



TECHNICKÁ UNIVERZITA V LIBERCI  
Fakulta mechatroniky, informatiky  
a mezioborových studií ■

# Pokročilý třídní jazykový slovníček

## Diplomová práce

*Studijní program:* N2612 – Elektrotechnika a informatika

*Studijní obor:* 1802T007 – Informační technologie

*Autor práce:* **Bc. Daniel Maděra**

*Vedoucí práce:* Ing. Jana Vítvarová, Ph.D.





TECHNICAL UNIVERSITY OF LIBEREC  
Faculty of Mechatronics, Informatics  
and Interdisciplinary Studies ■

# Advanced Class Language Dictionary

## Master thesis

*Study programme:* N2612 – Engineering and Informatics

*Study branch:* 1802T007 – Information Technology

*Author:* **Bc. Daniel Maděra**

*Supervisor:* Ing. Jana Vitvarová, Ph.D.



## ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Daniel Maděra**  
Osobní číslo: **M14000170**  
Studijní program: **N2612 Elektrotechnika a informatika**  
Studijní obor: **Informační technologie**  
Název tématu: **Pokročilý třídní jazykový slovníček**  
Zadávající katedra: **Ústav mechatroniky a technické informatiky**

### Z á s a d y p r o v y p r a c o v á n í :

1. Proveďte rešerši existujících řešení pro učení se a testování slovíček z cizích jazyků.
2. Navrhněte aplikaci jazykového slovníčku, která umožní studentům třídy motivující a personalizované procvičování slovíček z konkrétní učebnice.
3. Prozkoumejte a navrhněte způsoby, jak zjednodušit a automatizovat import požadovaných slovíček ve zvukové, obrazové a textové podobě.
4. Prozkoumejte metody vhodné na generování testovacích slovíček v závislosti na výsledcích předchozích testů a případně dalších parametrech.
5. Jazykový slovníček implementujte.
6. Slovníček naplňte slovíčky k vybrané cizojazyčné učebnici a otestujte.

Rozsah grafických prací: dle potřeby dokumentace

Rozsah pracovní zprávy: 40–50 stran

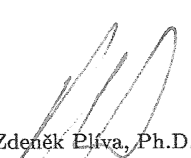
Forma zpracování diplomové práce: tištěná/elektronická

Seznam odborné literatury:


- [1] Russell S., Norvig P.: Artificial Intelligence: A Modern Approach, Pearson Education Limited, 2013, ISBN-13: 978-1292024202
- [2] Richardson L., Ruby S.: RESTful Web Services: Web services for the real world, O'Reilly Media, 2007, ISBN-13: 978-0596529260

Vedoucí diplomové práce: **Ing. Jana Vitvarová, Ph.D.**  
Ústav mechatroniky a technické informatiky

Datum zadání diplomové práce: **10. října 2016**  
Termín odevzdání diplomové práce: **15. května 2017**

  
prof. Ing. Zdeněk Pliva, Ph.D.  
děkan



  
doc. Ing. Milan Kolář, CSc.  
vedoucí ústavu

V Liberci dne 10. října 2016

## Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí vloženou do IS STAG.

Datum: 3. 1. 2014

Podpis:



## Poděkování

Rád bych touto cestou vyjádřil poděkování za trpělivost, užitečné rady a vedení práce Ing. Janě Vitvarové, Ph.D. Rovněž bych rád poděkoval rodičům za podporu při psaní této práce a za umožnění studia.

## **Abstrakt**

Tato práce se zabývá problematikou IT podpory učení slovní zásoby cizího jazyka. Byla vytvořena aplikace jazykového slovníčku v podobě webové služby, která má umožnit žákům třídy personalizované a motivující učení a testování školních slov v různých podobách. Frontend webové služby je tvořen JavaScriptovou single-page aplikací s podporou knihovny React, backend je tvořen webovým API v jazyce Python za použití Django frameworku.

## **Klíčová slova**

učení slovíček, rozložené opakování, webové API, SPA, React, Django

## **Abstract**

This thesis deals with the issues of IT support during learning foreign language vocabulary. A web service was developed as a part of the thesis. It serves as an personalized and motivating vocabulary tool which helps students to learn and practice words in various forms. A frontend of the web service is a JavaScript single-page application using the React library. A backend is a web API in Python using the Django framework.

## **Keywords**

vocabulary learning, spaced repetition, web API, SPA, React, Django

# Obsah

<b>1 Úvod</b>	<b>10</b>
<b>2 Analýza</b>	<b>11</b>
2.1 Hlavní cíle . . . . .	11
2.2 Existující řešení . . . . .	12
2.3 Učení slovíček . . . . .	14
2.4 Testování slovíček . . . . .	15
2.5 Typy testů . . . . .	16
2.6 Metoda rozloženého opakování . . . . .	17
2.7 Specifikace požadavků . . . . .	18
<b>3 Návrh aplikace</b>	<b>20</b>
3.1 USE-CASE diagram . . . . .	20
3.2 Uživatelské role . . . . .	20
3.3 Správa učebnic . . . . .	21
3.4 Správa slovíček . . . . .	22
3.5 Obtížnost slovíček . . . . .	23
3.6 Generování slov pro procvičování . . . . .	25
3.7 Procvičování slovíček . . . . .	26
3.8 Připomínání slov . . . . .	28
3.9 Motivace . . . . .	29
3.10 Architektura aplikace . . . . .	30
<b>4 Klientská část</b>	<b>32</b>
4.1 Návrh aplikace . . . . .	32
4.2 Vývojové prostředí . . . . .	36
4.3 Knihovna React . . . . .	37
4.4 Testování . . . . .	39
<b>5 Serverová část</b>	<b>42</b>
5.1 Technologie . . . . .	42
5.2 API (Application Interface) . . . . .	43
5.3 Zabezpečení . . . . .	45
5.4 Databáze . . . . .	47
5.5 Testování . . . . .	48
<b>6 Závěr</b>	<b>50</b>
<b>Použitá literatura</b>	<b>51</b>
<b>Přílohy</b>	<b>53</b>



## Seznam obrázků

1	Naznačení rozloženého opakování na základě křivky zapomínání . . . . .	18
2	USE-CASE diagram aplikace . . . . .	20
3	Ukázka struktury API . . . . .	33
4	Adresářová struktura klientské aplikace . . . . .	40
5	Ukázka části databázového modelu zodpovědné za správu slovíček . . . . .	48
6	Vyhodnocování odpovědi v procvičování slov . . . . .	54
7	GUI testovací stránky . . . . .	55
8	GUI výběr slov pro testování . . . . .	55
9	GUI tvorby slov v učebnici . . . . .	55

## Seznam tabulek

1	Používání slov v britské angličtině . . . . .	14
2	Matice koeficientů pro výpočet optimálních intervalů . . . . .	29

## Seznam ukázek zdrojových kódů

1	Příklady použití Levenshteinovy vzdálenosti . . . . .	34
2	Struktura výstupního souboru z první části importu . . . . .	36
3	Příklad komponenty pro zobrazování nápovědy při procvičování slovíček . . . . .	38
4	Hlavní renderovací funkce aplikace . . . . .	39
5	Serializační třída k transformaci dat o učebnici . . . . .	44
6	Uložení zvukové nahrávky . . . . .	45
7	Obsah metody <i>jwt_response_payload_handler</i> . . . . .	46
8	Ukázka třídy s přizpůsobeným oprávněním . . . . .	47
9	Implementace algoritmu pro výpočet Levenshteinovy vzdálenosti . . . . .	53

## Seznam zkratek

**AJAX** Asynchronous JavaScript and Extensible Markup Language (XML).

**API** Application Interface.

**CORS** Cross-Origin Resource Sharing.

**CSRF** Cross Site Request Forgery.

**CSS** Cascading Style Sheets.

**CSV** Comma Separated Values.

**DOM** Document Object Model.

**ES6** ECMAScript2015.

**HATEOAS** Hypermedia as the Engine of Application State.

**JS** JavaScript.

**JSON** JavaScript Object Notation.

**JSX** XML-Like Syntax Extension.

**JWT** JavaScript Web Token.

**MitM** Man in the Middle.

**MTV** Model Teplate View.

**MVC** Model View Controller.

**NPM** Node Package Manager.

**ODBC** Open Database Connectivity.

**ORM** Object-Relational Mapping.

**RAML** RESTful API Modeling Language.

**SEO** Search Engine Optimization.

**SPA** Single Page Application.

**URL** Uniform Resource Locator.

**XML** Extensible Markup Language.

**XSS** Cross-Site scripting.

# 1 Úvod

Výuka cizích jazyků je pro aktuální globální společnost jedna z nejzásadnějších otázek, ať už se jedná o pracovní příležitosti v zahraničí, získávání informací nebo sociální problematiku světa. Základním pilířem rozvoje cizího jazyka je slovní zásoba.

Žáci si při výuce cizího jazyka obvykle vedou vlastní slovníček, do kterého si zapisují nově nabytá slova. V době přípravy na test rychle slova zopakují a následně z krátkodobé paměti test napíší, což má za následek, že pokud naučená slova dále neopakují, dojde k jejich zapomenutí. Navíc se postupně slovíčka hromadí a není v silách žáka je opakovat.

Existuje řada řešení, které pomáhají rozvíjet slovní zásobu cizího jazyka. Problémem ale je, že tyto řešení žákům neposkytují přizpůsobení úrovni ani obsahu. Žáci se učí jinou slovní zásobu než tu, kterou aktuálně procházejí ve škole.

Z výše uvedených důvodů vznikl námět pro tuto práci. Cílem diplomové práce je provést analýzu, jak zefektivnit učení a motivovat žáky k procvičování slovíček. Na základě zjištěných poznatků navrhnout a implementovat aplikaci, která umožní rozvíjet cizojazyčnou slovní zásobu. A vyučujícím poskytne nástroj pro správu a prezentaci slovíček pro žáky.

Úvodem se práce zabývá rešerší existujících řešení podobných aplikací. Analýza rovněž zahrnuje shrnutí informací o rozvoji slovní zásoby cizího jazyka v podobě vybraných metod a technik k motivaci učení a k efektivnímu zapamatování slov. Na základě zjištěných informací z rešerše byla definována specifikace požadavků, které slouží jako hlavní podklad pro návrh aplikace v kapitole 2. V této kapitole jsou rovněž detailněji popsán průběh importu slov, způsoby generování testovací sady a vybrané metody zajišťující motivaci žáků. V kapitole 3 a 4 se práce zabývá implementací navržené slovníkové aplikace v podobě webové služby. Kapitola 3 se věnuje klientské části a kapitola 4 serverové. V obou jsou představeny technologie, které jsou využívány pro vývoj, jejich implementace a průběh testování.

## 2 Analýza

### 2.1 Hlavní cíle

Hlavním cílem aplikace je připravit žáky na hodinu cizího jazyka a zároveň přirozeně rozvíjet slovní zásobu tak, aby studenti neztratili motivaci a chuť k poznávání nových výrazů. Dále také umožnit žákům se otestovat a ověřit, zda naučenou sadu slov ovládají. Aplikace by se měla adaptovat na zdatnost a úroveň každého ze žáků. Vyučujícím by aplikace měla usnadnit správu a import slovíček, která třída má umět v rámci dané učebnice a následně předkládat žákům vhodnou slovní zásobu, například jako přípravu pro následující lekci ve škole.

#### 2.1.1 Personalizace

Důležitým cílem aplikace by měla být adaptace (personalizace) aplikace podle potřeb jednotlivých žáků a celých tříd. Adaptace na úrovni žáka znamená, přistupovat k němu individuálně na základě jeho úrovně, znalostí a dovedností. Umožnit mu jít vlastním tempem a předkládat mu slovíčka na základě předchozích výsledků. V případě konkrétní třídy jde o individuální přístup, a to zejména ve výběru sady slov.

#### 2.1.2 Motivace

Důležitou součástí obecného vzdělávání jsou motivace, dělí se na vnitřní a vnější nebo pozitivní a negativní [7]. Analýza bude zaměřena pouze na motivační prostředky, které lze zařadit do aplikace na procvičování slovíček.

Motivace založená na základě vlastních úspěchů je důležitá pro utvrzení sebevědomí žáků. Ocenění v případě zvládnutí sady slov nebo gramatického bloku lze v případě aplikace implementovat například hláškami s projevem pochvaly nebo jiným upozorněním na dosažený výsledek. Zajímavým prvkem v aplikaci by mohlo být i herní prostředí. Žáci si rádi hrají a již J. A. Komenský poukázal na důležitost her ve vzdělávacím procesu.

