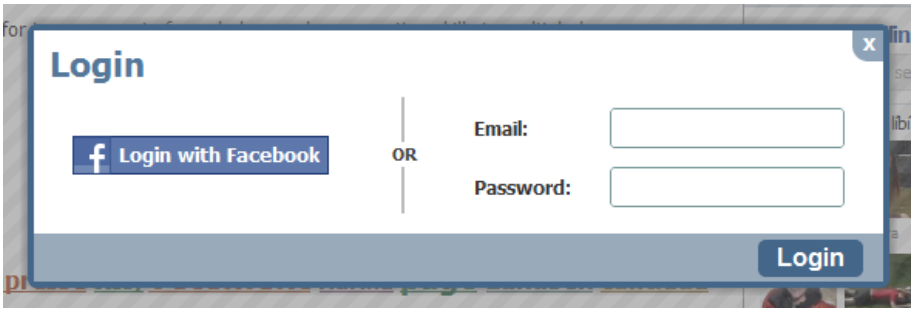
5.4 Databáze pojmů

Při návrhu zobrazení pojmů obsažených v databázi jsem se snažil přehledně zobrazit co nejvíce informací o daném významu slova na jednom místě. Každý nalezený význam slova je tak zobrazen na jedné kartičce, jejíž výslednou podobu znázorňuje obrázek 5.7. V levé části kartičky nalezneme slovní druh daného významu slova a s ním související obrázek. Pravá část kartičky je vertikálně rozdělena na dvě oblasti, z nichž horní zobrazuje informace, které se váží k významu slova. Spodní oblast pak zobrazuje data související s překladem pojmu



Obrázek 5.6: Ukázka psané výuky

Obrázek 5.7: Položka databáze pojmů



Obrázek 5.8: Grafický návrh dialogového okna

ve vybraném jazyce, mimo jiné se zde nachází příklady užití tohoto pojmu v konverzaci. U spodního okraje každé kartičky je lišta, která sdružuje ovládací tlačítka pro uživatele.

5.5 Dialogová okna

Příliš častá změna obsahu celé stránky může být pro uživatele matoucí a nepohodlná. Menší akce je často dobré řešit pomocí vyskakovacích oken, která nenaruší právě zobrazený obsah. Tento prvek uživatelského rozhraní je typický pro desktopová uživatelská rozhraní, ale na webu pro něj žádnou implicitní podporu nenajdeme. Proto jsem pro tuto aplikaci navrhl a implementoval vlastní dialogová okna, která jsou pak využita na několika místech.

Grafickou podobu dialogového okna můžeme vidět na obrázku 5.8. Při jeho návrhu jsem se nepouštěl do žádných nestandardních řešení, ale použil jsem typické rozložení prvků, které uživatelé znají z nejpoužívanějších operačních systémů a dalších aplikací. Toto dialogové okno lze zobrazit buď jako *nemodální*, kdy lze volně přecházet na všechny prvky na stránce pod ním, anebo jako *modální*, přičemž pak je celý dokument překryt tmavou poloprůhlednou vrstvou a uživateli je znemožněno pracovat s jinými prvky aplikace.

Navržené dialogové okno může uživateli nejen prezentovat jakýkoliv obsah, ale i přijímat libovolná data prostřednictvím formuláře, takže se jedná o poměrně univerzální komunikační prostředek. Způsobu implementace tohoto prvku se věnuje sekce 7.5.

Kapitola 6

Použité technologie

V této kapitole představím základní technologie, na kterých je tato aplikace postavena. Další malé knihovny a nástroje, které jsou v systému použity, budou popsány v kapitole 7.

Většina webových služeb je výsledkem sloučení několika různých technologií. Na nejvyšší úrovni abstrakce se webová aplikace obvykle skládá z komponent:

1. databáze,
2. serverová aplikace,
3. klientská aplikace.

Pro každou z těchto částí se používají odlišné technologie a je potřeba pečlivě zvolit, které knihovny použít. Následující text popisuje technologie, které byly vybrány pro jednotlivé části systému, a vysvětluje, proč byly zvoleny. Nástroje jsem volil tak, abych co nejlépe vyhověl stanoveným požadavkům na aplikaci a aby aplikace zároveň byla dostatečně robustní a rozšiřitelná.

6.1 Databáze

Nároky, které klade tato aplikace na databázové úložiště, jsou především:

- uchování relativně velkého množství pojmů a jejich překladů,
- rychlý přístup k jakýmkoliv položkám,
- konzistence dat.

6.1.1 NoSQL

Nějaký čas jsem uvažoval o přístupu dnes moderních NoSQL[1] databází, které mají oproti relačním databázím několik výhod. Především umožňují ukládat libovolně strukturovaná data přímo do jednoho záznamu, nemají pevné schéma a jsou velmi výkonné a dobře škálovatelné.

Podrobně jsem nastudoval systém MongoDB a dokonce jsem na něm zkusil velkou část aplikace implementovat. Zde jsem však začal narážet na zásadní problém – MongoDB a další

[1] Obecné označení pro databázové systémy postavené na principu odlišném od relačních databází.

NoSQL databáze negarantují konzistenci dat. Tento jev se nazývá *eventual consistency* a podrobně je popsán v knize o systému CouchDB [1].

Stejná kniha názorně vysvětluje také tzv. *CAP Theorem*, který říká, že z trojice konzistence dat, dostupnost dat a škálovatelnost, si lze při výběru databáze vybrat pouze dvě vlastnosti. Jelikož je pro tento systém zásadní udržovat konzistentní data a druhou požadovanou vlastností je jejich vysoká dostupnost, rozhodl jsem se NoSQL databáze opustit.

6.1.2 Relační databáze

Systémy, které podle výše uvedeného CAP Theoremu garantují konzistenci a vysokou dostupnost dat, jsou právě relační databáze. Schéma 4.6, které bylo navrženo v kapitole 4, navíc obsahuje velké množství vzájemných referencí mezi entitními množinami, což je typické pro strukturu relační databáze.

Ve chvíli, kdy ukládáme několik souvisejících záznamů do různých entitních množin, potřebujeme využít transakční zpracování, aby se databáze nedostala do nekonzistentního stavu. To je vlastnost, kterou relační databáze běžně nabízí, ale u NoSQL systémů jí nenajdeme. Z uvedených důvodů jsem se rozhodl použít databázi relační.

Při volbě konkrétní relační databáze jsem se rozhodl jít ověřenou cestou, takže jsem vybral systém MySQL[2]. Jedná se zřejmě o dnes nejpoužívanější databázový systém v prostředí webu, což přináší výhody v podobě existence mnoha různých nástrojů a studijních materiálů.

6.2 Mapování dat na objekty

Jak je uvedeno v kapitole 4, návrh tohoto systému je značně objektově orientovaný. Objektový popis jsem použil i při návrhu databáze a pro implementaci jsem hledal vhodnou technologii, která umožňuje mapování dat ze zvolené relační databáze na objekty jazyka PHP. Takové systémy v architektuře MVP, která byla popsána v sekci 4.1, zaujímají roli *modelu*. Nette Framework modelovou vrstvu neimplementuje vůbec a pro rozsáhlejší aplikace je potřeba použít externí řešení.

6.2.1 ORM

Systémy, které mapují data z relační databáze na objekty programovacího jazyka, se obecně nazývají ORM[3] a ve vyspělých jazycích, jako je Java nebo C#, se běžně používají. V PHP je situace o něco složitější. Do verze 5.2 byly implementace ORM nevyspělé, protože PHP nepodporovalo všechny pokročilé objektové vlastnosti. S příchodem PHP verze 5.3 se situace značně zlepšila a začala vznikat plnohodnotná řešení objektově-relačního mapování.

6.2.2 Doctrine 2

Nejznámější a pravděpodobně nejrozšířenější implementací ORM v PHP je knihovna Doctrine 2[4]. Doctrine sestavuje modelovou vrstvu podle schématu Entity-Repository-Mapper, o němž se více můžeme dočíst na webu Nette Framework [19]. Až na pětivrstvý model toto schéma abstrahuje článek [22], který rovněž vysvětluje, proč je tato architektura pro

[2] http://www.mysql.com/
[3] Object-Relational Mapping
[4] http://www.doctrine-project.org/projects/orm/2.0/docs/en

modelovou vrstvu vhodnější než alternativní *ActiveRecord*. Doctrine 2 má navíc kvalitní dokumentaci a je značně podporováno ze strany komunity Nette Framework. Z uvedených důvodů jsem pro implementaci modelové vrstvy aplikace použil právě Doctrine 2.

6.3 Serverová aplikace

Použití jazyka PHP a systému Nette Framework pro serverovou část aplikace bylo součástí zadání této práce, takže jsem se do žádného většího porovnávání frameworků nepouštěl.