Jedním z dalších stěžejních faktorů motivace je kolektiv. Právě díky kolektivu, ve kterém funguje přirozená rivalita, jsou žáci schopni dosáhnout mnohem vyšších výsledků, než kdyby se vzdělávali odděleně a samostatně. Rivalita a soutěživost se může projevovat i velmi negativním způsobem. Místo kamarádských vztahů mezi dětmi mohou vznikat nepřátelské vazby, ve kterých dochází k posměchu méně nadaných dětí, a ty se potom necítí v kolektivu dobře. Zajímavější motivací pro kolektiv je například vidina společně dosažených výsledků. V případě učení slovíček by to byl počet slovíček naučených v rámci celé třídy. Dochází zde k utužování kolektivu a děti by mohlo těšit to, že nějakým způsobem přispívají k úspěchům celé třídy.

### 2.1.3 IT podpora

Posledních několik let se společnost ubírá trendem informačních technologií. Každý ze žáků má už od útlého věku přístup k počítači nebo k chytrému telefonu. Orientace a schopnost tato zařízení používat není pro děti žádný problém. Přirozeně se tedy tato zařízení postupně stávají součástí každodenní přípravy žáka na následující školní den. V některých případech plně nahrazují klasické učebnice a jsou přímo začleněny do výuky. Použití informačních technologií má za následek i zlepšení motivace při výuce. Obecně je známo, že žáci raději studují slovíčka interaktivní formou hádanek, křížovek nebo her, než nekonečných seznamů slov.

Tato zařízení umožňují interaktivní výuku, kde lze využít nejen textových, ale také obrázkových a zvukových prostředků pro lepší zasazení nově nabytých vědomostí do kontextu.

## 2.2 Existující řešení

Na trhu lze nalézt nepřeberné množství aplikací pro výuku cizích jazyků. Řada z nich jsou téměř komplexní systémy, které provází studenta od základních frází a slovíček až po gramatické standardy cizího jazyka. Analýza existujících řešení byla zaměřena na aplikace, které se zabývají především testováním slovíček a frází.

Do analýzy existujících řešení byly zahrnuty tři desktopové, jedna webová a jedna mobilní aplikace.

### 2.2.1 TS Angličtina

Z analyzovaných řešení se jevila desktopová aplikace TS Angličtina od firmy Terasoft ta nejlépe funkčně zpracovaná. Tato firma se zabývá širokou škálou výukových nástrojů se zaměřením na základní školy. V případě cizích jazyků se zabývají výukou anglického a německého jazyka. Hlavní předností aplikace je podpora nejvíce používaných učebnic cizího jazyka. Aplikace umožňuje testování různými způsoby - psaný překlad slova, porozumění mluvenému slovu, vybírání správných významů nebo doplňování vynechaných slov [1]. Analýza byla tvořena pouze z webových informací o produktu od vydavatele. Bohužel firma Terasoft neposkytuje DEMO nebo TRIAL verzi aplikace, která by hlubší analýzu umožňovala.

### 2.2.2 Langsoft Teacher

Dalším analyzovaným řešením byla aplikace Langsoft Teacher, která je dostupná nejen v podání desktopové aplikace, ale také jako mobilní aplikace pro platformy iOS a Android. Aplikace je velmi komplexní, obsahuje různé moduly pro testování například v obrazové

formě pro předškolní děti. Zajímavá vlastnost, kterou aplikace disponuje, je pamatování problematických slov a nabízení těchto slov častěji než těch bezproblémových. Dále program umožňuje automaticky rozšiřovat slovní zásobu, která je v testování zahrnuta [2].

### 2.2.3 Duolingo

Jedná se o mobilní aplikaci pro Android. Zahrnuje učivo cizího jazyka od základních komunikačních frází až po tvorbu gramaticky složitějších vět. V aplikaci je připravena dlouhá řada cizích jazyků - němčina, angličtina, španělština, italština a další. Chybí ale více referenčních neboli mateřských jazyků. Aktuálně lze využít pouze angličtinu. Aplikace kromě standardních funkcí zahrnuje i rozpoznávání mluvených odpovědí [3]. Zajímavým poznatkem byl systém motivace uživatelů, kde si každý mohl pozvat své přátele, mezi kterými docházelo k sdílení dosažených výsledků. Dalším motivačním prvkem bylo udržení plánu pravidelného testování, jelikož v opačném případě docházelo k automatickému zvyšování objemu testovacích dat [4]. Nevýhodou aplikace byla nutnost připojení k internetu. V případě použití mobilních dat, docházelo ke zpoždění zejména při rozpoznání slov.

### 2.2.4 Vocabulary Trainer

Vocabulary Trainer je webová aplikace zdarma, napsaná v jazyce PHP, která naučí 5000 nejvíce používaných slov daného jazyka. Aplikace umožňuje různá nastavení. K dispozici je i řada jazyků včetně češtiny a to jako referenční i jako učený jazyk. Testování spočívá nejdříve ve čtení slov. A následně uživatel vybírá odpověď z daným možnostmi. Aktuálně překládané slovo si lze kdykoliv přehrát různou rychlostí. Jako motivační prvek aplikace využívá jednoduchý bodový systém. Součástí je i kalendář s emailovou upomínkou k dalšímu testování. Program je propracovaný, ale poměrně pomalý a dlouho trvá zejména úvodní načítání dat. Její výrobce LanguageCourse S.L. poskytuje i mobilní aplikaci pro Android k učení anglických slov a frází [5].

### 2.2.5 Supermemo aplikace

Za zmínku ještě stojí aplikace Supermemo. Není to aplikace s připravenými daty k testování slov cizího jazyka, ale slouží čistě jako šablona pro testování jakéhokoliv druhu otázek. Aplikace implementuje algoritmus Supermemo, který je založen na metodě postupného zvyšování intervalu dotazování na naučené informace [6]. Při každé odpovědi program spočítá, kdy by si uživatel měl danou otázku zopakovat tak, aby odpověď na ni nezapomněl a zároveň se co nejvíce zvyšoval interval mezi aktuální a předchozí odpovědí. Aplikace se adaptuje na schopnosti uživatele, v přiměřeném měřítku buď zvyšuje nebo snižuje interval dalšího připomenutí.

Z výše uvedených aplikací až na Supermemo žádná neumožňuje personalizovaný výběr učiva. Tedy nelze vložit vlastní slovíčko nebo si určit sadu slov pro testování. Proto jsou tyto aplikace především cílené pro uživatele, kteří vnímají výuku jazyka jako samostatné vzdělávání sami sebe. Pro žáky, kteří absolvují lekce z cizího jazyka ve škole je tento typ vzdělávání nevyhovující, jelikož se musí učit dvě nezávislé skupiny slov. Sice dochází k rozvinutí slovní zásoby studenta i do jiných okruhů než je jeho učebnice. Ale málokterý student má ještě energii, časovou dotaci a vlastní iniciativu na to, aby se připravoval na školní test ze slovíček a ještě rozvíjel samostatně svoji cizojazyčnou slovní zásobu.

## 2.3 Učení slovíček

Standardně učení slovní zásoby cizího jazyka je založeno na častém opakování slov. Dle výzkumu Biemillera a Boote lze ročně zvládnout až 400 slov u studentů 3.—6. tříd [8]. Což je v případě cizího jazyka poměrně velké číslo, ale problém je, do jaké míry je slovo správně ukotveno v dlouhodobé paměti. Porovnání, kolik je potřeba slov pro ovládnutí anglického jazyka, usnadní následující tabulka 1, která zahrnuje procentuální využití nejvíce používaných slov v každém z odvětví [10]. Z uvedeného výzkumu vyplývá, že není důležitá kvalita ale kvantita slovíček. Pro naši aplikaci budeme předpokládat, že správný výběr slovíček je zaručen výběrem učebnice.

	Konverzace	Noviny	Akademický text
1. 1000 slov	84,3%	75,6%	73,5%
2. 1000 slov	6%	4,7%	4,6%
Akademické výrazy	1,9%	3,9%	8,5%
Ostatní	7,8%	15,7%	13,3%

Tabulka 1: Používání slov v britské angličtině

Je všeobecně známo, že informace se lépe zapamatují, pokud jsou přijímány ve více podobách. Pro učení slovíček se jedná o textovou, zvukovou a obrazovou podobu.

### 2.3.1 Problematika obtížnosti

Velkým problémem v učení slov je přizpůsobit obtížnost každému ze žáků individuálně. Jelikož ne všichni mají stejnou úroveň znalostí cizího jazyka a každý potřebuje jiné tempo pro zapamatování sady slov. Jsou žáci s výbornou pamětí, kterým stačí si slova pouze jednou přečíst a dokáží je používat. Ale jsou i žáci, kterým nestačí je pětikrát zopakovat. Dalším problémem je obtížnost z pohledu jednotlivých slov. Každé slovo má odlišnou obtížnost, kterou lze soudit například podle míry používanosti v jazyce, délky slova, zdali obsahuje přehlasování, dvojité písmena nebo podobnost s referenčním slovem.

### 2.3.2 Učení slov v kontextu

Pro učení slov cizího jazyka je rovněž důležité správné zasazení významu slova do kontextu. Dle průzkumu Biemillera a Bootea z roku 2006 bylo zjištěno, že u žáků od 10—13 let docházelo k nárůstu zapamatovaných slov o 4%, pokud byla slova předložena v kontextu vět. Důležitým poznatkem z tohoto průzkumu je, že žáci si nejen déle naučené slovo pamatovali, ale správně ho i interpretovali, když ho měli za úkol svými slovy vysvětlit [8]. Ve stejném průzkumu rovněž docházelo ještě k vyššímu zlepšení v učení, kdy si žáci zapisovali vlastními slovy definici a použití slova.

### 2.3.3 Aktuální způsob učení

Dle vlastního průzkumu žáci základních škol nevyužívají k učení slovíček. Většinou si udržují vlastní slovníček, do kterého si poznamenávají nově nabytá slova. Při učení zakrývají část s cizojazyčnými slovy a na základě českého ekvivalentu se snaží vybavit překlad slova. Tato metoda neposkytuje prakticky žádnou zpětnou vazbu. Žáci většinou pouze do krátkodobé paměti uloží slova, později při testu rychle zodpoví a následně během pár hodin si na slovo už ani nevzpomenou. Nevýhod má tato metoda celou řadu, ale jednou z klíčových vad je, že si žáci nevytvoří dostatečně souvislostí, aby řádně ukotvily slovo v paměti. Velice oblíbená metoda jsou kartičky s obrázky s českým a cizojazyčným překladem. Využívají i pokročilejších technologií, ale tvoří to vždy pouze doplněk k výuce.

## 2.4 Testování slovíček

### 2.4.1 Aktivní a pasivní slovní zásoba

Slovní zásobu, kterou využíváme k tvorbě vět, ať už v cizím nebo mateřském jazyce, rozdělujeme na dvě skupiny - aktivní a pasivní. Pasivní zásoba je sada slov, která jsou pevně a spolehlivě uložena v naší paměti. Průměrný žák zná cca 50 000 výrazů. Velikost pasivní zásoby je ovlivněna věkem, vzděláním a četbou. Slova ze této sady využíváme zejména při písemné formě, ať už se jedná o čtení nebo psaní. Aktivní zásoba je sada slov, ve které lze najít žádané slovo během desítek milisekund. U většiny lidí dosahuje velikosti 4 000 až 8 000 výrazů [12]. Slovo se díky používání dostává z pasivní do aktivní slovní zásoby.

### 2.4.2 Metody testování

Metod testování slovní zásoby je mnoho. Základní rozdělení je na pasivní a aktivní metody. V případě pasivních metod se jedná například o výběr z nabídnutých možností. Jde o případ, kdy student nemusí znát přesnou odpověď, ale dokáže otázku vyhodnotit



správně vylučovací metodou. Při aktivním testování žáci musí odpověď znát, aby otázka byla vyhodnocena správně. Pasivní metoda má pozitivní vliv například na motivaci žáka, který u ní tolik netápe a za pomoci zdravého rozumu může test vyhodnotit správně. Problém ale vzniká při používání získaných a procvičených informací. V případě cizího jazyka lze pasivní slovní zásobu využít pro čtení a náslech, ale pro ovládnutí cizího jazyka je nedostatečná a tudíž nevhodná.

Dalším rozdělením pasivního a aktivního testování slovíček je rozpoznávání (*recognition*) a vzpomnutí (*recall*). Rozpoznávání spočívá v předložení cizího slova žákovi, který pro správné zodpovězení musí najít český ekvivalent. Vzpomnutí je způsob testování opačným způsobem než rozpoznání. Uživateli je předložen výraz v jeho mateřském jazyce a on musí nalézt správný výraz v cizím. Rozpoznání je z principu jednodušší pro uživatele než vzpomnutí. Lze ho tedy využít rovněž pro zvýšení motivace při procvičování slov. Ale pro ověření, zda je slovo ovládnuté či nikoliv je metoda rozpoznání také nedostatečná.

## 2.5 Typy testů

V analýze existujících řešení bylo procvičování a testování slov interpretováno různými způsoby. Obecně by se typ testů dal rozdělit na tři kategorie - textové testy, multimediální a herní testy.

### 2.5.1 Textové testy

Textové testy se vyskytovaly například jako výběr z více možností nebo spojováním spolu souvisejících významů, jak už ale bylo uvedeno v kapitole 2.4.2, jedná se o rozvíjení pasivní slovní zásoby. Zajímavějším typem testů je doplňování slov do vět a klasický překlad slova. Tyto typy rozvíjejí žádanou aktivní slovní zásobu. V existujících řešeních textové testy byly doplněny zvukovou interpretací cizího slova.

### 2.5.2 Multimediální a herní testy

Zajímavým využitím multimédií by mohlo být zahrnutí otestování výslovnosti. Tedy možnost nahrání odpovědi a následně by došlo k rozpoznání. Ale analýza a rozpoznání řeči není nikterak triviální záležitost. Dalším zajímavým řešením byly herní testy. Díky nim docházelo k zvýšení motivace uživatelů. Problém je, že tento typ procvičování většinou není zas tolik efektivní, jelikož si uživatel procvičí například v případě doplňování písmen do křížovky nebo známé hry *Hangman* pouze pár slov za poměrně dlouhý čas.

## 2.6 Metoda rozloženého opakování

Až na výjimečné paměti je obecně známo, že pokud chceme informaci v paměti uchovat dlouhodobě, je nutné ji opakovaně připomínat. Pokud informace není vyžívána, s největší pravděpodobností dojde ke ztrátě této informace. V případě, že uživatel s touto informací pracuje častěji, výrazně navyšuje šance pro její zapamatování. Metoda rozloženého opakování (*spaced repetition*) je učební technika založená na opakovaném připomínání informace se zvyšujícím se intervalem [11]. Na začátku učebního procesu, jsou intervaly krátké například na hodinu, 4 hodiny nebo celý den. S postupem se tyto intervaly mohou zvyšovat až na týdny a měsíce. Ideální systém rozloženého opakování nabídne zopakování informace těsně před tím, než dojde k jejímu eventuálnímu nevzpomenutí.

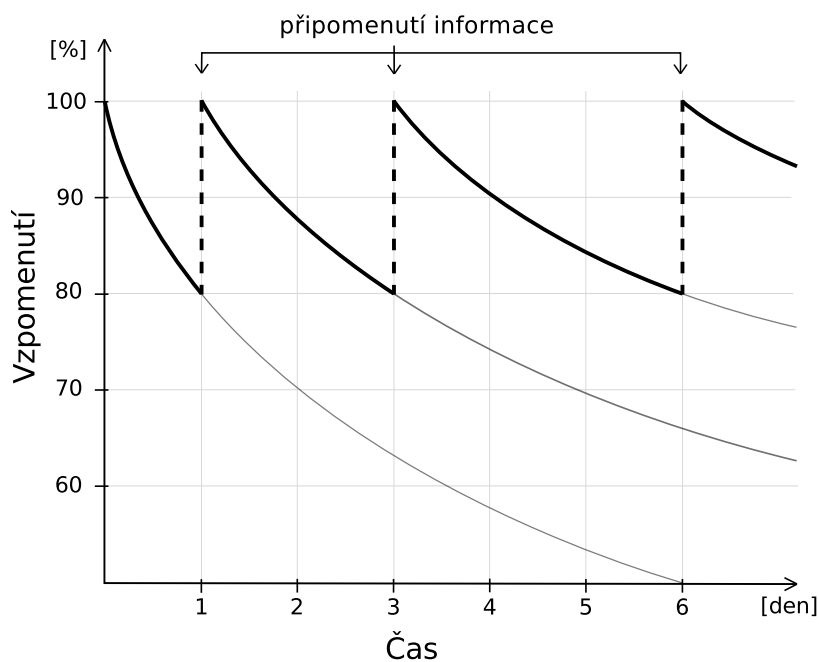
### 2.6.1 Křivka zapomínání

Křivka zapomínání je graf exponenciálního charakteru, který znázorňuje, za jak dlouho a s jakou pravděpodobností je možné si na nově nabytou informaci vzpomenout. První tuto křivku specifikoval už v roce 1885 Hermann Ebbinghaus. Na ose Y lze nalézt procentuální pravděpodobnost vzpomenutí na informaci, na ose X je čas. V čase 0 dojde k nabytí informace a s postupem času dochází k zapomínání. U průměrných pamětí dojde k potencionálnímu zapomenutí nové nesouvisející informace bez kontextu během pěti dní [15].

Následující obrázek 1 znázorňuje metodu rozloženého opakování na křivce zapomínání. V čase s 80% pravděpodobností vzpomenutí na danou informaci dojde k připomenutí. Křivka zapomínání se posune na ose Y a znovu dochází k postupnému zapomínání. Díky exponenciálnímu charakteru interval mezi jednotlivými opakováními narůstá také exponenciálně. V ideálním případě se posune exponenciála tak, že se bude limitně blížit 80% hranici.

### 2.6.2 Leitnerův algoritmus

Na Leitnerův algoritmus lze nahlížet jako na známý učební postup v podobě karet s otázkami. Tyto karty jsou rozděleny do skupin, které jsou označeny například 1 až 3. Kategorii ale obecně může být více. Nová karta je zařazena do první kategorie. Karty v první kategorii jsou opakovány častěji. V poslední kategorii jsou karty opakovány méně často. Karty jsou přemisťovány podle toho, jak uživatel zvládá odpovídat na jednotlivé otázky. V případě, že uživateli otázka na kartě nedělá problémy, přesune kartu do vyšší kategorie. V opačném případě, kdy si uživatel nemůže vzpomenout na danou otázku, karta se přesune do první kategorie a cyklus pro danou kartu začíná znovu.



Obrázek 1: Naznačení rozloženého opakování na základě křivky zapomínání

## 2.7 Specifikace požadavků

Na základě analýzy byla provedena specifikace požadavků, které budou implementovány v aplikaci pro testování slovíček z cizího jazyka.

1. správa učebnic
  - tvořit, editovat a mazat vlastní učebnice
2. správa slovíček v učebnici
  - tvořit, editovat a mazat slovíčka
  - hromadně slovíčka do učebnice importovat
  - částečně automatizovat zvukovou a obrazovou interpretaci slovíčka
3. správa modulů (lekci) a tématických okruhů v učebnici
  - tvořit, editovat a mazat moduly učebnice
  - přiřazovat slovíčka do daných modulů
4. správa uživatelských skupin (tříd)
  - tvořit, editovat a mazat uživatelské skupiny
  - umožnit uživatelům se přihlásit do skupiny
5. správa testovacích sad
  - tvořit, editovat a mazat testovací sady

- umožnit vybírat slova pro testovací sadu z vlastních i veřejných učebnic
- přiřazovat testovací sady ke skupinám uživatelů

#### 6. procvičování slovíček

- vybrat testovací sadu slovíček
- generovat slova na základě úrovně uživatele
- zahrnout textovou, obrazovou a zvukovou interpretaci do procvičování
- možnost uložit stav testování a umožnit pozdější navázání

#### 7. připomínání a procvičování ovládnuté slovní zásoby

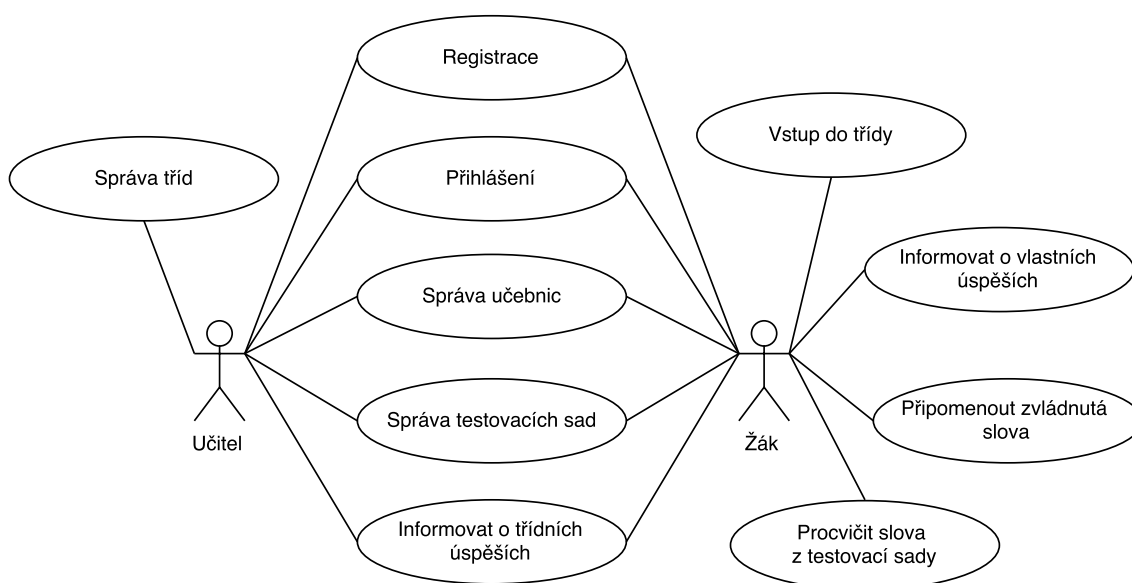
#### 8. motivace

- motivovat v rámci uživatelské skupiny
- motivovat vlastní iniciativu k procvičování

## 3 Návrh aplikace

### 3.1 USE-CASE diagram

Na základě specifikace požadavků byl vytvořen USE-CASE diagram aplikace. Diagram na obrázku 2 znázorňuje pouze obecné bloky funkčnosti aplikace. V následujících kapitolách bude každý z jednotlivých USE-CASE bloků popsán detailněji.



Obrázek 2: USE-CASE diagram aplikace

### 3.2 Uživatelské role

Do aplikace by měly vstupovat dva typy uživatelů - učitel a žák. Učitel z principu má za úkol spravovat testovací sady, učebnice a třídy. Žáci oproti nim mají za úkol vstupovat do svých tříd, procvičovat slova z testovacích dat, které jsou připravené od učitele a připomínat si zvládnutá slova.

Po konzultacích s vedoucím práce a vyučujícím cizího jazyka na základní škole došlo k několika změnám v návrhu právě v oblasti uživatelských rolí a i v principu aplikace. Vznikl požadavek, že aplikace by mohla fungovat spíše jako portál pro vzdělávání a ne jako výukový nástroj pro učitele. V návrhu tudíž vzniklo rovnocenné partnerství mezi učitelem a žákem.

#### 3.2.1 Sjedení uživatelských skupin

Podnětem pro sjednocení uživatelských skupin bylo umožnit žákům procvičovat si slova i v případě, že nejsou součástí žádné třídy. Tak aby se mohl kdokoli do aplikace přihlásit, vybrat si testovací sadu a vyzkoušet svoje znalosti slovíček. Tato funkce umožňuje žákům

volnější přístup k výuce. Nebylo totiž v požadavcích, aby měl učitel přehled o znalostech žáka - právě naopak. Účelem aplikace je poskytnout žákům možnost se samostatně vzdělávat a procvičovat slovní zásobu, a ne vyučujícím poskytnout nástroj pro otestování, zda žák disponuje požadovanými znalostmi.

### 3.3 Správa učebnic

Z důvodu personalizace slovíček bylo nutné v návrhu aplikace zařadit učebnice, díky nimž si žáci budou procvičovat pouze slovíčka, která jsou pro ně aktuální ve výuce cizího jazyka. Učebnice obsahují moduly a v modulech se nacházejí jednotlivá slova. Při tvorbě učebnice bude zvolen cizí jazyk, pro který je učebnice připravená. V rámci jedné učebnice jsou slova unikátní. Tzn. do konkrétní učebnice nemohou patřit dvě stejná slova. Majitel učebnice je může editovat, vytvářet a mazat i v případě, že je využívána některým z ostatních uživatelů.

V aplikaci se může vyskytovat i více stejných učebnic. Například dva různí vyučující vedou své hodiny cizího jazyka podle stejné učebnice, ale každý si chce přizpůsobit sadu slov po svém. Proto se v aplikaci musí učebnice identifikovat majitelem a názvem učebnice.

#### 3.3.1 Sdílení učebnic

Jelikož aplikace slouží jako portál pro vzdělávání, ve výchozím stavu jsou učebnice a slovíčka v nich veřejně přístupná jakémukoliv uživateli. Příchozí uživatel si tedy bude moci buď vytvořit vlastní učebnici se svými slovy, které si chce procvičit, nebo si může vybrat slova už z vytvořených učebnic. Díky sdílení si nový uživatel může prakticky okamžitě po přihlášení začít procvičovat slova bez nutnosti tvorby učebnice. Tento přístup je cílený právě především pro žáky, kteří si chtějí samostatně procvičovat slova a nepatřit do žádné konkrétní třídy.