Nette Framework poskytuje velice kvalitní a poměrně minimalistický základ pro tvorbu webových aplikací, což je přesně to, co jsem pro tuto aplikaci potřeboval. Příliš velké a robustní frameworky by pro tento systém byly pravděpodobně předimenzované, což by se mohlo negativně projevit na výkonu aplikace.

Od Nette Framework se částečně odvíjel i výběr dalších technologií a návrh architektury systému. Značné množství času bylo potřeba věnovat studiu tohoto frameworku zejména na základě oficiálního diskusního fóra, protože oficiální dokumentace se v době vývoje tohoto systému bohužel nachází ve velmi špatném stavu.

6.4 Klientská aplikace

Použití jazyků HTML a CSS je pro webové aplikace samozřejmostí. Popisovat účel a použití těchto jazyků není smyslem této práce, ale rád bych se zde pozastavil nad použitými verzemi těchto technologií.

Nové HTML5, jehož standard je zatím ve fázi návrhu, se už v současné době začíná ve webových aplikacích běžně používat. V rámci zachování co nejvyšší rozšiřitelnosti této aplikace do budoucna jsem se rozhodl aplikaci napsat v právě v HTML5, ovšem používám jenom takové vlastnosti HTML5, aby aplikace byla plně funkční v dnešních majoritních prohlížečích.

Obdobná je situace s CSS3. V aplikaci používám takové vlastnosti CSS3, aby si nimi poradila většina prohlížečů. V kapitole 7 dále vysvětlím, jak jsem použil pro jednotlivé prohlížeče specifické atributy a speciální skript pro simulaci některých CSS3 vlastností ve starších verzích Internet Exploreru.

6.5 Dynamické prvky uživatelského rozhraní

Výukové prostředí a některé další části aplikace obsahují dynamické prvky, na které bylo potřeba použít JavaScript. Žádný robustní framework zde nebyl potřeba, ale protože se samotný JavaScript chová napříč různými prohlížeči nekonzistentně a některé jeho konstrukce vedou na poněkud složité a nepřehledné zápisy, rozhodl jsem se v rámci přehlednosti a znovupoužitelnosti zdrojových kódů použít framework jQuery[5].

Framework jQuery navíc značně usnadňuje práci s DOM[6] selektory, grafickými efekty, AJAXovou komunikací a má perfektně zpracovanou dokumentaci.

[5] http://jquery.com/
[6] Document Object Model

6.6 Grafy a uživatelský slovník

Pro grafické zobrazení statistik jsem hledal knihovnu, která by umožnila efektní zobrazení grafů, které by se vykreslovaly na straně uživatele, tedy v klientské části aplikace. Přípravu grafů na straně serveru jsem zavrhl jako řešení, které by server zbytečně zatěžovalo.

Mnoho kvalitních zpracování vykreslování grafů nabízí řešení postavená na technologii Flash. Ačkoliv je Flash dnes poměrně běžně používaný, zdá se, že s příchodem HTML5 by mohl být opět na ústupu. V některých operačních systémech bývají navíc problémy s jeho ovladači.

V době, kdy jsem pracoval na implementaci tohoto systému, vyšla beta verze JavaScriptového frameworku ExtJS 4[7], která právě v této verzi přináší vyspělé rozhraní pro vykreslování grafů pouze na bázi JavaScriptu. Vykreslovací jádro pochází z frameworku Raphaël a po otestování této betaverze jsem dospěl k závěru, že framework ExtJS naprosto vyhovuje mým požadavkům.

Kromě grafů jsem ExtJS 4 použil ještě na výpis pojmů v uživatelském slovníku prostřednictvím komponenty `Grid`.

[7] http://www.sencha.com/products/extjs/

Kapitola 7

Implementace

Vybrané technologie pro implementaci již byly probrány v kapitole 6. V této kapitole se zaměřím na propojení použitých technologií, popíši další podpůrné technologie, které byly při tvorbě systému využity, a také podrobněji rozeberu důležité implementační detaily některých konkrétních částí systému.

7.1 Autentizace

Tradiční způsob autentizace na webových portálech obvykle vyžaduje registraci uživatele do systému, automatický kontrolní email ze strany aplikace a následnou aktivaci účtu. Tento způsob nepovažuji za příliš pohodlný, protože uživatel musí pro každý webový portál znovu volit uživatelské jméno a heslo a vyplňovat základní osobní data.

7.1.1 Facebook API

Facebook nabízí programátorům webových aplikací přihlašovací rozhraní, které je z pohledu uživatele velmi pohodlné. Místo složité registrace stačí uživateli, aby byl přihlášen na Facebook a při prvním přihlášení do externí aplikace odsouhlasit přístup k datům, která tato aplikace požaduje. Sám toto API využívám v několika aplikacích jako uživatel, takže jsem se rozhodl jej nastudovat i z pohledu programátora a zařadit do této aplikace. Podrobný popis tohoto rozhraní popisují vývojáři webu Facebook v článku [6].

7.1.2 Implementované řešení

Ačkoliv jsem jako hlavní autentizační mechanismus implementoval zmíněné Facebook API, nechtěl jsem, aby aplikace byla na tomto rozhraní závislá. Uživatelé, kteří Facebook nepoužívají, by neměli možnost se do této aplikace přihlásit a navíc by byl systém velmi těžko rozšiřitelný o další autentizační mechanismy.

Implementoval jsem tedy vlastní řešení, které každého uživatele přes Facebook API pouze ověří. Aplikace si poté pomocí autentizačních prostředků Nette Framework vytvoří vlastní relaci, která je na Facebooku nezávislá. Základní uživatelská data jsou uložena v databázi a při každém přihlášení automaticky aktualizována.

Tímto řešením jsem dosáhl nejen nezávislosti na relaci Facebooku, ale také možnosti snadného přidání dalších způsobů autentizace. Implementace nového autentizačního mechanismu vyžaduje jakýmkoliv způsobem získat od uživatele základní data a ověřit jeho identitu. Navíc je možné jednomu uživateli umožnit přihlášení několika odlišnými způsoby.

Pro účely testování jsem implementoval jednoduché přihlášení pomocí uživatelského jména a hesla.

7.2 Automatizace načítání a minifikace JavaScriptu a CSS

Jelikož je část aplikace napsaná v jazyce JavaScript a jazyku CSS se u webové aplikace nevyhneme téměř nikdy, rozhodl jsem se pro pohodlnější načítání těchto komponent použít doplněk WebLoader [12] z oficiálního repositáře doplňků Nette Framework.

7.2.1 WebLoader

WebLoader samotný zajišťuje pouze pohodlnější vkládání JavaScriptových a CSS souborů do šablon, spojování více souborů do jednoho a ukládání těchto zdrojových kódů do paměti cache. Jeho velmi silnou vlastností je ale možnost použití jakýchkoliv filtrů na vstupní soubory předtím, než se uloží do cache a vloží do šablony. Zde je právě možnost minifikace zdrojových kódů, která sníží objem přenášených dat a zvýší tak výkon aplikace.

Prostředí Nette Framework umí automaticky detekovat, zda je aplikace spuštěna na vývojovém či na produkčním serveru, díky čemuž jsem se rozhodl implementovat automatickou minifikaci zdrojových kódů pouze v produkčním prostředí. Ve vývojovém prostředí by tato vlastnost byla naopak velmi nepraktická pro ladění programu. Nástroje, které jsem použil pro minifikaci zdrojových kódů, popisuje následující část textu.

7.2.2 Minifikace zdrojových kódů

Minifikace zdrojových kódů JavaScriptu i CSS je aplikována na produkčním serveru a to ve fázi před uložením zpracovaného souboru do paměti cache. Tyto soubory jsou v paměti cache přegenerovány pouze v případě, že se změní zdrojový soubor, takže server není zatěžován jejich opakovaným vytvářením.

Pro minifikaci JavaScriptu jsem použil třídu JavaScriptPacker[1], pro minifikaci CSS pak třídu CssMin[2], která navíc nabízí další možnosti předzpracování vstupního CSS.

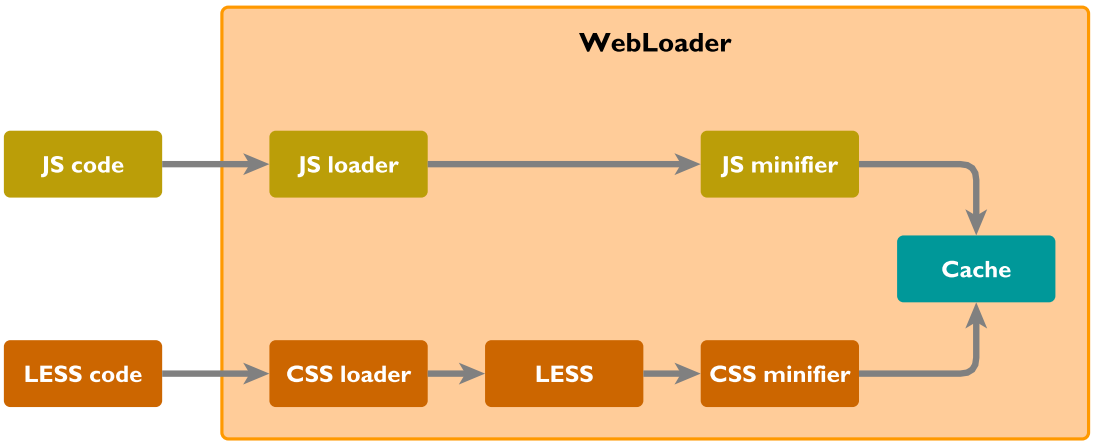
Kompletní schéma načítání, předzpracování a minifikace JavaScriptových a CSS zdrojových kódů znázorňuje obrázek 7.1. Jak si můžeme všimnout, ve schématu je zahrnut také framework LESS, kterému se věnuji v sekci 7.3.