#### 3.3.2 Import slov do učebnice

Pro usnadnění zadávání dat je v aplikaci umožněno importovat celou sadu slovíček z učebnice. Původní myšlenkou bylo vytvořit parser, který bude schopný přečíst slovíčka přímo z učebnice ve PDF formátu. Po konzultaci s vyučujícím bylo rozhodnuto, že postačí import v podobě textového formátu odděleného čárkou. Důvodem je, že vyučující potřebují slovíčka postupně upravovat a doplňovat. Proto je pro ně sada slovíček v učebnici nevyhovující a spravují si vlastní slovníček a to nejčastěji ve formátu XLS aplikace Microsoft Excel. Z této aplikace není problém vyexportovat slova do textového formátu, kde hodnoty jsou oddělené čárkou.

### 3.4 Správa slovíček

Jednotlivá slovíčka se budou ukládat do modulů učebnice. Slovíčko se bude charakterizovat - překladem v cizím jazyce, významem v mateřském jazyce, definicí slova v cizím i mateřském jazyce a použitím ve větách z učebnice. Dalšími atributy, kterými slovíčko disponuje, jsou obtížnost a slovní druh. Slovíčko bude možné doplnit o multimediální interpretaci a to v podobě zvuku a obrázku.

#### 3.4.1 Textová forma

Textová forma slovíčka spočívá v překladu slova a jeho významu v mateřském jazyce. Pro zjištění významu slova se může využít již hotová aplikace. Například Google Translate poskytuje aplikační rozhraní pro překlady slov i celých vět. Jelikož se aplikace bude zabývat aktivním rozpoznáním a vzpomínáním, je vhodné neautomatizovat významy slov. Učebnice nebo učitelé se mohou s Google Translate lišit ve významech slov. Dalším problémem je cena využívání překladů z Google Translate, která je účtována měsíčně \$20 za 1 milion znaků [14].

#### 3.4.2 Zvuková forma

Do aplikace by měla být implementována i zvuková interpretace slovíčka v cizím jazyce. Zvukové nahrávky mohou být zařazeny do aplikace dvěma způsoby - manuálně vlastními nahrávkami nebo využít Google Text to Speech API, které Google poskytuje zdarma. Aplikace bude záznam lokálně ukládat, aby se co nejvíce omezilo využívání API. Zvuková nahrávka bude získána při importu slovíček do učebnice. Zadávající uživatel si bude moci zvolit, k jakému slovu chce zvukovou interpretaci.

#### 3.4.3 Obrazová forma

Obrazová interpretace slovíčka bude rovněž získána z Google Custom Search API. Toto aplikační rozhraní není omezené a lze v parametrech omezit vyhledávání pouze na obrázky. Při importu si uživatel zvolí, pro jaká slovíčka chce obrazovou formu. Automatizace v tomto případě není vhodná, jelikož ne všechny slovíčka lze interpretovat jako obrázek. Uživatel si také bude moci zvolit obrázek z několika možností. Po vybrání obrázku bude vytvořena komprimovaná kopie souboru a uloží se rovněž zdroj, odkud je obrázek získán. Tato forma bude v aplikaci sloužit pouze jako nápověda pro žáky, díky níž si mohou ke slovíčku vytvořit hlubší asociaci.

Díky Google Custom Search API lze vyhledávání parametrizovat. Výrazem pro vyhledávání bude referenční slovo v cizím jazyce, jelikož cílový jazyk bude nejspíš anglický,

německý nebo španělský, a všechny tyto jazyky jsou více rozšířené, než ten český. Dalším parametrem vyhledávání je omezení pouze na obrázky s licencí „Volně užívat nebo sdílet“, která umožňuje obrázek zkopírovat a nekomerčně publikovat s uvedením zdroje. Google nezaručuje, pod jakou licencí se obrázek aktuálně na stránkách prezentuje. Proto je uživatel při výběru obrázku vyzván, aby zkontroloval licenci použití.

### 3.5 Obtížnost slovíček

Při zadávání slova do učebnice bude automaticky vyhodnocena jeho obtížnost, která je rozdělena do čtyř kategorií - snadné, střední, těžké a nemožné. Vyhodnocení obtížnosti je založeno na kombinaci několika parametrů slova - jeho délky, podobnosti s českým ekvivalentem a dalšími atributy jako počet přehlasovaných písmen nebo počet výskytů dvojitého písmen. V případě nevhodně vygenerované obtížnosti slova, ji uživatel bude moci opravit dle vlastního uvážení. Délka slova je prahově rozdělena do kategorií obtížnosti, následně bude vyhodnocena podobnost s významem pomocí Levenshteinovy vzdálenosti popsané v podkapitole 3.5.2. Poté následuje vyhodnocení přehlasování a dvojitého hlásek, které ale ovlivňují obtížnost už jen minimálně.

#### 3.5.1 Výběr algoritmu

Levenshteinova vzdálenost je velmi podobná známé Hammingově vzdálenosti s tím rozdílem, že Hammingova vzdálenost uvádí počet pozic se stejným symbolem v obou řetězcích. Kdežto Levenshteinova vzdálenost uvádí počet jednoznakových editací. Pro určení míry podobnosti mezi slovem v cizím jazyce a jeho překladem je vhodnější Levenshteinova vzdálenost, jelikož akceptuje řetězce s různou délkou a v případě, že písmeno chybí, je to vyhodnoceno pouze jako odstraněné písmeno a algoritmus pokračuje dál. V případě Hammingovy vzdálenosti vynechané písmeno naruší porovnávání zbytku řetězce.

#### 3.5.2 Levenshteinova vzdálenost

Levenshteinova vzdálenost v informatice je míra rozdílu mezi dvěma řetězci. Pro výpočet vzdálenosti se využívá matice s rozměry velikostí délek řetězců. V podstatě je vzdálenost minimální počet jednoznakových úprav ve slově, aby vzniklo slovo referenční. Za jednoznakovou úpravu se považuje smazání, záměna nebo vložení písmene na dané místo [?].

Vzorec 1 je matematický zápis Levenshteinovy vzdálenosti, kde  $a$  a  $b$  jsou řetězce  $a(i)$  a  $b(j)$  jsou indexované znaky v řetězcích. Výsledná vzdálenost je rovna hodnotě v matici  $dist_{a,b}$  na pozici  $dist_{a,b}(|a|, |b|)$ , kde  $|a|$  a  $|b|$  jsou délky řetězců [13].



$$dist_{a,b}(i,j) = \begin{cases} max(i,j) & min(i,j) = 0 \\ min \begin{cases} dist_{a,b}(i-1,j) + 1 \\ dist_{a,b}(i,j-1) + 1 \\ dist_{a,b}(i-1,j-1) + \begin{cases} 0 & a(i) = b(j) \\ 1 & a(i) \neq b(j) \end{cases} \end{cases} & min(i,j) \neq 0 \end{cases} \quad (1)$$

V případě, že se pohybujeme v matici v prvním sloupci a prvním řádku, přiřazujeme hodnoty 1 až délka řetězce  $a$  do sloupce a 1 až délka řetězce  $b$  do řádku. Tedy první sloupec a první řádek v matici slouží jako referenční inicializace. V generování matice se následně pokračuje a porovnávají se jednotlivé znaky. První položka v minimu je případ, kdy písmeno bylo odstraněno z  $a$ . Druhá položka je případ, kdy došlo k vložení znaku do  $a$  na základě znaku v  $b$  řetězci. A poslední položka je případ porovnání znaků. V okamžiku, kdy se znaky rovnají, hodnota vzdálenosti na diagonále zůstává. V opačném případě se přičítá jednička.

Výpočet Levenshteinovy vzdálenosti dvou slov znázorňuje matice 2, kde dochází k porovnání českého slova *cyklus* a anglického výrazu *cycle*. Tučně je znázorněna cesta výpočtu. Prováděné jednoznakové operace jsou žádná, žádná, žádná, záměna, záměna a vložení.

		<i>c</i>	<i>y</i>	<i>c</i>	<i>l</i>	<i>e</i>	
	<b>0</b>	1	2	3	4	5	
<i>c</i>	1	<b>0</b>	1	2	3	4	
<i>y</i>	2	1	<b>0</b>	1	2	3	
<i>k</i>	3	2	1	<b>1</b>	2	3	
<i>l</i>	4	3	2	2	<b>1</b>	2	
<i>u</i>	5	4	3	3	2	<b>2</b>	
<i>s</i>	2	5	4	4	3	<b>3</b>	

(2)

### 3.5.3 Určení globální obtížnosti

Základním parametrem globální obtížnosti slova je jeho délka, která má váhu 4 v celkovém průměru jednotlivých hodnot obtížnosti. Na základě prahových hodnot je délka rozdělena do čtyř kategorií -  $\langle 1, 6 \rangle$  je snadné,  $\langle 6, 11 \rangle$  je střední a  $\langle 11, \infty \rangle$  je slovo těžké. Dalším parametrem k určení obtížnosti je Levenshteinova vzdálenost mezi referenčním slovem a jeho překladem, jejíž váha je 2. Znovu zde figurují nastavitelné prahové hodnoty - v případě, že se procentuální podobnost (tj.  $pp = (1 - a/l) * 100$ ; kde  $l$  je délka slova a  $a$  je počet jednoznakových editací) nachází v intervalu  $\langle 70, 100 \rangle$ , je obtížnost nejlehčí; v intervalu  $\langle 30, 70 \rangle$  je střední a v intervalu  $\langle 0, 30 \rangle$  je slovo těžké. Přehlasování znaku a dvojitý výskyt stejného znaku mají oba stejnou váhu 1. Bez výskytu přehlasování a

dvojitého znaku je obtížnost kvalifikována jako snadná, jeden výskyt je střední obtížnost a více výskytů je obtížnost vyhodnocena jako těžká.

### 3.6 Generování slov pro procvičování

Způsob generování slov je důležitý pro správné ukotvení slova v dlouhodobé a ne krátkodobé paměti. Při testování bude docházet i k několikanásobnému opakování slova. Základem generování je častěji a vícekrát předkládat slova, která jsou pro žáka problematická a slova, která jsou zvládnuta žákem s přehledem, jsou méně častěji testována. Pro generování byla zvolena metoda rozloženého opakování v podobě přizpůsobeného Leitnerova algoritmu, který je popsán v kapitole 2.6.2.

Algoritmus pro procvičování slov se dělí na dvě části. V první části se zjišťuje, jaká slova jsou pro zkoušeného uživatele jednoduchá a naopak s jakými slovy má problém - tato fáze se bude dále nazývat inicializační. Po této fázi následuje fáze s použitím Leitnerova algoritmu - tato fáze bude nazývána testovací.

#### 3.6.1 Inicializační fáze

Na začátku testování jsou k dispozici slova s přiřazenou obtížností. Tato obtížnost je buď globální v případě, že se uživatel nesetkal ještě s daným slovem v aplikaci, nebo uživatelská v případě, že uživatel dané slovo již procvičoval. Dle této obtížnosti se slovíčka seřadí tak, aby se střídala nejnižší a nejvyšší obtížnost. Na prvním místě tedy je slovo s nejnižší obtížností, následuje slovo s nejvyšší, dále slovo s druhou nejnižší atd. Důvodem tohoto seřazení je zvýšení motivace dětí. V případě, že by žák dostal na začátku nejtěžší slova, mohlo by si vytvořit odpor k procvičování v domněnku, že slova jsou pro něj příliš složitá.

Uživatel tedy postupně odpovídá na slovíčka. Jeho odpověď je vyhodnocena buď správně, špatně nebo neúplně. Na základě vyhodnocené odpovědi a obtížnosti je ke slovu přiřazen počet nutných správných opakování pro dokončení slova.

#### 3.6.2 Testovací fáze

Po inicializační fázi následuje fáze testovací. V této fázi je již známo, jaká slova dělají uživateli problém. Oproti standardnímu Leitnerovu systému popsanému v kapitole 2.6.2, kde na začátku jsou všechny otázky zařazeny do první kategorie, se zde otázky rozdělují do třech kategorií na základě inicializační fáze. Následuje první fáze, kde dochází k postupnému procházení otázek v první Leitnerově kategorii od nejtěžších po nejlehčí. V případě správně nebo neúplně zodpovězené otázky, je slovo přesunuto do vyšší kategorie. Následuje druhá fáze, kde se prochází otázky z první a druhé Leitnerovy kategorie. I zde platí to, že na základě odpovědi dochází k přesouvání otázek mezi kategoriemi, jak už tomu je

ve standardním provedení Leitnerova systému. Ve třetí fázi jsou nabízeny otázky z první, druhé a třetí Leitnerovy kategorie. Po průchodu třetí fáze následuje znovu fáze první. Takto algoritmus cykluje do doby, dokud uživatel nepřeruší test nebo nezůstanou žádná slova, která by uživatel nedokončil.

Za dokončené slovo se považuje takové, kde uživatel odpověděl správně nebo neúplně tolikrát, kolik má slovo přiřazených nutných správných opakování a zároveň poslední odpověď nesmí být neúplná, ale pouze správná. Je to z důvodu, aby bylo ověřeno, že uživatel má správně zapamatovanou informaci.