7.3 Organizace CSS a knihovna LESS

Nástroj WebLoader zmíněný v části 7.2.1 umožňuje před uložením zpracovaných zdrojových souborů do paměti cache aplikovat na tyto soubory libovolné množství filtrů. To kromě snadného použití v sekci 7.2 uvedených minifikátorů značně ulehčuje nasazení jakýchkoliv knihoven, které by rozšiřovaly vlastnosti použitého jazyka.

Jazyk CSS v prohlížečích stále nepodporuje některé základní vlastnosti, které vývojářům jiné jazyky běžně nabízí a i když CSS3 přichází například se zavedením proměnných, stále kvůli slabé podpoře ze strany prohlížečů tyto pokročilejší vlastnosti nelze využívat.

[1] http://joliclic.free.fr/php/javascript-packer/en/
[2] http://code.google.com/p/cssmin/



Obrázek 7.1: Schéma WebLoaderu a jím využívaných komponent

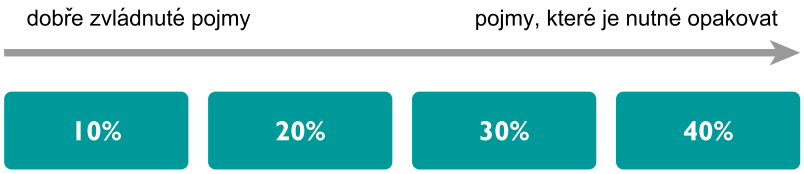
Mnoho z těchto problémů řeší CSS framework LESS[3], který jazyk CSS rozšiřuje o proměnné, funkce, zanořené výrazy, namespaces a další vlastnosti, které jsou typické spíše pro procedurální programovací jazyky. Framework LESS je ve své originální podobě napsán v JavaScriptu, což bych v této aplikaci považoval za zbytečné zatěžování prohlížeče na straně klienta. Místo toho jsem se rozhodl použít LessPHP[4], tedy PHP adaptaci tohoto frameworku. Dosáhl jsem tak velmi praktického rozšíření vlastností jazyka CSS, přičemž se toto nijak nepodepsalo na výkonu aplikace, protože vyrovnávací paměť s výsledným CSS bude přegenerována pouze v případě změny vstupního CSS souboru.

7.3.1 Vlastní knihovna CSS

Pokročilé schopnosti frameworku LESS jsem využil pro napsání malé knihovny funkcí, která je využívána prakticky ve všech CSS souborech v aplikaci. Jednou z nejdůležitějších implementovaných funkcí je metoda `clearfix`, která opravuje chybné obtékání plovoucích bloků. Tato metoda je založena na technice *jednoduchého ukončení obtékání*[5] Tonyho Assleta, kterou popisuje kniha [4]. Metodu uvedenou v knize jsem upravil tak, aby šla použít uvnitř libovolného zanořeného prvku a dosáhl jsem tak řešení, které bez přidávání jakýchkoliv atributů do HTML kódu vyřeší tento problém pouze na úrovni CSS.

Dále jsem do této knihovny implementoval funkce, které sjednocují rozdílný zápis pokročilých vlastností CSS verze 3 napříč různými prohlížeči. Jedná se o funkce pro zaoblení rohů blokových elementů, vykreslování stínů, průhlednost, či nastavení výpočtu velikosti prvků elementů. Pro starší verze prohlížeče Internet Explorer jsem použil knihovnu CSS3Pie[6], která umí prohlížečem nepodporované vlastnosti vykreslit pomocí skriptu.

[3] http://lesscss.org/
[4] http://leafo.net/lessphp
[5] Metoda je často nazývána právě *clearfix*, zejména v angličtině.
[6] http://css3pie.com/



Obrázek 7.2: Rozdělení slovíček do boxů při výběru z databáze

7.4 Výběr pojmů z databáze na základě pravděpodobnosti

O způsobu výběrů pojmů k procvičení v průběhu výuky jsem psal již v sekci 3.3. Navržené řešení počítá s náhodným výběrem, přičemž na základě předchozí výuky mají mít některé pojmy vyšší pravděpodobnost než jiné.

Z implementačního hlediska jsem se snažil dosáhnout toho, aby z databáze byl vždy přenesen pouze jeden pojem. Ke každému uživateli jsou v databázi k jeho pojmům ukládány i statistiky, které ovlivňují pravděpodobnost příštího výskytu daného pojmu, a to v takové formě, aby se podle těchto databázových sloupců dala data řadit. Před výběrem pojmu je vygenerováno číslo v rozsahu počtu položek uživatelského slovníku, přičemž nižší hodnoty tohoto čísla mají vyšší pravděpodobnost. Poté je použito řazení na databázový sloupec, který odpovídá zvolenému kritériu výběru a z databáze je vybrán pojem, který se nachází na pozici předtím vygenerovaného čísla. Takto lze tedy častěji vybírat například pojmy, ve kterých uživatel často chybuje.

Algoritmus výběru pojmů z databáze vychází z Leitnerova způsobu učení, který byl popsán v části 3.1.3. Pojmy jsou před výběrem z databáze rozděleny do několika boxů, z nichž každý má jinou pravděpodobnost pro dané kritérium výběru. Abych předešel boxům s příliš mnoha (nebo naopak příliš málo) pojmy, rozhodl jsem se implementovat proměnný počet boxů podle množství pojmů ve slovníku uživatele. Jak je vidět na obrázku 7.2, pravděpodobnost mezi jednotlivými boxy roste rovnoměrně a součtem pravděpodobností všech boxů samozřejmě dostaneme 100 %.

Později při testování tohoto systému jsem shledal jako matoucí, pokud je určitý pojem náhodou vybrán vícekrát po sobě. Přidal jsem tedy do algoritmu výběru výjimku, která zajišťuje, že uživateli nikdy nebude nabídnut stejný pojem dvakrát po sobě. Toto doplnění je implementováno pomocí ukládání identifikačního čísla posledního zkoušeného pojmu do uživatelské *session*. Tato session má nastavenou krátkou platnost, protože pokud uživatel na nějaký čas výuku přeruší, může mu být po návratu bez problémů znovu nabídnut stejný pojem, kterým předchozí výuku ukončil.

7.5 Dialogová okna

V sekci 5.5 byly vysvětleny výhody používání dialogových oken z pohledu uživatele a představen jejich grafický návrh. Z implementačního hlediska je dialogové okno tzv. *vykreslitelnou komponentou*.

7.5.1 Komponenty v Nette Framework

Pojmem *komponenta* v Nette Framework označujeme jakoukoliv třídu, jejíž předkem je třída `Nette\Component`. Komponenty tvoří v Nette stromovou hierarchii a programátorovi poskytují základní rozhraní pro tvorbu samostatných celků aplikace, které mají vlastní životní cyklus a reagují na definované signály zvenčí. Každá komponenta si udržuje svůj stav nezávisle na ostatních, takže její akce nemusí být začleněny do životního cyklu presenteru. To umožňuje tvorbu znovupoužitelných komponent, které mohou být používány na různých místech aplikace. Speciálním případem komponenty je i samotný presenter.

Vykreslitelná komponenta je komponenta, která má vlastní šablonu a definuje metodu, která obstarává její grafické zobrazení. Každá vykreslitelná komponenta je potomkem třídy `Nette\Application\Control`. Podrobnější informace nalezneme v dokumentaci Nette Framework [16].

7.5.2 Implementace dialogových oken

Všechna dialogová okna, která jsem implementoval v této aplikaci, mají jako společného předka třídu `App\Controls\Dialog`. V této třídě jsem definoval základní životní cyklus dialogového okna a parametry, kterými jej lze přizpůsobit. Okno tak může být modální, může obsahovat formulář a umožňuje přidávat tlačítka do speciální lišty, která je k tomu určena.

Komponenta dialogového okna není závislá na JavaScriptu a zobrazí se i při běžném požadavku pomocí znovunačtení celé stránky. Pokud je ale požadavek na zobrazení okna AJAXový, je jako odpověď ze serveru vráceno opravdu pouze toto okno a všechny jeho prvky jsou navíc aktivovány jako AJAXové. Znamená to, že například tlačítka pro zavření okna nebo odeslání jeho obsahu jsou také zpracována pomocí AJAXu a okno je takto i uzavřeno. Naopak v případě běžného požadavku okno vždy vykreslí ovládací prvky tak, aby vyvolaly přenačtení celé stránky.

Konkrétní dialogová okna použitá v aplikaci jsou pak přímým potomkem výše popsaného okna a definují obsluhu akcí, pro které jsou určena. V systému jsou okna použita pro přidávání položek do databáze, nastavení jazyků pro studium a přihlašování uživatelů.

7.6 Nástroje použité při implementaci

Vývoj aplikace probíhal v prostředí NetBeans IDE s nainstalovaným pluginem pro Nette Framework. Některé prvky uživatelského rozhraní byly vytvořeny v programu Adobe Photoshop. Změny byly zanášeny do verzovacího systému Git a pro testování funkčnosti aplikace jsem použil prohlížeče Google Chrome, Mozilla Firefox, Internet Explorer, Opera a Safari. Pro ladění aplikace ve starších verzích prohlížeče Internet Explorer jsem používal vývojové módy programu Internet Explorer 9 a aplikaci IETester[7].

[7] http://www.my-debugbar.com/wiki/IETester/HomePage

Kapitola 8

Produkční nasazení aplikace

Nasazení aplikace do produkčního prostředí obvykle není jednoduchý proces a jak ukazuje anketa u článku [11], potřeba jeho automatizace je mezi vývojáři webových aplikací poněkud podceňována. Ruční nahrávání souborů na server je nejen nepohodlné, ale může při změnách v aplikaci přinést závažné chyby. Částečným řešením tohoto problému může být využití verzovacích systémů jako jsou Git, Mercurial či SVN, ovšem ani zde nemáme nad procesem produkčního nasazení dostatečnou kontrolu a není zde zajištěna atomicita při přechodu na novou verzi aplikace.

Ideální řešení nabízí nástroje, které jsou k produkčnímu nasazení aplikace přímo určené. Ve světě PHP bohužel takové nástroje prakticky neexistují. Naopak výborné a komplexní řešení přináší nástroj Capistrano[1], který je napsán v jazyce Ruby a původně určen pro framework Ruby on Rails. Díky svému úspěchu se tento nástroj začal používat i pro aplikace postavené na jiných frameworcích, které ani nemusí být založeny na jazyce Ruby. Nástroj Capistrano jsem proto vybral pro produkční nasazení této aplikace.

8.1 Capistrano

Pro použití nástroje Capistrano stačí mít na vývojové stanici nainstalován jazyk Ruby s balíčkem Capistrano a v případě aplikace postavené na PHP ještě balíček Railsless-deploy.

Capistrano se ovládá z příkazové řádky a jediným příkazem `cap deploy` lze nahrát novou verzi aplikace na server. Úpravy v aplikaci jsou na server přeneseny pomocí libovolného verzovacího systému a změna aplikace na novou verzi je atomická. Tuto atomicitu Capistrano zajišťuje tak, že napřed vytvoří adresář s novou verzí aplikace a poté z kořenového adresáře webu připojí tento adresář pomocí symbolického odkazu. Žádnému uživateli se tak v tu chvíli nemůže stát, že by nějakou chvíli pracoval se systémem v nekonzistentním stavu.

Pokud zjistíme, že nám změny v nové verzi aplikace nevyhovují anebo dělají na produkčním serveru problémy, můžeme se příkazem `cap rollback` vrátit k předchozí verzi aplikace anebo libovolné verzi starší. Tento přechod je opět realizován atomicky pomocí změny symbolického odkazu. Přenosy mezi vývojářským a produkčním serverem jsou realizovány pomocí nástroje SSH.

[1] https://github.com/capistrano/capistrano

8.1.1 Soubor `capfile`

Abychom nástroj Capistrano mohli používat výše uvedeným způsobem, je potřeba definovat jeho nastavení a chování v souboru `capfile`. Tento soubor je nástrojem Capistrano automaticky vyhledán v aktuálním adresáři.

V souboru `capfile`, který jsem vytvořil pro tuto aplikaci, jsem kromě základního nastavení v jazyce Ruby definoval úlohu, která nastaví práva přístupu pro složky, které jsou aplikací sdíleny s předchozími verzemi. Jedná se o složky s dočasnými soubory vyrovnávací paměti a obrázky, které byly uživateli systému nahrány na server. Navíc jsem implementoval úlohu pro jednorázové automatické vygenerování tzv. *proxy classes*, které využívá systém Doctrine 2.

8.2 Rozdílné chování aplikace v development a production módu

Nette Framework dokáže na základě IP adresy automaticky detekovat, zda se aplikace nachází ve vývojovém či produkčním prostředí. Jednak tuto informaci sám používá pro rozhodování, zda zobrazovat ladící výpisy a vývojářský panel anebo automaticky regenerovat vyrovnávací paměť načítání souborů, jednak je tato informace přístupná programátorovi. Uvedené detekce jsem v tomto systému využil na následujících místech:

- V produkčním prostředí jsou automaticky minifikovány zdrojové kódy CSS a JavaSriptu a poté jsou spojovány do jednoho souboru. To sníží objem přenášených dat a požadavků na server. Ve vývojovém prostředí se tak neděje, aby bylo možné aplikaci snáze ladit.

- JavaScriptové knihovny jQuery a ExtJS jsou v produkčním prostředí načítány ve svých minifikovaných verzích. Ve vývojovém prostředí nikoliv. Důvod tohoto chování je stejný jako v předchozím bodě.

- Doctrine 2 umí automaticky za běhu generovat tzv. *proxy classes*, které jsou důležité pro pozdní načítání referencí mezi entitami. Ve vývojovém prostředí je toto chování velmi pohodlné, v prostředí produkčním je z hlediska výkonu naopak nežádoucí a je lepší vygenerovat tyto třídy jednorázově při produkčním nasazení aplikace.

- V produkčním prostředí využívá Doctrine 2 pro uchování struktury databáze a vygenerované dotazy vyrovnávací paměť Alternative PHP Cache, ve vývojovém prostředí by to bylo spíše nevhodné, protože zdrojové soubory jsou při vývoji často měněny a paměť by se musela neustále obnovovat.

Kapitola 9

Testování a ladění

Ještě před dokončením implementace proběhla schůzka se zástupci společnosti Lingea, s.r.o. Tato firma se zabývá tvorbou slovníků a konverzačních publikací pro nejrůznější jazyky, a i když výukový software nepatří mezi jejich produkty, projevili jistý zájem a ochotu spolupracovat na této aplikaci. Po domluvě mi byla poskytnuta testovací data čítající téměř 1000 pojmů v 6 různých jazycích.

9.1 Úpravy aplikace pro testování

Data poskytnutá společností Lingea, s.r.o. bohužel neobsahovala slovní definice významu jednotlivých pojmů. To ale nebyl velký problém, protože aplikace byla naimplementována tak, aby byla schopna spolupracovat s anglickým výkladovým slovníkem Wiktionary. Systém jsem tedy upravil tak, aby v případě nenalezení definice pro určitý pojem dokázal uživateli nabídnout definice ze slovníku Wiktionary. Uživateli pak stačí vybrat vhodnou definici daného pojmu a přiřadit ji k položce v databázi.

Oproti původnímu návrhu aplikace tedy nejsou tyto definice stahovány z externího zdroje při vyhledávání v databázi, ale jsou uživateli pouze nabídnuty pro přiřazení k danému pojmu z dat od společnosti Lingea, s.r.o.

9.2 Průběh testování

Testování bylo řízeno pomocí dotazníku, který se skládal z 27 bodů. Dotazník byl rozdělen na tři základní části:

1. *Úvodní část* zkoumala jazykové a počítačové schopnosti testerů. Uživatelé zde navíc byli rozděleni do různých věkových skupin.
2. Část *testování aplikace* provázela uživatele systémem a stanovila mu přesné úkoly, které měl splnit. Uživatel byl poté tázán na přehlednost aplikace, ergonomii uživatelského rozhraní a na detaily ohledně jednotlivých funkcí aplikace.
3. *Závěrečná část* zjišťovala, jaké pocity zanechala na uživatelích aplikace jako celek. Testeři byli tázáni na pozitivní i negativní podněty v průběhu testování a měli možnost vyjádřit detailní názor na aplikaci.

Dotazník byl realizován pomocí technologie Google Docs. V průběhu celého testování byl sledován pohyb uživatelů v systému a automaticky zaznamenávány statistiky a pozice kliknutí myši. Jejich analýze se věnuje sekce 9.5.

9.3 Výsledky jednotlivých testů

Procesu testování se zúčastnilo 25 uživatelů, z nichž 72 % pokrývalo věkovou skupinu v rozmezí 20–30 let. Dalších 16 % účastníků tvořili uživatelé pod 20 let, ale našli se i účastníci v rozmezí 30–40 let a uživatelé ve věku nad 50 let.