### 3.6.3 Adaptivní a globální obtížnost

V případě přerušení testu dochází k ukládání aktuální Leitnerovy kategorie k danému uživateli. Díky tomu lze znovu navázat v případě, že uživatel se bude chtít k testování dané skupiny slov vrátit. Leitnerova kategorie vypovídá, jak problematické je slovo pro uživatele. V případě, že se slovo nachází v první kategorii, je problematické, a v případě, že se nachází ve třetí, uživatel ho zvládá bez obtíží. Díky této informaci dochází k adaptivnímu generování obtížnosti. V případě, že není u daného slova a uživatele uložena Leitnerova kategorie, je v inicializační části brána globální obtížnost.

Například se může stát, že globální obtížnost slova je v kategorii těžkých, ale uživatel se s tímto slovem už několikrát setkal a dobře se mu pamatuje. Je proto zbytečné slovo mu opakovat vícekrát, jelikož na základě obtížnosti a správnosti odpovědi se generuje právě i počet opakování.

## 3.7 Procvičování slovíček

Základním cílem procvičování a testování slovíček je pomoci se žákům efektivně naučit slovíčka, nikoliv umožnit vyučujícím posuzovat, jak na tom žáci jsou. Následující podkapitoly popisují, jakým způsobem v aplikaci probíhá testování, napovídání, vyhodnocování odpovědí a kontrola podvádění.

### 3.7.1 Metody testování

Pro procvičování slovíček je zvolen pouze aktivní přístup a to v metodách aktivního rozpoznání a aktivního vzpomnutí. Tyto metody se automaticky střídají v závislosti na odpovídání uživatele. V případě inicializační fáze procvičování, popsané v kapitole 3.6.1, se metoda testování nemění, aby uživatel měl stejné podmínky pro vyhodnocení úvodní obtížnosti slova. V následující fázi, kde probíhá samotné procvičování, se metody střídají. Když uživatel odpoví špatně na vzpomnutí a u slova má více než jednu nutnou správnou

odpověď pro dokončení slova, následuje metoda rozpoznání. Je to z důvodu zvýšení motivace, aby procvičování uživatele neodradilo, když si nevzpomene na cizojazyčný výraz.

Jelikož vzpomenutí je paměťově náročnější úkol, je nutné, aby metoda poslední odpovědi byla vzpomenutí. Jinak není možné ověřit, zda si uživatel dobře pamatuje cizojazyčný výraz slova. Tzn. v případě, že počet nutných správných odpovědí byl vyhodnocen na číslo 2, první metoda testování bude rozpoznání - uživatel napíše význam slova v mateřském jazyce. Pokud odpoví správně, následuje metoda vzpomenutí. Pokud ale odpoví špatně, následuje znovu metoda rozpoznání.

### 3.7.2 Nápořědy

V procvičování slovíček má uživatel k dispozici i nápořědy. V případě, že vlastník učebnice vyplní ke slovům i doplňující informace, jako jsou definice a použití ve větě. Tyto nápořědy jsou použity pouze v případě, že se testuje metodou vzpomenutí. Další nápořědou je zobrazení prvního písmene odpovědi. Definice je nápořěda, kterou uživatelé vidí jako doplňující atribut otázky. V případě, že uživatel klikne na tlačítko "Použít nápořědu", zobrazí se mu první písmeno odpovědi. Pokud klikne na tlačítko znovu, zobrazí se mu použití slova ve větě.

### 3.7.3 Vyhodnocování odpovědi

K vyhodnocování odpovědi se využívá znovu Levenshteinovy vzdálenosti popsané v kapitole 1. Odpověď je vyhodnocena do tří kategorií - správně, špatně a neúplně. Před porovnáním slova, jsou z odpovědi odstraněny i násobné mezery, tabulátory a další speciální znaky. Správně je vyhodnocena pouze odpověď, která se plně shoduje s referenčním slovem. Jako neúplná je vyhodnocena v případě, že na 5 písmen je pouze jedna jednoznačková editace - tedy chybovost 20%. V ostatních případech je odpověď vyhodnocena jako špatná. Pokud uživatel využije alespoň jednu z nápořěd, automaticky se vyhodnocuje odpověď jako neúplná i v případě zcela správné odpovědi.

### 3.7.4 Kontrola podvádění

Aplikace nemá za úkol testovat žáky, zda zodpovědně a bez pomoci procvičují slova. Filozofií aplikace má být motivovat žáky a poskytnout jim příjemný nástroj na efektivnější učení slovíček z cizího jazyka. Proto v aplikaci nejsou zakomponovány žádné mechanismy, které mají za úkol kontrolovat uživatele, zda si například nevyhledávají slovíčka ve slovníku nebo nepracují na procvičení společně s další osobou.

Jsou způsoby, jak by se ale tato funkčnost dala do aplikace implementovat a to například v podobě měření času odpovědi. Vyhodnocení by bylo na základě délky odpovědi a časem

odpovědi. Další, více striktnější, omezení podvádění by mohla být kontrola, zda uživatel opustil aktuální okno aplikace.

### 3.8 Připomínání slov

Připomínání slov je důležitou součástí aplikace. Funkce slouží k tomu, aby studenti nezapomněli na již naučená a zvládnutá slova z procvičování. Jelikož se jedná o webovou aplikaci, není úplně možné připomínat slova v přesných intervalech přímo. Způsob implementace připomínání slov je závislý na tom, aby se uživatel sám přihlásil a spustil funkci připomínání slov. Je ale možné například na základě softwarového démonu Cron v určitých intervalech ze serveru odesílat emailová upozornění se slovy, které by si měl uživatel připomenout.

#### 3.8.1 Supermemo algoritmus

Supermemo algoritmus je rovněž další implementace metody rozloženého opakování popsané v kapitole 2.6. První implementace Supermemo algoritmu v papírové podobě byla vytvořena již v roce 1985. V aplikaci je využívání aktuální verze 11, poprvé implementována v roce 2005 [6].

Algoritmus má dvě možnosti implementace - optimální interval a pokročilé opakování [6]. Verze s optimálním intervalem je pro účely aplikace dostačující, jelikož pro implementaci s pokročilým opakováním by bylo nutné ukládat ke každému uživateli nepřehledné množství dat - každá odpověď ke každému ze slovíček, na základě nichž se algoritmus adaptuje na schopnosti uživatele. Oba algoritmy je možné si vyzkoušet v podobě desktopové aplikace popsané v kapitole 2.2.5. V aplikaci lze pozorovat změny na uživateli v podobě křivky zapomínání a dalších statistických informací.

Definice spočtení optimálního intervalu [6]:

$$I(1) = OF[1, L + 1] \tag{3}$$

$$I(n) = I(n - 1) * OF[n, AF] \tag{4}$$

kde:

- $n$  počet již provedených připomenutí
- $I(n)$  je  $n$ -tý interval v řadě intervalů pro opakování
- $AF$  je absolutní obtížnost
- $L$  počet, kolikrát si uživatel nemohl vzpomenout na slovo
- $OF$  matice optimálních koeficientů pro zvyšování intervalu

- $OF[1, L + 1]$  koeficient z matice na prvním řádku a  $L+1$  sloupci
- $OF[n, AF]$  koeficient z matice, který koresponduje  $n$ -tému opakování a obtížnosti slova

Matice koeficientů k výpočtu optimálních intervalů je získána z freeware aplikace Super-Memo 15. Tato matice je statisticky nejvhodnější pro žáky základních tříd. Ukázku matice znázorňuje následující tabulka 2.

	1	2	...	8	9	10
1	2.48	2.22		1.12	1.00	0.89
2	1.20	1.80		5.40	6.00	6.60
3	1.20	1.50		3.30	3.60	3.90
4	1.20	1.40		2.60	2.80	3.00
...						
17	1.20	1.24		1.46	1.50	1.54
18	1.20	1.24		1.45	1.48	1.52
19	1.20	1.23		1.43	1.47	1.50
20	1.20	1.23		1.42	1.45	1.48

Tabulka 2: Matice koeficientů pro výpočet optimálních intervalů

### 3.8.2 Příklad řady optimálních intervalů

V případě, že si uživatel pokaždé správně vzpomene na naučenou informaci spočtené intervaly pro středně těžké slovo budou přibližně - 2 dny, 6 dní, 14 dní, 27 dní, 49 dní, 82 dní, 131 dní, 7 měsíců, 10 měsíců, 15 měsíců, 21 měsíců, 2,5 roku, 3,5 roku atd. V případě, že dojde k zapomenutí slova, tj. uživatel si při připomenutí nebude moci slovo vybavit, interval se ze začátku zkrátí a začíná se v matici koeficientů na dalším sloupci - což znázorňuje parametr  $L$  v rovnici 3.

## 3.9 Motivace

Motivace je důležitou součástí, jelikož aplikace apeluje na vlastní iniciativu. Aby se žáci rádi vraceli k aplikaci, je nutné zaujmout je designem nebo nějakou vnitřní psychologickou motivací. Další možností motivace je zařadit do aplikace nějakou formu hry. Například za milník naučených slov žáky odměnit například jednoduchou hrou. To by ale vyžadovalo složitější implementaci aplikace.

### 3.9.1 Třídní počet zvládnutých slov

Jednou z vnitřních psychologických motivací je ukázání uživateli, že nějakým způsobem přispívá do kolektivu. Pokud je uživatel přihlášen do třídy nebo skupiny uživatelů, vidí v horním panelu počet zvládnutých slov v rámci třídy. Tento způsob má za následek to, že

může dojít k soutěživosti mezi třídami, přičemž přispívání zvládnutými slovy na úrovni uživatelů je anonymní. V případě, že uživatel dokončí slovo v rámci dané třídy, aplikace ho upozorní v podobě zvukové signalizace a zvýrazní sekci, kde se nachází tyto statistické údaje.

### 3.9.2 Vlastní počet zvládnutých slov

Další vnitřní psychologickou motivací je zobrazení uživateli, kolik slov doposud v aplikaci zvládl procvičit. Informace je uživateli skryta ve výchozím stavu. Pokud uživatel chce vidět vlastní pokrok, musí kliknout na doplňující statistické informace. Důvodem skrytého výchozího stavu je to, aby uživatel neviděl, kolik úsilí je nutné vynaložit, aby slovo bylo korektně dokončeno. Mohlo by se totiž zdát, že se slova do dokončených načítají příliš pomalu, jelikož procvičení slova vyžaduje několikanásobné opakování.

Dalším problémem je unikátnost slov, protože učebnice jsou editovatelné prostřednictvím různých vlastníků a do každé z nich lze zapsat identické slovo. Nelze proto počet zvládnutých slov vyhodnocovat pouze na základě úspěšně dokončených slov, ale na základě unikátních úspěšně dokončených slov. Unikátnost se provádí na referenčním slově v cizím jazyce. Problém ale vzniká, pokud nějaký uživatel při tvorbě slovíček zadává slovo včetně členu a jiný ne. Tato slova se posuzují jako dvě různá.

### 3.9.3 Design aplikace

Design aplikace má také velký vliv na motivaci žáků. Hlavně v případě, kdy design disponuje dostatkem barev, různých designových prvků v podobě zvířat nebo jiných entit, ze kterých jsou děti odvázané. Aplikace je cílená pro děti základních škol zejména prvního stupně, tedy ve věku 6 až 10 let, u nichž design hraje velkou roli.

Pro kvalifikovaný přístup k designu by se měla aplikace zadat profesionálnímu grafikovi, který by design pro děti přizpůsobil.

## 3.10 Architektura aplikace

Aplikace je implementovaná v podobě webové služby a skládá se ze dvou částí - klientská a serverová. Základní myšlenkou návrhu bylo oddělit klientskou a serverovou část, aby bylo možné do budoucna aplikaci rozšířit implementací například pro chytrý telefon.

Serverová část je zodpovědná za servírování klientské aplikace, rovněž aplikace poskytuje aplikační rozhraní, kde na základě dotazů lze získat data ve formátu JavaScript Object Notation (JSON). V serverové části je implementovaný Supermemo algoritmus pro připomínání slov. Dále je server zodpovědný za získání zvukové nahrávky z Google Text to

Speech API, ukládání její kopie a její poskytnutí pro klienta. V neposlední řadě se server zabývá problematikou autorizace a autentizace uživatelů v podobě JavaScript Web Token (JWT).

Klientská aplikace je zodpovědná za zobrazení načtených dat ze serveru, poskytnutí přehledného grafického rozhraní pro tvorbu, editaci a mazání dat nutných pro procvičování a připomínání slov (slovíčka, učebnice, třídy apod.). V klientské části jsou implementovány algoritmus Leitnerova systému a spočtení Levenshteinovy vzdálenosti. Dále klient je zodpovědný za načítání obrázků z Google Search API.