Téměř polovinu testerů tvořili běžní uživatelé, kteří jsou zvyklí pracovat s internetovými aplikacemi, ale nemají žádné pokročilejší znalosti počítače. Necelá čtvrtina testerů byli uživatelé, kteří své schopnosti práce s počítačem označili za pokročilé a zbývající část testerů tvořili programátoři.

Většinu testerů tedy tvořili studenti středních a vysokých škol, kteří mají základní zkušenosti s prací na počítači, přičemž tato skupina bude také pravděpodobně zastupovat největší procento mezi budoucími potenciálními zákazníky aplikace.

Každý uživatel strávil v systému v průměru 19 minut.

9.3.1 Jazykové schopnosti testerů

Zhruba dvě třetiny testerů tvořili uživatelé, kteří se věnují studiu dvou cizích jazyků. Čtvrtinu pak pokrývali uživatelé, kteří studují pouze jeden cizí jazyk a zbytek uživatelé, kteří studují 3 nebo 4 cizí jazyky. Přes 75 % testerů se tedy věnuje více než jednomu cizímu jazyku, což je důležitý fakt pro testování aplikace, která se zaměřuje na vícejazyčnost. Aplikace je však bez problémů použitelná i pro ty, kteří chtějí procvičovat pouze jeden cizí jazyk.

Zastoupení jednotlivých jazyků mezi testery znázorňuje obrázek 9.1. Graf ukazuje počty studentů těch jazyků, ve kterých byla poskytnuta testovací data.

Pouze 8 % uživatelů procvičuje slovní zásobu a konverzační schopnosti každý týden, každý den se ke studiu nedostane dokonce žádný z tázaných. Téměř čtvrtina pak tyto schopnosti rozvíjí několikrát za rok a ještě o něco více testerů uvedlo, že slovní zásobu a konverzační schopnosti neprocvičují prakticky vůbec. Zbylých 40 % tázaných se k tomu dostane několikrát za měsíc. Zhruba polovina uživatelů se tedy k procvičování těchto dovedností dostane maximálně několikrát do roka.

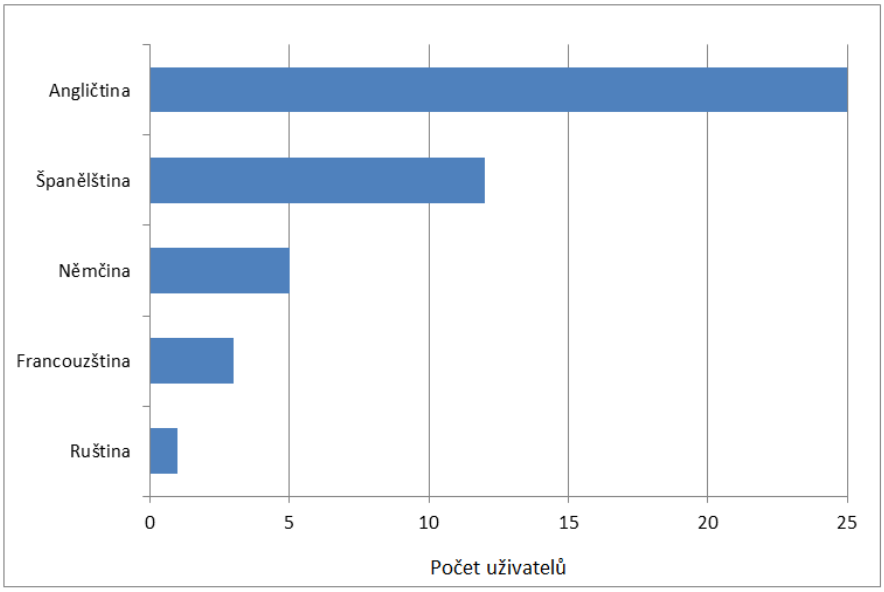
Všichni tázaní uživatelé mají pocit, že by tyto dovednosti měli procvičovat častěji. Přes 90 % respondentů je o tom přesvědčeno s jistotou, zbytek si pak myslí, že by častější studium zřejmě bylo vhodné.

9.3.2 Práce s databází pojmů

Při testování měli uživatelé za úkol vyhledat v databázi alespoň 10 různých pojmů a to tak, aby si vyzkoušeli hledání v celé databázi i v jednotlivých jazycích. Dvě třetiny respondentů nemělo s tímto úkolem žádné problémy, zbylá třetina pak označila některá místa za lehce nepřehledná. Závažnější problémy s orientací v této části neměl nikdo. Ačkoliv se v této části nakonec bez větších problémů zorientovali všichni tázaní uživatelé, je zde zřejmě určitý prostor pro zlepšení, na který je třeba do budoucna myslet.

V další fázi testování byli uživatelé požádáni, aby se pokusili vytvořit nové položky v databázi pojmů. Test se skládal z následujících úloh:

- *Přidání užití daného pojmu v konverzaci* – ačkoliv celých 80 % testerů označilo tuto funkci systému jako snadno pochopitelnou a dalších 12 % zmínilo pouze mírné obtíže při prvním použití, našel se i uživatel, kterému se tato úloha zdála nepohodlná, a také uživatel, který systém vkládání konverzací nepochopil vůbec. Při kontrole vložených



Obrázek 9.1: Zastoupení jazyků mezi účastníky testování

konverzací jsem navíc zjistil, že několik z nich bylo přiřazeno k jinému jazyku, než je ten, ve kterém byly konverzace uvedeny.

Z uvedených skutečností vyplývá, že tento prvek uživatelského rozhraní je potřeba v dalším vývoji vylepšit. Zřejmě bude potřeba uživatele názorněji vést k cíli, například v několika krocích.

- *Přidání obrázku k významu slova a vložení nového překladu* – tyto úkony byly pro testery označeny jako volitelné, nicméně většina z nich si aspoň jednu z úloh vyzkoušela. S výjimkou jediné odpovědi výsledky testování nenaznačují, že by uživatelé měli s těmito akcemi problémy.

V této části navíc dostal každý uživatel za úkol přidat alespoň 10 pojmů do svého uživatelského slovníku. S těmito daty pak byly prováděny testy výukové části aplikace, které popisuje sekce 9.3.4.

9.3.3 Vyhledávací lišta

Vyhledávací lišta je jedním z nejdůležitějších prvků uživatelského rozhraní, a tak jí byla věnována samostatná část testování. Při předchozích úlohách, ve kterých měli uživatelé pracovat s databází pojmů, nebyla vyhledávací lišta úmyslně zmíněna. Po splnění zadaných úkolů byli uživatelé dotázáni, zda pro procházení databáze pojmů použili právě tuto lištu. Kladně odpověděla polovina z nich, ostatní dostali za úkol vyzkoušet práci s vyhledávací lištou dodatečně.

Vyhledávací lišta jako prvek uživatelského rozhraní byla pak všemi testery hodnocena velmi kladně. Z uvedených skutečností lze usuzovat, že by nový uživatel měl být systémem na možnost použití této lišty upozorněn, případně by měl tento prvek být na obrazovce výraznější.

V dalším kroku byla uživatelům popsána možnost pokročilého použití vyhledávací lišty pro omezení výsledků na určený jazyk. Celým 60 % respondentů se tato funkce velmi za-

mlouvala a dalších 36 % ji přijalo poměrně kladně. Tato funkce tedy určitě není zbytečná, ale je potřeba do budoucna dobře promyslet, kam v uživatelském rozhraní umístit její vysvětlení uživateli.

9.3.4 Jednotlivé typy výuky

K výukové části aplikace nedostali testeři prakticky žádný popis, jak ji používat, protože tato část musí být intuitivní sama od sebe, aby uživatele neodrazovala od studia. Respondenti tedy dostali za úkol vyzkoušet si všechny typy výuky, které aplikace nabízí, a poté je ohodnotit.

Výsledek byl poměrně překvapivý, protože většina uživatelů označila jako nejlepší způsob výuky metodu *typing practise*. Metoda *simple practise*, kterou jsem osobně považoval za uživatelsky nejpřívětivější, se umístila na druhém místě. Zdá se, že uživatelům vyhovuje aktivnější zapojení do výuky a následné automatické vyhodnocení odpovědi systémem. Metoda *conversation practise* dopadla v testu nejhůř, ovšem výsledek může být ovlivněn skutečností, že se pro tuto část bohužel nepodařilo sehnat testovací data a uživatelé tak byli odkázáni na data, která sami do systému přidali.

9.4 Zhodnocení aplikace jako celku

Jednotlivé testy popsané v předchozích sekcích uživatele postupně prováděly aplikací a zkoumaly ergonomii, funkčnost, intuitivnost a užitečnost jednotlivých prvků aplikace. Na závěr těchto testů byl každý uživatel vyzván, aby si dále vyzkoušel libovolné funkce aplikace. Respondenti tak dostali prostor, aby si utvořili celkový náhled na aplikaci bez toho, aniž by na ně byly kladeny další cílené požadavky. Ve chvíli, kdy byli se zkoušením systému hotovi, byla jim předložena závěrečná část dotazníku.

9.4.1 Způsob výuky

Na dotaz, zda by uživatelům vyhovoval způsob výuky, který nabízí tato aplikace, odpovědělo kladně přes 90 % respondentů. Zhruba polovina z nich odpověděla jednoznačně kladně, druhá polovina se k této odpovědi přikláněla. Nikdo z uživatelů tento způsob výuky vyloženě neodmítl, ačkoliv 8 % odpovědělo, že by jim zřejmě spíše nevyhovoval. Téměř čtvrtina respondentů odpověděla, že by se s aplikací tohoto typu rozhodně dostala k procvičování slovní zásoby a konverzačních schopností častěji než nyní a dalších 60 % si myslí, že by tomu tak zřejmě bylo. Zbylých 16 % odpovědělo, že by jim v tom takováto aplikace pravděpodobně nepomohla.

Navzdory určité skepsi některých zástupců společnosti Lingea, s.r.o. odpovědělo více než tři čtvrtiny respondentů, že jim vyhovuje procvičovat slovní zásobu a konverzační schopnosti ve více jazycích najednou. Zhruba stejný počet uživatelů považuje za užitečné mít přehled o své slovní zásobě na jednom místě a ještě nepatrně větší procento respondentů považuje tento přehled vlastní slovní zásoby ve spojení se statistikami výuky za motivaci k dalšímu studiu.

Tyto výsledky považuji za velmi důležité pro budoucnost aplikace a myslím, že v dalším vývoji má rozhodně smysl pokračovat.

9.4.2 Celkový dojem

Na závěr dostali uživatelé za úkol zhodnotit přívětivost a přehlednost uživatelského rozhraní pomocí známek 1–5, podobně jako ve škole[1]. Po vypočtení aritmetického průměru z jednotlivých hodnocení dostáváme výslednou známku 1,64. Výsledek není špatný, ale naznačuje, že v této oblasti ještě existuje prostor pro zlepšení.

Obdobným způsobem ohodnotil každý uživatel design aplikace. Vypočtený průměr je roven známce 1,8. Vzhled aplikace je však z velké části věcí individuálního vkusu, takže lze poměrně těžko hledat konkrétní podněty pro zlepšení. Žádoucí ovšem je, aby se design systému zamlouval co největšímu procentu jeho uživatel.

Poslední otázkou, která byla položena respondentům, bylo celkové zhodnocení aplikace. Zhruba dvěma třetinám uživatelů se aplikace jako celek líbí „docela“, zbylé třetině pak „hodně“. Žádné negativní ohlasy se u této otázky neobjevily.

9.5 Zhodnocení klikacích map

V průběhu celého testování byla anonymně zaznamenávána jednotlivá kliknutí myši ve výukové části aplikace. K záznamu těchto dat a následnému vytvoření map intenzity jednotlivých kliknutí[2] jsem použil službu mYx[3].

Klikací mapy slouží zejména k otestování ergonomie uživatelského rozhraní. Uživatelé by měli klikat tam, kam bylo při návrhu designu aplikace zamýšleno. Ovládací prvky by měly být využity, pokud nejsou, může to značit jejich nesprávné umístění, případně nepotřebnost na dané stránce.

Použití klikacích map při testování této aplikace pomohlo odhalit několik nedostatků návrhu uživatelského rozhraní. Klikací mapa obrazovky *simple practise* je znázorněna na obrázku 9.2. Zvyšující se intenzita kliknutí je zde znázorněna přechodem od modré barvy přes červenou, místy až k bílé. Jak je vidět z obrázku, určitá část uživatelů se pokoušela kliknout na předložený pojem, jeho obrázek a definici. Tento prvek uživatelského rozhraní ale uživateli pojem pouze předkládá a na kliknutí nijak nereaguje. Ještě o něco větší část testerů se pokoušela kliknout na názvy jednotlivých jazyků. Opět se jedná o informační prvek, který na kliknutí myši nereaguje. Jak jsem se tento problém pokusil vyřešit, popisuje část 9.6.

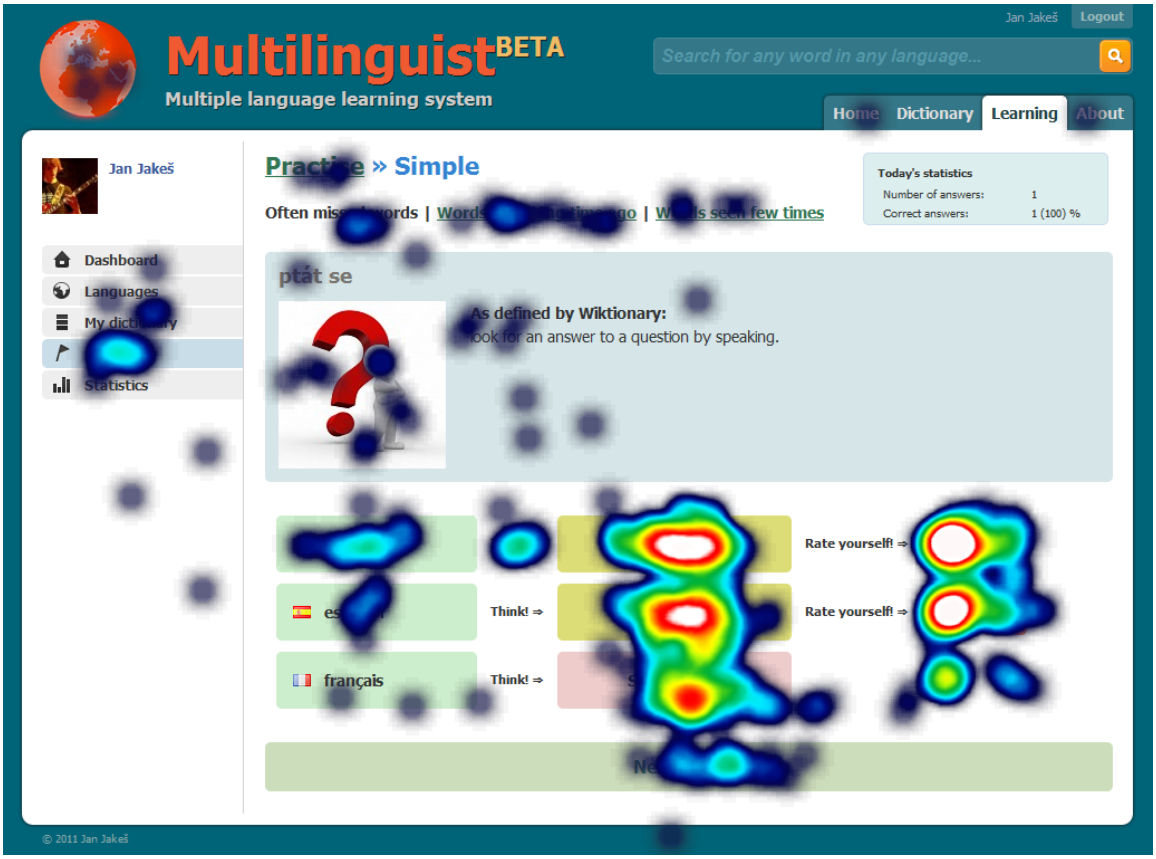
Z uvedené klikací mapy lze ale vyčíst také pozitivní stránky návrhu uživatelského rozhraní. Jak je vidět, uživatelé očekávají reakci na kliknutí na logo aplikace, pravděpodobně v podobě přechodu na úvodní stránku systému. Přesně tímto způsobem aplikace na tuto akci také zareaguje. Dále lze usuzovat, že se na stránce nenacházejí zbytečné prvky, protože prakticky všechny byly využity. Jedinou výjimkou je horní vyhledávací lišta, kterou nikdo z respondentů v této části aplikace nepoužil, ovšem z ostatních klikacích map a odpovědí v části 9.3.3 lze odhadovat její značné využití.

Kompletní sada klikacích map z výukové části aplikace je přiložena na CD A.