## 4 Klientská část

Tato kapitola se zbývá klientskou částí aplikace od technologického návrhu aplikace, přes vývojové prostředí a vlastní implementaci, až po testování. Aplikace je napsána v jazyce JavaScript za podpory kódovacího jazyka HTML a kaskádovými styly CSS. Základní specifikace jazyků jsou doplněny o knihovny React s využitím MobX rozšíření a o knihovnu pouze s čistým CSS z Bootstrap frameworku.

### 4.1 Návrh aplikace

Využití webových technologií bylo jasnou volbou v návrhu aplikace ihned po specifikování požadavků, protože hlavním požadavkem bylo vytvořit portál pro žáky a učitele, kteří si budou moci sdílet mezi sebou učebnice se slovíčky. Budou se moci přihlašovat do svých tříd a společně pracovat na vylepšení slovní zásoby.

Jelikož se mělo jednat o aplikaci, která do jisté míry měla nahrazovat desktopové aplikace, bylo nutné zvolit takovou technologii, která se nejeví úplně staticky. Kde se nemusí čekat na obnovení celé stránky, ale kde dochází k okamžitým reakcím na uživatelské podněty. Na základě tohoto požadavku je využita technologie Single Page Application (SPA).

#### 4.1.1 Single Page aplikace

SPA je jedna webová stránka, která představuje celou aplikaci. Při prvním načtení je přenesen veškerý kód pro aplikační běh, tj. HTML šablony, JavaScript i CSS. Na základě akcí prováděných uživatelem jsou poté dynamicky načítány ostatní zdroje jako data, obrázky apod. Celá stránka se nikdy neobnovuje, jako tomu je v klasické webové aplikaci. Veškerá aplikační logika je přenesena ze serveru na klienta za účelem server odlehčit. V případě SPA server většinou po načtení aplikace poskytuje pouze data, a to nejčastěji ve formátech JSON nebo XML.

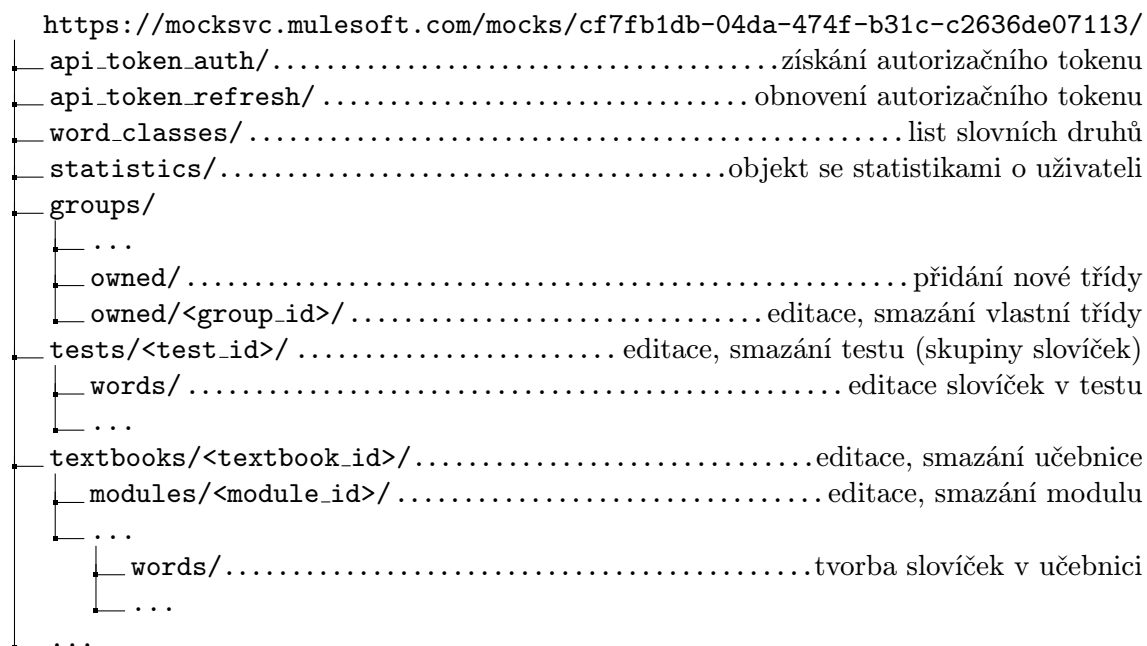
Hlavní výhodou SPA je kromě úvodního načítání dat rychlá odezva. Jelikož se ze serveru nestahuje celý HTML kód, nýbrž jen čistá data. Dochází tedy hlavně k datové i výkonové úspoře, protože prohlížeč nemusí renderovat celou stránku, ale pouze tu část, kde dochází ke změně. Výhodou je také možnost implementace tzv. lazy-loadingu, kdy dojde k rychlému vykreslení obsahu a náročnější operace mohou být načteny nezávisle později [16]. SPA má i řadu nevýhod. Například v prohlížeči pro chod aplikace je nutné mít zapnutý JavaScript, další nevýhodou je obtížné zpracování obsahu pro roboty vyhledávačů. Ale zařazení Search Engine Optimization (SEO) je nepodstatné pro aktuální potřeby aplikace.

### 4.1.2 Komunikace se serverem

Klient komunikuje se serverem obvykle dvěma způsoby - web sokety nebo AJAX (Asynchronous JavaScript and XML) dotazy. Další technologií pro komunikaci, která by mohla být využita je Server-sent events (SSEs), která umožňuje serveru zasílat data klientovi přes HTTP protokol. Pro účely aplikace bude omezenou pouze na AJAX komunikaci, jelikož pro web sokety a tudíž obousměrnou komunikaci v aplikaci využití není.

### 4.1.3 Model aplikace

Před začátkem vývoje klientské aplikace bylo vytvořeno prozatímní aplikační rozhraní pomocí platformy Anypoint od firmy Mulesoft. Je to volně dostupná služba, kde lze velmi jednoduše, staticky a poměrně variabilně popsat serverové API v podobě souborů se syntaxí RAML (RESTful API Modeling Language). Na základě tohoto API byla vytvářena klientská část aplikace. Anypoint umožňovalo API zveřejnit a umožnit testování klienta.



Obrázek 3: Ukázka struktury API

### 4.1.4 Editovatelné seznamy

Pro jednotnost aplikace všechny data lze editovat přímo prostřednictvím seznamů. Vazba  $M : 1$  je v řádcích editovatelná prostřednictvím výběrových listů, vazby  $M : N$  a  $1 : M$  jsou editovatelné skrz vyskakovací okna s možnými návrhy. K získání potřebných dat dochází před načtením editovatelného seznamu. Obsah seznamu je vkládán přímo do editovatelného pole, v případě, že uživatel změní jeho obsah a odejde z buňky, obsah se

automaticky uloží. Ukládaný obsah je omezen pouze na změněnou položku a identifikátor pro optimalizaci přenášeného obsahu.

#### 4.1.5 Design

Pro design aplikace byla využita čistá CSS knihovna z frameworku Bootstrap. Původní verze aplikace byla vyvíjena pod tématickou šablonou Yeti Bootswatch. Na základě konzultace bylo od této šablony opuštěno, jelikož působila příliš hranatě. Téma bylo vyměněno za Cerulean, které disponuje oblejšími prvky a výraznějšími barvami, aby žáky více zaujalo. Design aplikace je také responzivní.

#### 4.1.6 Adaptovaná Levenshteinova vzdálenost

Pro vyhodnocování odpovědi a určení podobnosti mezi referenčním slovem a jeho překladem se využívá Levenshteinovy vzdálenosti popsané v kapitole 1. Tato vzdálenost pro určení podobnosti slov musela být v jisté míře upravena, aby lépe vyhodnocovala podobnost slov mezi jazyky. Standardní Levenshtein, který měl parametry dva řetězce byl rozšířen o další nepovinné atributy - možnost odebrat diakritiku a možnost předat funkci slovník, kde klíč ve slovníku je písmeno v mateřském jazyce a hodnota ve slovníku je písmeno v cizím jazyce.

Pro příklad v anglickém jazyce písmeno *c* odpovídá písmenu *k* v tom českém. Po předání slovníku do funkce pro výpočet Levenshteinovy vzdálenosti dojde k porovnání znaků na daných pozicích a v případě, že nesouhlasí, funkce se pokusí porovnat s odpovídajícím písmenem ve slovníku. Implementace algoritmu pro výpočet vzdálenosti lze nalézt v příloze.

```
1 let dictionary = [{'k': 'c'}, {'ch': 'š'}, ...]
2 "šance".levenshtein("chance", dictionary) // === 0
3 "šance".levenshtein("chance") // === 2
4 "korektní".levenshtein("correct", dictionary) // === 3
5 "korektní".levenshtein("correct") // === 5
```

Ukázka kódu 1: Příklady použití Levenshteinovy vzdálenosti

#### 4.1.7 Google Search API

Pro získání obrázků se využívá Google Search API. Obrázky jsou získány v podobě odkazů. Uživatel si může vybrat z deseti obrázků. Pokud žádný z nich nevyhovuje, vytvoří se nový dotaz s další stránkou. Pro možnost tvorby dotazů bylo nutné v Google API Console vytvořit vlastní API klíč, který se přidává jako parametr URL. K API klíči je také

nutné vytvořit vlastní Google vyhledávač se svým identifikátorem. Tento atribut je rovněž přidáván jako parametr dotazu. API vlastního dotazu vypadá takto:

```
https://www.googleapis.com/customsearch/v1?q=<slovo>&searchType=image&cx=
<id_vyhledavace>&key=<api_klic>
```

Odpověď na dotaz je JSON soubor s odkazy na jednotlivé obrázky. K dispozici jsou jak náhledové, malé obrázky (řádkově několik desítek kB), tak obrázky v plné velikosti. Jelikož nelze zaručit permanentní přístup k souboru, po vybrání obrázku se na serveru vytvoří komprimovaná kopie obrázku s jednotnou šířkou.

#### 4.1.8 Zabezpečení

Pro všechny funkce v aplikaci je nutné se přihlásit popřípadě registrovat. Uživatel vytvoří na základě svého jména a hesla účet, pod kterým bude v aplikaci vystupovat. Aplikace komunikuje pouze přes zabezpečenou komunikaci HTTPS. Aby nebylo možné heslo a jméno odposlechnout technikou MitM (Man in the Middle). Jakmile je uživatel na základě hesla a jména ověřen, je mu zaslán JWT token, se kterým se identifikuje v hlavičce s každým dalším dotazem. Generování a ověřování JWT tokenu je popsáno v kapitole 5.3.1.

#### 4.1.9 Import slovíček

Import slovíček se skládá ze dvou částí, první část má za úkol zpracovat vstupní soubor (například CSV), výstup z této části je JSON soubor s pevně danou strukturou. Ukázka 2 znázorňuje, jak struktura výstupního JSON souboru vypadá. Druhá část má za úkol JSON soubor rozložit na části akceptovatelné API rozhraním, se kterým klient komunikuje, a uložit je do databáze. Důvodem rozdělení importu na tyto dvě části je možnost do budoucna rozšířit aplikaci o další typy a struktury vstupního souboru. Pro parsování CSV souboru je využita aplikace třetí strany Papa Parse verze 4, která za definovaných pravidel umožňuje z CSV souboru vytvořit soubor typu JSON.

Z návrhu aplikace je dané, že slovíčka musí být součástí modulu. Pokud se ve vstupním souboru nachází prázdný řádek nebo v řádku je jen jedna hodnota, parser to vyhodnotí jako nový modul. Pokud chce vyučující hromadně přidat slova pomocí importu, názvy modulů musí souhlasit s těmi aktuálními v aplikaci.

```

1  [{
2      "title": "Module 1",
3      "words": [{
4          "value": "father",
5          "meaning": "otec",
6          "definition": "", // nepovinný atribut
7          "usage": "", // nepovinný atribut
8      }, { ... }]
9  }, {
10     "title": "Module 2",
11     "words": [...]
12 }]

```

Ukázka kódu 2: Struktura výstupního souboru z první části importu

## 4.2 Vývojové prostředí

Pro vývoj SPA je důležitá správná konfigurace vývojového prostředí, jelikož dokáže hodně ovlivnit produktivnost programátora. Základními požadavky na vývojové prostředí je předzpracování kódu, automatické obnovení stránky po uložení editovaného souboru, vytvoření jednotného JS (JavaScript) bundle souboru, který obsahuje všechny potřebné zdrojové kódy rozdělené do logických celků a poskytnutí HTTP serveru k servírování aplikace. Dále pro produkční verzi aplikace je nutná například komprimace zdrojových kódů.

### 4.2.1 Webpack

Webpack je nástroj napsaný pro NodeJS, který umožňuje vytvořit požadované vývojové prostředí. Pro vývoj klientské aplikace jsou vytvořeny dvě webpack konfigurace - vývojová a produkční. Pro kompilaci JS souborů je využíván Babel kompilátor, který je popsán v následující kapitole. CSS soubory jsou ve vývojové konfiguraci vkládány přímo jako inline styly do souboru *index.html*, aby bylo možné v reálném čase sledovat změny designu a nečekat na obnovení stránky. V produkční verzi je vytvořen zvláštní CSS soubor, protože při stahování aplikace do prohlížeče dochází k paralelnímu načítání CSS a JS souborů, což vede k urychlení úvodního načtení aplikace. Pro vytvoření produkční verze je využívána v kompilaci řada webpackových modulů, jako například *favicons-webpack-plugin* k tvorbě ikon pro řadu různých prohlížečů nebo *html-webpack-plugin* pro minifikaci (kompresi) a alespoň částečnému znemožnění přečtení vytvořeného JS souboru [17].

### 4.2.2 Babel

Babel je kompilátor, který transformuje rozšířený JavaScript do standardu, který většina prohlížečů podporuje. Důvod využití tohoto kompilátoru je vysoká podpora EC-

MAScript2015 (ES6) (standard ECMA-262) a integrace JSX (XML-Like Syntax Extension) popsaných v následujících kapitolách. Babel je mimo jiné také doporučován komunitou zabývající se knihovnou React a používají ho společnosti jako Facebook nebo Evernote. Výhodou je, že Babel pouze rozšiřuje současnou verzi JavaScriptu a snaží se zachovat jeho dopřednou kompatibilitu [18].

### 4.2.3 Podpora ES6

Díky kompilátoru Babel lze využít novinky ze specifikace ES6, i když ještě není implementována do prohlížečů. Při vývoji bylo nedílnou součástí klíčové slovo *export*, které umožňovalo rozdělení kódů do mnoha modulů a souboru. V aplikaci je také hojně využívána nová syntaxe tříd, která umožňuje přehlednější dědičnost, tvorbu konstruktorů a statických metod. Novinek, které jsou využívány v aplikaci, je celá řada. Za zmínku stojí zkrácená šipková syntaxe  $\Rightarrow$ , která umožňuje sdílet klíčové slovo *this* uvnitř vnořeného logického bloku [19].

### 4.2.4 JSX

JSX je specifikace, která umožňuje kompilovat stromovou strukturu podobnou XML do objektů standardu ECMAScript [20]. Jelikož má stejnou strukturu jako XML, je možné předávat značkám atributy klíč-hodnota. Díky kompilaci této struktury lze tedy uložit do JS proměnných celé šablony HTML kódu. Na základě těchto šablon je poté renderováno grafické rozhraní.

## 4.3 Knihovna React

React je v JS komunitě rozšířená JavaScriptová knihovna pro vytváření webových komponent. Její předností je efektivní aktualizace a renderování pouze těch komponent, ve kterých dochází ke změnám dat. Uživatel při jejím použití neprování změny přímo na DOM (Document Object Model) dokumentu, ale na jeho abstraktní vrstvě. Výhodou je, že uzly v abstraktní vrstvě mohou uchovávat jejich vnitřní stav. V případě jeho změny React zajistí automatické a efektivní promítnutí změn v DOM dokumentu.

### 4.3.1 Komponenty

Prezentační vrstvu aplikace lze rozdělit na jednotlivé logické celky v podobě jednoduchých a složitých komponent. Jednotlivé komponenty lze jednoduše vnořovat do sebe a tím vytvořit složitější interaktivní uživatelské rozhraní. Každá komponenta dědí od třídy *Component* v knihovně *React* a obsahuje funkci *render* vracející obsah, který je následně vykreslený v těle dokumentu. Dále komponenty obsahují funkci *setState*, která mění její

vnitřní stav. Dále důležitými atributy jsou *props* a *state*. *Props* obsahuje konstantní atributy, které jsou předané komponentě a *state* obsahuje stavové objekty komponenty. V případě, že dojde ke zavolání *setState* funkce a tedy změně objektu *state*, dojde k zavolání funkce *render*, a tudíž k překreslení obsahu komponenty [21].

```

1 export default class Hint extends React.Component {
2   showHint = e => {
3     this.setState({showHint: true})
4   }
5   render() {
6     const { definition, value, examStore } = this.props
7     const { showHint } = this.state
8     let firstLetter = value.substring(0, 1)
9     return <div className="hints">
10       <Button onClick={this.showHint}>Zobrazit nápovědu</Button>
11       {definition && <p>Definice: {definition}</p>}
12       {showHint && <p>První písmeno odpovědi: {firstLetter}</p>}
13     </div>
14   }
15 }

```

Ukázka kódu 3: Příklad komponenty pro zobrazování nápovědy při procvičování slovíček

### 4.3.2 MobX

MobX poskytuje mechanismus k ukládání a aktualizaci dat a tudíž i stavů aplikace [22]. Lze si ho představit jako oddělené úložiště dat, které zároveň obsahuje veškeré možné manipulace s daty. V případě, že v úložišti jsou data označená jako *@observable* a uživatel jejich obsah změní, dojde k překreslení všech komponent, ve kterých se daná *@observable* proměnná vyskytuje. To přináší React aplikaci možnost zcela oddělit datovou vrstvu aplikace od zobrazovací vrstvy a přiblížit se tak MVC modelu. V aplikaci je využitý MobX společně s dekorátory a anotacemi z ES.Next [22].

Všechny komponenty, které mění data MobX úložiště, musí být anotovány *@observer*. Pro získání reference na data z úložiště v komponentě, je využívána anotace *@inject*. V aplikaci jsou dále využívány i doplňkové funkce MobX jako například *@computed* atribut nebo *@action* funkce.

V úložištích se rovněž provádí veškeré AJAX dotazy prostřednictvím externí knihovny Axios, která umožňuje mimo jiné generické nastavení hlaviček u jednotlivých HTTP dotazů.

### 4.3.3 Směrování a historie

Pro směrování a historii v SPA jsou využity externí knihovny z NPM (Node Package Manager) - *react-router* a *mobx-react-router*. Používat navigační tlačítka a historii prohlížeče umožňuje modul *syncHistoryWithStore*, který synchronizuje historii v podobě JS objektu s historií prohlížeče. K manipulaci historie prohlížeče se využívá JS funkcí *pushState* a *replaceState*, definovaných ve standardu HTML5.

Směrování v aplikaci umožňuje modul *Router* z knihovny *react-router*. Tento modul lze do sebe nesčetněkrát vnořovat a umožňuje lépe oddělit logické části aplikace. *Router* mimo jiné disponuje nepovinným atributem *onEnter*, ve kterém lze zajistit oprávnění pro různé sekce aplikace. Dalším důležitým modulem je *Provider* z knihovny *react-router*, který poskytuje pro komponenty Reactu MobX úložiště. Následující ukázka zdrojového kódu 4 představuje hlavní vykreslovací funkci aplikace, jejíž první atribut je obsah a druhý je element, kam se má obsah vložit. Jednotlivé *Route* komponenty operují jako selektor stránek na základě URL.

```

1 ReactDOM.render(
2   <Provider {...stores}>
3     <Router history={history}>
4       <Route path="/" component={Layout}>
5         <IndexRoute component={Welcome}/>
6         <Route path="/auth" component={Authentication}></Route>
7         ...
8         <Route path="/textbooks/:textbookId" component={
9           TextbookEditor} onEnter={requireAuth}></Route>
10        </Route>
11      </Router>
12    </Provider>,
13    document.getElementById('app'));
```

Ukázka kódu 4: Hlavní renderovací funkce aplikace

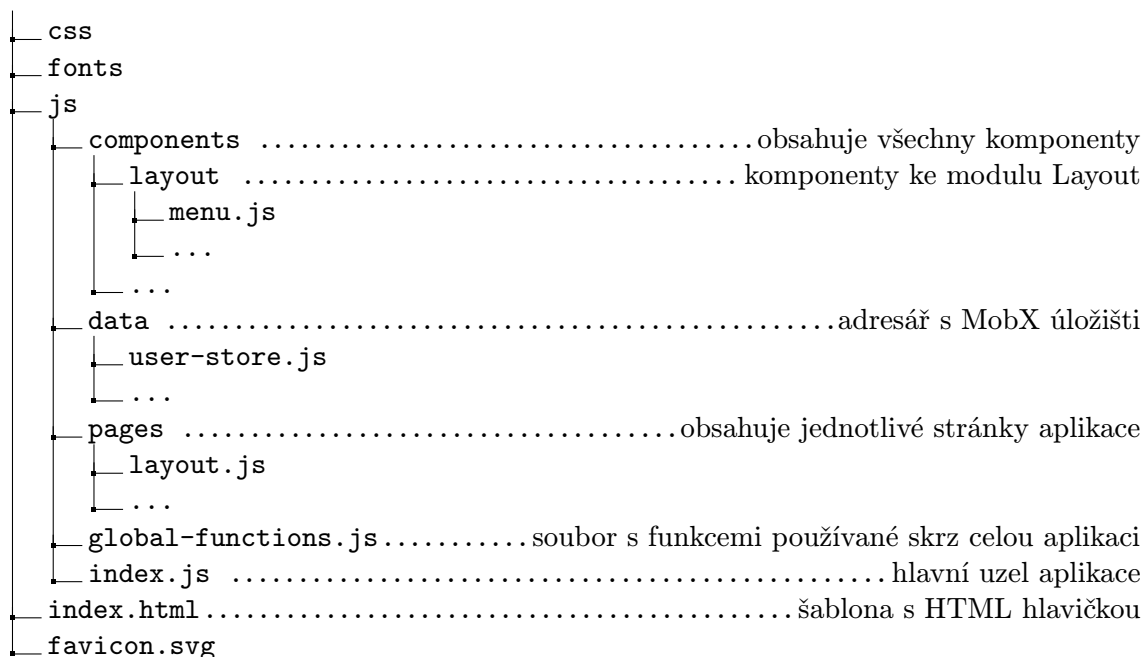
### 4.3.4 Adresářová struktura

Následující obrázek 4 popisuje adresářovou strukturu klientské aplikace.

## 4.4 Testování

Pro testování klientské části byly využity nástroje Enzyme a Jest. Enzyme je testovací nástroj, který umožní testovat React komponenty. Jeho syntaxe je velmi podobná známé knihovně JQuery. Jest je nástroj, který tyto testy spouští a vyhodnocuje. Pro kompilaci ES6 kódu byl využit balíček *babel-jest*.





Obrázek 4: Adresářová struktura klientské aplikace

#### 4.4.1 Unit testování

Díky MobX úložišti je velmi jednoduché otestovat jednotlivé funkce, které manipulují s daty. MobX úložiště je JS objekt, který lze v jakémkoliv stavu aplikace uložit a pro potřeby testů znovu načíst. Díky této vlastnosti jsou jednotkové testy zcela odděleny od aplikace a lze je spouštět nezávisle. Enzyme umožňuje simulovat i JS události zavoláním funkce *simulate* na daný React element. První atribut je typ události a druhý atribut je objekt, který simuluje objekt události.

Ke každému úložišti byl vytvořen soubor s příponou *.test.js*, tyto testy jsou umístěny v adresáři *./data/\_\_tests\_\_*/. Pro testování jsou k dispozici funkce *describe*, *it* a *expect*. Enzyme je doplněn i o další standardní funkce jednotkového testování, jako například *beforeEach*, která umožňuje načíst stejná data do úložiště pro každý test zvlášť.

#### 4.4.2 Integrační testování

Pro testování, zda komponenty společně správně pracují, slouží integrační testování. Článek, který komponenty spojuje, je právě již zmiňované MobX úložiště. Každý z integračních testů obsahuje úložiště, do kterého jsou vložena data představující určitý stav v aplikaci. Toto úložiště je poté předloženo daným komponentám. Test následně provede sadu operací nad komponentami a na závěr zkontroluje, zda výsledná komponenta vrací požadovaný obsah.

Důležitou součástí v kontrole obsahu je funkce *shallow*, která obalí JS objekt představující

uzel v dokumentu funkcemi, které umožní jednoduché kontrolování jednotlivých elementů, jejich obsahu a atributů.

## 5 Serverová část

Serverová část aplikace je v podstatě aplikační rozhraní, které poskytuje data ve formátu JSON se specifikací ECMA-404 pro klientskou aplikaci. API je napsané v programovacím jazyce Python s využitím Django webového frameworku a Django REST frameworku.

### 5.1 Technologie

Důvody, proč bylo zvoleno Django a Django REST framework jako platforma pro aplikační rozhraní, jsou dřívější zkušenosti s variabilitou a rychlostí vývoje. Pro potřeby aplikace se jedná možná o zbytečně komplexní framework, jelikož aplikace jako celek stojí převážně na zpracování klientské části.