9.6 Úpravy aplikace na základě testování

Testování aplikace přineslo důležité poznatky ohledně jejího návrhu a odhalilo několik chyb v implementaci. Všechny chyby byly opraveny a některé drobnější náměty na vylepšení byly

[1] Známka 1 tedy značí nejlepší hodnocení, 5 pak nejhorší.
[2] Tzv. *klikací mapy*, v angličtině též *heatmaps*.
[3] http://www.myx.cz/



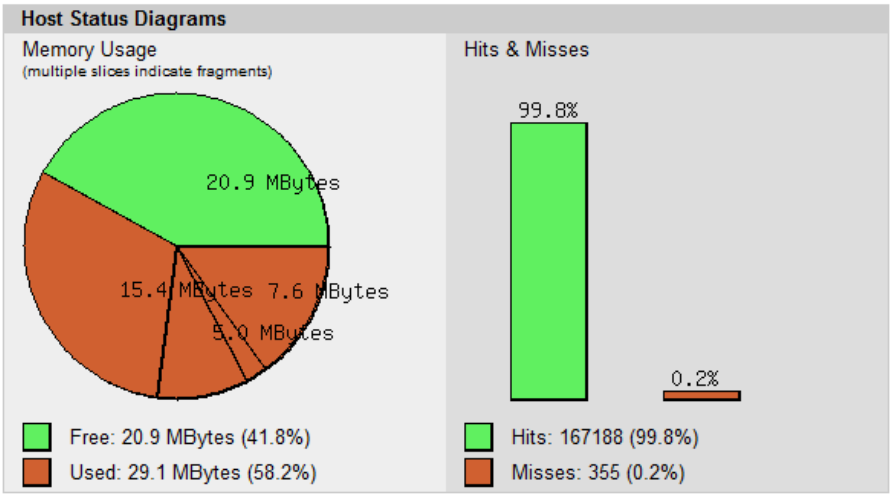
Obrázek 9.2: Klikací mapa jako výstup z testů výukové části aplikace

ještě do systému zaneseny. Další informace, které přineslo testování, případně konkrétní nápady testerů, zvažuji pro další vývoj v kapitole 10.

9.6.1 Opravy chyb a drobná vylepšení

Na základě podnětů od uživatelů aplikace byly provedeny následující úpravy:

- Prvky, na které lze kliknout, jsou od prvků, které na kliknutí nereagují, důsledně odlišeny pomocí kurzoru myši při přechodu nad danou oblastí. Do budoucna je potřeba zvážit, jestli by nebylo vhodné zavést ještě výraznější vizuální odlišení.
- Ignorování velikosti zadaných znaků při použití vyhledávací lišty. Jeden uživatel upozornil, že by toto bylo vhodné zejména pro němčinu, ve které se často objevují velká písmena na začátku slov.
- Nastavení maximální výšky pro dialogové okno tak, aby v případě dlouhého obsahu místo přetečení přes okraj obrazovky zobrazilo posuvník.
- Oddělování více významů slov při výpisu v uživatelském slovníku. Jednalo se o chybu, kdy více pojmů bylo ve výpisu slito v jeden.
- Umožnění vkládání obrázků i v jiných formátech než JPEG.



Obrázek 9.3: Využití vyrovnávací paměti APC v průběhu testování

9.6.2 Zrušení našeptávání v části psané výuky

V komentářích na závěr testování se uživatelé jednohlasně shodli, že jim napovídání slov při psaném způsobu výuky nevyhovuje. Považují to za příliš velké usnadnění výuky a argumentují tím, že takto si student neprocvičí hláskování slov. Na základě těchto názorů jsem se rozhodl napovídání při psané výuce zrušit.

Dále jsem podle poznámky z jednoho uživatelů umožnil nezadávat při této výuce žádný text v případě, že vůbec nevíme, jak odpovědět. Pokud tedy odešleme prázdné textové pole, odpověď bude vyhodnocena jako nesprávná.

9.6.3 Nasazení vyrovnávací paměti APC

Testování bylo prováděno na nevýkonném virtuálním serveru, na kterém je provozováno několik dalších aplikací. Všichni uživatelé navíc testovali systém zhruba ve stejnou dobu a odezvy serveru byly místy velmi pomalé. Z tohoto důvodu jsem se rozhodl na server nasadit vyrovnávací paměť APC[4].

Do této vyrovnávací paměti byly automaticky ukládány veškeré PHP skripty zkompilované na tzv. *opcode*[5]. V produkčním prostředí jsem navíc nastavil cachování databázových dotazů systému Doctrine 2 právě do této cache.

Jak jsem později zjistil při zkoumání vytížení serveru, pomalé odezvy aplikace nebyly způsobeny nízkým výkonem serveru nebo neefektivním zdrojovým kódem, ale pouze chybou v nastavení DNS záznamů. Po opravě začala aplikace reagovat svižně, nicméně nasazení APC rozhodně nebylo zbytečné. Jak můžeme vidět na grafu 9.3, který byl vygenerován zhruba půl hodiny po nasazení APC, již v této době bylo 99,8 % požadavků načítáno z právě z této vyrovnávací paměti.

[4] Alternative PHP Cache

[5] Operation Code – kód, který vznikne po kompilaci PHP skriptů a je již přímo prováděn interpretem PHP.

Kapitola 10

Budoucí vývoj

Testování aplikace na vybrané skupině uživatelů přineslo důležité poznatky, které byly představeny v kapitole 9. Zásadním poznatkem provedených testů je fakt, že aplikace pro vybranou uživatelskou skupinu zřejmě není nezajímavá. S dalším vývojem nad rámec této práce jsem navíc počítal již v začátcích plánování návrhu aplikace.

V této kapitole představím nejen vlastní náměty na další vývoj aplikace, ale i nápady, které vznesli uživatelé při testování aplikace a zmíním také možnosti spolupráce, které byly diskutovány například se společností Lingea, s.r.o.

10.1 Směrování budoucího vývoje

Důležitým předmětem dalšího vývoje bude zejména organizace databáze pojmů a konverzací. Tato práce byla zaměřena především na výukové prostředí, databáze pojmů zde představuje první funkční prototyp a je potřeba její strukturu dále rozvíjet. Hlavním předmětem dalšího vývoje by zde měl být propracovanější systém uživatelského rozšiřování databáze a editace dat. Systém, který by umožnil uživatelské komunitě udržovat kvalitu obsahu například pomocí hlasování o správnosti vložených dat, větší možnosti editace dat a možnost mazat chybná data z databáze. K tomu by zřejmě bylo užitečné vést kompletní historii změn v databázi, podobně jako to často řeší systémy postavené na technologii Wiki.

Dále bych se chtěl zaměřit na výraznější propojení systému s komunitními sítěmi, zejména s aplikací Facebook. Domnívám se, že pokud uživatelé budou mít možnost vidět aktivity svých přátel, intenzitu jejich výuky a zlepšování, bude to sloužit nejen jako motivace pro intenzivnější studium, ale i jako způsob, jak do systému přivést nové uživatele. S tímto záměrem jsem položil účastníkům testování otázku, zda by je aktivity přátel motivovaly k dalšímu studiu, přičemž kladně odpovědělo 60 % dotázaných.

10.2 Náměty na další vývoj na základě testů

Uživatelé měli na závěr testování aplikace prostor vyjádřit libovolné názory na aplikaci, zhodnotit, co se jim líbí a co naopak nelíbí. Byl jsem příjemně překvapen množstvím námětů na rozšíření aplikace o další funkce. Nejzajímavější nápady ze strany uživatelů, jejichž zařazení do systému rozhodně zvážím, jsou následující:

- *Chat, neboli interaktivní konverzace*, je velmi zajímavý nápad jednoho z testerů. Myšlenka je taková, že by systém nabízel uživatelům živé spojení pomocí chatu a to na základě jejich mateřských a studovaných jazyků. Uživatel by se tak mohl zeptat

na určitá fakta přímo rodilého mluvčího, případně s ním v konverzovat v jeho jazyce. Příspěvek dále zmiňoval možnost společné výuky uživatel na dálku v podobě vzájemného zkoušení jednotlivých pojmů.

- *Ovládání výuky pomocí kláves* byl nápad dalšího uživatele, který si všiml, že při delším používání aplikace musí příliš často používat myš. Jelikož klávesové zkratky mohou bez problémů sloužit jako další alternativa k ovládání myší, bude tato funkce určitě zařazena do budoucího vývoje.

- *Okruhy pojmů* pro jednotlivé uživatele zmínilo dokonce více testerů. Ačkoliv jsem se v zájmu co největší jednoduchosti a minimalistického ovládání systému této funkci při návrhu bránil, musím uznat, že při větším počtu pojmů bude zřejmě tato funkce nutná. Otázkou k zamyšlení je také vhodný způsob kategorizace obsahu databáze pojmů.

10.3 Možnosti spolupráce

Při schůzce se zástupci společnosti Lingea, s.r.o. bylo diskutováno mnoho aspektů výuky a problémů, na které lze při dalším vývoji této aplikace narazit. Ačkoliv si někteří ze zástupců této společnosti nebyli jisti, zda bude uživatelům vyhovovat výuka několika jazyků zároveň, byl z jejich strany vyjádřen jistý zájem o aplikaci a ochota v budoucnosti spolupracovat. Podoba databáze pojmů se tedy bude odvíjet také od toho, zda bude či nebude postavena na datech od společnosti Lingea.

Do skupiny testerů tohoto systému jsem kromě běžných uživatelů počítače záměrně vybral také několik studentů informatiky, abych tak získal zpětnou vazbu více technického charakteru. Dva z nich projevili o aplikaci značný zájem a nabídli mi spolupráci na dalším vývoji. Vzhledem k tomu, že rozrůstající se systém je pro jednoho vývojáře čím dál tím těžší udržovat, budu o této nabídce určitě uvažovat.

Kapitola 11

Závěr

Hlavním cílem této práce bylo vytvořit webovou aplikaci, s jejíž pomocí by uživatelé mohli snadno rozvíjet svou slovní zásobu a konverzační schopnosti, a to současně ve všech jazycích, jejichž studiu se věnují. V průběhu vývoje jsem se snažil využít výhod, které prostředí webu nabízí oproti jiným formám aplikace. Systém jsem tak kromě výukové části rozšířil o centralizovanou databázi pojmů a využil možnosti komunikace s jinými webovými službami, konkrétně s weby Wordnik a Facebook.

Tato práce nenavazuje na žádné předchozí projekty.

V tomto textu postupně popisuji kompletní vývojový cyklus aplikace od stanovení základních požadavků na systém a jejich následné analýzy přes návrh architektury systému a uživatelského rozhraní až po popis implementace a metodologii produkčního nasazení aplikace. Významná část práce byla věnována také testování systému s pomocí uživatelů, které bylo zaměřeno především na ergonomii a funkčnost systému, přičemž ale zkoumalo i další aspekty, zejména vhodnost způsobu výuky. Na základě dat získaných pomocí testování byly zhodnoceny dosažené výsledky a v aplikaci byly provedeny opravy nalezených chyb a vylepšeny některé funkce. Závěrečná kapitola se věnuje nápadům a podnětům k dalšímu vývoji a možnostem budoucí spolupráce jak se společností Lingea, s.r.o., tak s dalšími vývojáři.

Z technologického hlediska je základním stavebním prvkem aplikace systém Nette Framework, jehož podrobnému studiu jsem věnoval značné množství času, částečně také z důvodu absence kvalitní dokumentace. Jako hlavní studijní materiály mi posloužily nejen přispěvky v rozsáhlém fóru komunity okolo Nette Framework, ale často také přímé studium zdrojových kódů tohoto systému. Aplikace dále využívá několik dalších frameworků a technologií, které byly podrobněji popsány v tomto textu.

Testování naznačilo, že by tento systém mohl oslovit určitou část uživatel internetu, a proto jsem přesvědčen, že v jeho vývoji má smysl pokračovat. Rád bych tuto aplikaci rozšířil o další funkce a za nějaký čas zprovoznil na internetu jako novou webovou službu. Další prioritou vývoje bude systém pro kontrolu správnosti uživateli přidaných dat a možnost oprav chybně vložených údajů. Celý systém bych navíc rád přiblížil podobě komunitního webu.

Práce pro mě byla velkým přínosem zejména v podobě značného zlepšení znalostí vývoje webových aplikací, o který jsem se vždy aktivně zajímal. Výrazný posun vidím zejména v detailním poznání systému Nette Framework a knihovny Doctrine 2. Zároveň se jedná o největší projekt, jaký jsem dosud vytvořil a zkušenosti, které jsem jeho vývojem získal, pro mě budou velmi užitečné při práci na dalších aplikacích.

Literatura

[1] ANDERSON, J. C.; LEHNARDT, J.; SLATER, N.: *CouchDB: The Definitive Guide*. O'Reilly Media, první vydání, leden 2010, ISBN 78-0-596-15589-6, 272 s.

[2] BOREK, B.: MVC a další prezentační vzory. *Zdroják*, [online]. Květen 2009, [cit. 2011-01-22], ISSN 1803-5620.
Dostupný na:
http://zdrojak.root.cz/serialy/mvc-a-dalsi-prezentacni-vzory

[3] BOREK, B.: Reakce na článek o Nette + MVC. *BorBer*, [online]. Březen 2009, [cit. 2011-01-22].
Dostupný na: http://www.borber.com/blog/reakce-na-clanek-o-nette-mvc

[4] CROFT, J.; LLOYD, I.; RUBIN, D.: *Mistrostvi v CSS*. Computer Press, první vydání, červen 2007, ISBN 978-80-251-1705-7, 409 s.

[5] ČERNÝ, M.: Učte se s flashcards. *Zdroják*, [online]. Únor 2011, [cit. 2011-03-02], ISSN 1803-5620.
Dostupný na: http://www.root.cz/clanky/ucte-se-s-flashcards/

[6] Authentication. *Facebook Developers*, [online]. 2011, [cit. 2011-04-18].
Dostupný na: http://developers.facebook.com/docs/authentication/

[7] GRUDL, D.: Nette Framework: MVC & MVP. *Zdroják*, [online]. Březen 2009, [cit. 2011-01-22], ISSN 1803-5620.
Dostupný na: http://zdrojak.root.cz/clanky/nette-framework-mvc--mvp

[8] GRUDL, D.: Nette Framework: Refactoring. *Zdroják*, [online]. Březen 2009, [cit. 2011-01-25], ISSN 1803-5620.
Dostupný na: http://zdrojak.root.cz/clanky/nette-framework-refactoring/

[9] Codes for the Representation of Names of Languages. *Infoterm*, [online]. Říjen 2010, [cit. 2011-04-15].
Dostupný na: http://www.loc.gov/standards/iso639-2/php/code_list.php

[10] JANOVSKÝ, D.: Optimální tvar URL. *Jak psát web*, [online]. [cit. 2011-04-15].
Dostupný na: http://www.jakpsatweb.cz/seo/seo-url.html

[11] MALÝ, M.: „Prostě to tam nahrajte FTPčkem" – nebo ne? *Zdroják*, [online]. Leden 2011, [cit. 2011-04-15], ISSN 1803-5620.
Dostupný na:
http://zdrojak.root.cz/clanky/proste-to-tam-nahrajte-ftpckem-nebo-ne

[12] MAREK, J.: WebLoader. *Nette Framework Addons*, [online]. Březen 2010, [cit. 2011-01-25].
Dostupný na: http://addons.nette.org/cs/webloader

[13] Učební metody. *Memostation*, [online]. 2008, [cit. 2011-02-04].
Dostupný na: http://www.memostation.net/cs/ucebni-metody

[14] MICHELOUD, F.: How to use flash cards to improve your vocabulary using time you did not know you had. *How to learn any language*, [online]. Duben 2009, [cit. 2011-01-26].
Dostupný na:
http://how-to-learn-any-language.com/e/guide/flash-cards.html

[15] Slovníček pojmů - Model a modul. *Nette Framework*, [online]. 2010, [cit. 2011-01-24].
Dostupný na: http://doc.nette.org/cs/slovnik#toc-model-a-modul

[16] Nette\Application\Control. *Nette Framework*, [online]. 2010, [cit. 2011-01-25].
Dostupný na: http://doc.nette.org/cs/nette-application-control

[17] Nette\Application\Presenter. *Nette Framework*, [online]. 2010, [cit. 2011-01-25].
Dostupný na: http://doc.nette.org/cs/nette-application-presenter

[18] NIELSEN, J.; TAHIR, M.: *Homepage Usability: 50 Websites Deconstructed*. New Riders Publishing, první vydání, 2002, ISBN 0-7357-1102-X, 315 s.

[19] PROCHÁZKA, F.: Model: Entity-Repository-Mapper. *Nette Framework*, [online]. Duben 2011, [cit. 2011-04-08].
Dostupný na:
http://wiki.nette.org/cs/cookbook/model-entity-repository-mapper

[20] ROLOFF, J.; ROLOFF, C.: Browser Specs. *978 Grid System*, [online]. Leden 2011, [cit. 2011-04-08].
Dostupný na: http://978.gs/browsers/

[21] Metoda sugestopedie. *Slunečnice*, [online]. 2011, [cit. 2011-02-02].
Dostupný na: http://www.sugestopedie.cz/metoda-sugestopedie/

[22] TICHÝ, J.: Pět vrstev modelu. *Jan Tichý*, [online]. Duben 2010, [cit. 2011-04-08].
Dostupný na: http://www.phpguru.cz/clanky/pet-vrstev-modelu

[23] Study skills. *Wikipedia*, [online]. 2011, [cit. 2011-01-26].
Dostupný na: http://en.wikipedia.org/wiki/Study_skills

[24] List of flashcard software. *Wikipedia*, [online]. 2011, [cit. 2011-02-02].
Dostupný na: http://en.wikipedia.org/wiki/List_of_flashcard_software

Příloha A

Obsah CD

Na přiloženém CD se nalézají následující adresáře:

- `aplikace` – Zdrojové kódy vytvořené aplikace.
- `dokumentace` – Stručná uživatelská příručka k aplikaci.
- `pisemna-zprava-pdf` – Tato písemná zpráva ve formátu PDF.
- `pisemna-zprava-src` – Zdrojové kódy této písemné zprávy v jazyce LaTeX.
- `testovani` – Testovací data a kompletní výsledky testů.
- `video` – Krátká demonstrace aplikace v podobě videa.