#### 5.1.1 Django

Django je open-source webový framework, který využívá vlastní implementaci MVC modelu, kterou označuje jako MTV (Model Template View) [23]. Z databázového modelu se získávají data, které jsou interpretovány v případě aplikace skrz serializéry. A následně jsou data podstrčeny podhledům, které odešlou uživateli data například v podobě JSON souboru. Django je velmi variabilní a lze ho použít jak pro tvorbu standardní webové aplikace, tak s pomocí REST frameworku i pro API.

Z Django projektu je v aplikaci využívána hlavně modelová vrstva, která je implementována přes ORM (Object-Relational Mapping) přístup. U přístupu do databáze se nepíšou SQL dotazy, nýbrž se provádí operace nad objekty za pomoci definovaných funkcí v Pythonu. ORM implementované v Django transformuje objekty na dotazy, snaží se o jejich optimalizaci, a poskytuje nám vyšší abstrakci [23]. Využitím Django ORM v aplikaci se detailněji zabývá kapitola 5.4.

#### 5.1.2 Django REST framework

Webový framework Django je v aplikaci rozšířen o nadstavbu jménem Django REST framework. Jeho přínosy jsou rychlá tvorba vlastního HTTP nebo i RESTful API, připravené bezpečnostní principy s možností využití OAuth2. Dále obsahuje připravené serializační, parsovací a pohledové komponenty s možností generické a automatické validace dat a dalších užitečných funkcí [24].

### 5.1.3 HTTP a RESTful API

Klientská aplikace se dotazuje serveru pomocí požadavků metod GET, POST, PUT a DELETE. Jelikož je klient úzce svázaný se serverem, API nemusí nést principy HATEOAS (Hypermedia as the Engine of Application State) ani RESTful rozhraní. Tedy takové, které mimo jiné poskytuje dynamicky vytvářené hypertextové odkazy pro manipulaci stavů aplikace. Toto rozhraní je vhodnější pro aplikace, kde veškeré logické části aplikace jsou prováděny na serveru a klient slouží pouze jako prohlížeč dat. Ve slovníkové aplikaci je naopak veškerá logika a stavovost přesunuta na klienta a API slouží pouze jako vrstva pro ukládání dat. Rozhraní neposkytuje ani odkazy pro navigaci mezi daty, z důvodu optimalizace velikosti přenášených dat, jelikož jak už bylo zmíněno, klient je úzce svázaný se serverovým rozhraním. Z těchto důvodů by se dalo rozhraní klasifikovat spíše jako HTTP API.

## 5.2 API

Tato sekce poskytuje náhled na vnitřní strukturu a principy serverové části aplikace.

### 5.2.1 Struktura dat

Data odesílané klientovi jsou číté reprezentace atributů v modelové vrstvě. Každý z objektů je identifikován svým číslem datového typu *integer*. Co se týče relačních vztahů mezi modely, ty jsou reprezentovány pomocí zaslaného identifikátoru, popřípadě listu identifikátorů, pokud se jedná o násobný vztah. Pro získání doplňujících informací k zaslanému identifikátoru, je nutné vytvořit nový dotaz typu GET a uvést identifikátor v parametru API.

Z důvodu optimalizace počtu dotazů na server v určité části klientské aplikace bylo nutné nahradit identifikátory, představující vztah mezi objekty, vlastními daty, aby nedocházelo k několikanásobným dotazům s velikostně velmi nízkou odpovědí. Například v případě získání listu učebnic, je klientovi zaslán list objektů, které obsahují atributy, jako název učebnice, datum vytvoření, jazyk, identifikátor vlastníka atd. Aby klient nemusel dotazovat server několikanásobně o další informace, jsou základní data o vlastníkově vnořena přímo do listu učebnice.

V Django REST frameworku tuto funkci zajišťují serializátory. Jsou to třídy, které transformují modelové objekty a dotazové sety do Pythonových objektů tak, aby je bylo možné zapsat do formátu JSON nebo XML [24]. Následující ukázka 5 představuje serializer pro transformaci informací o učebnici. Výstupní objekt bude nést atributy, které jsou uvedeny v tuplu *fields*. Na základě použití funkce *PrimaryKeyRelatedField* bude hodnota *language* atributu pouze identifikátor. Funkce *PrimaryKeyRelatedField* akceptuje atribut *queryset*,

který definuje list možných identifikátorů v podobě dotazu. Naproti tomu atribut *owner* bude ve výstupu objekt, který obsahuje právě dodatečné informace o vlastníkovi - id, uživatelské jméno, jméno a příjmení.

```

1 class OwnerSerializer(serializers.ModelSerializer):
2     class Meta:
3         model = User
4         fields = ('id', 'username', 'first_name', 'last_name')
5
6 class TextbookSerializer(serializers.ModelSerializer):
7     owner = OwnerSerializer()
8     language = serializers.PrimaryKeyRelatedField(
9         queryset=models.Language.objects.all())
10    class Meta:
11        model = models.Textbook
12        fields = ('id', 'title', 'language', 'owner')

```

Ukázka kódu 5: Serializační třída k transformaci dat o učebnici

### 5.2.2 Implementace Supermemo algoritmu

Algoritmus Supermemo popsaný v kapitole 3.8.1. pro výpočet optimálních intervalů je implementovaný na serverové části aplikace. V okamžiku, kdy uživatel dokončí neboli ovládne slovo v procvičování, se ukládá na server informace o tom, že slovo bylo pro daného uživatele dokončeno. Při ukládání se spočítá absolutní obtížnost slova na základě adaptované obtížnosti a zjistí se první interval v podobě data, kdy má slovo být připomenuto. Tento datum společně s hodnotou 0 představující počet, kolikrát si uživatel nemohl vzpomenout na slovo, uloží aplikace do databáze. Matice optimálních faktorů je v aplikaci interpretována jako standardní list listů.

Následně v okamžiku nového přihlášení uživatele do aplikace, server společně s informacemi o statistikách uživatele odešle list slov, u kterých je aktuální datum vyšší, než datum připomenutí. V aplikaci si uživatel může spustit mód připomenutí již naučených slov. V případě, že si uživatel nevzpomene na slovo, při ukládání informací o připomínání, se přičte jednička do hodnoty s počtem, kolikrát si uživatel nemohl vzpomenout na slovo, a pro každé slovo se přepočítá další datum připomenutí.

### 5.2.3 Získání zvukové nahrávky

Pro získání zvukové nahrávky slova je využitý Python modul s názvem gTTS. Stažení nahrávky je velmi jednoduché. Nahrávky se v aplikaci identifikují jazykem a názvem omezeným na ASCII znaky, což v ukázce 6 představuje funkce *slugify*. K uložení nahrávky dochází pouze v případě, že stejné slovo již není v adresáři uložené. Klientovi se následně zasílají odkazy na nahrávku, kde si uživatel může slovo přehrát.

```

1 from gtts import gTTS
2 def save_sound(text, lang):
3     fname = slugify(text)
4     filename = 'static/sounds/%s/%s.mp3' % (lang, fname)
5     if not os.path.exists(filename):
6         tts = gTTS(text=text, lang=lang)
7         tts.save(filename)

```

Ukázka kódu 6: Uložení zvukové nahrávky

## 5.3 Zabezpečení

Serverová a klientská aplikace spolu komunikují skrz zabezpečený protokol HTTPS. Pro autorizaci uživatelů v rozhraní slouží standard JWT popsáný v následující kapitole. Kapitola dále rozvíjí, jak jsou uživatelé autorizováni k jednotlivým operacím a jaké bezpečnostní mechanismy jsou použity proti internetovým útokům.

### 5.3.1 JavaScript Web Tokens

JSON Web Token (JWT) je standard (RFC 7519) definující způsob posílání zabezpečené informace ve formě zakódovaného JSON objektu. Tyto informace jsou zabezpečeny buď s využitím tajného klíče a algoritmu HMAC a nebo pomocí páru privátní/veřejný klíč a algoritmu RSA. Zašifrovaný obsah JWT tokenu obsahuje důležité informace o uživateli a klient tedy nemusí dotazovat server pro tyto informace, což vede k snížení režie serveru [27].

Způsob přihlášení uživatele probíhá následovně. Uživatel zadá do klientské aplikace jméno a heslo, které je dále interpretováno pouze jako MD5 hash. Tyto informace jsou zaslány na server. Server na základě ověření hesla a jména vygeneruje nový JWT token, který zašle zpět klientské části. Ta si token uloží do lokálního úložiště a přidává token ke každému HTTP dotazu v hlavičce se strukturou **Authorization: JWT <token>**.

Na serveru generování JWT obstarává knihovna *django-rest-framework-jwt*. Expirace tokenu je nastaveno na 3 hodiny. Knihovna poskytuje možnost obnovení tokenu s novou expirační dobou. Na klientské části je obnovení řešeno Timeout callback funkcí, která obnoví token ještě než vyprší jeho expirace. Společně s tokenem se klientské části zasílají i statistické informace o uživateli. To zajišťuje vlastní implementace funkce *jwt\_response\_payload\_handler* na následující ukázce 7.

```

1 def jwt_response_payload_handler(token, user=None, request=None):
2     user_info = models.UserInfo.objects.get(user=user)
3     return {
4         'token': token,
5         'statistics': UserInfoSerializer(user_info, context={'request':
6             request}).data
7     }

```

Ukázka kódu 7: Obsah metody *jwt\_response\_payload\_handler*

### 5.3.2 CSRF a XSS

Cross Site Request Forgery (CSRF) je útok, kdy útočník donutí oběť odeslat upravený požadavek na server. V podstatě se jedná o nějaký podvržený odkaz na jiné doméně a jelikož je uživatel stále do napadené aplikace přihlášen, má práva provést například změnu emailové adresy [25]. Proti tomuto typu útoku je aplikace chráněná díky JWT, jelikož s každým požadavkem na server je v hlavičce dotazu zaslán token. JWT nelze z jiné domény zjistit, jelikož prohlížeč pro každé okno vytváří samostatné instance pro provoz JavaScriptu, které mezi sebou data nesdílí.

Cross-Site scripting (XSS) je případ, kdy útočník pomocí vstupního formulářového prvku zapíše JS kód do stránky, který se následně vykoná [26]. XSS je mnohem závažnější útok, jelikož útočník může získat například Cookie dokumentu, které může obsahovat například JWT. Jelikož klientská část funguje jako SPA, nedochází k obnovení stránky a tudíž nedochází k resetování JavaScriptové instance. JWT token je uložen pouze v JS objektu. Proto v případě, že uživatel z aplikace odejde, musí znovu projít autentizačním cyklem.

### 5.3.3 CORS

Pro testovací účely bylo nutné v serverové aplikaci a Django REST frameworku povolit CORS (Cross-Origin Resource Sharing), jelikož server a klient nesdílely stejnou doménu. CORS je mechanismus, který spočívá v povolení klientům dotazovat rozhraní, které je poskytováno na jiné doméně. Pro povolení CORS v serverové aplikaci byl využit další middleware s názvem *corsheaders.middleware.CorsMiddleware*, ve kterém lze nastavit parametrem *CORS\_ORIGIN\_WHITELIST* soupis klientů, kteří mohou dotazovat server. V produkční verzi aplikace tento problém nenastává, jelikož serverová a klientská část sdílí servírovanou doménu.

### 5.3.4 Autorizace

Autorizace ke zdrojům je v aplikaci jednoduchá. Jelikož došlo ke sloučení rolí učitele a žáka, není nutné role v aplikaci rozdělovat. Proto všichni uživatelé mají přístup do všech sekcí

aplikace. Aby ale uživatel měl možnost vůbec dotazovat API, musí být identifikován a přihlášen. K nastavení a zprovoznění tohoto základního oprávnění posloužila třída, která je součástí REST frameworku *rest\_framework.permissions.IsAuthenticated*, kterou je nutné uvést v nastavení pod atributem *DEFAULT\_PERMISSION\_CLASSES*.

Tato autorizace nestačí, jelikož je nutné také omezit editační a mazací metody pouze na vlastníky dat. Následující ukázka 8 představuje třídu oprávnění, která kontroluje, zda vlastník záznamu je přihlášený uživatel v případě, že se snaží provést nějakou úpravu. Pro správnou funkčnost je samozřejmě nutné, aby daný záznam přesněji objekt modelové třídy měl definovaný atribut *owner*.

```

1 class IsOwnerOrReadOnly(permissions.BasePermission):
2     def has_object_permission(self, request, view, obj):
3         if request.method in permissions.SAFE_METHODS:
4             return True
5
6         return obj.owner == request.user

```

Ukázka kódu 8: Ukázka třídy s přizpůsobeným oprávněním

## 5.4 Databáze

K ukládání dat v aplikaci je využita open-sourcová databáze PostgreSQL. Django podporuje celou řadu databázových backendů jako například MySQL, SQLite nebo lze nalézt i obecný konektor ODBC (Open Database Connectivity). Ale největší podporu ve vývoji frameworku dostává právě zmíněná databáze PostgreSQL [23].

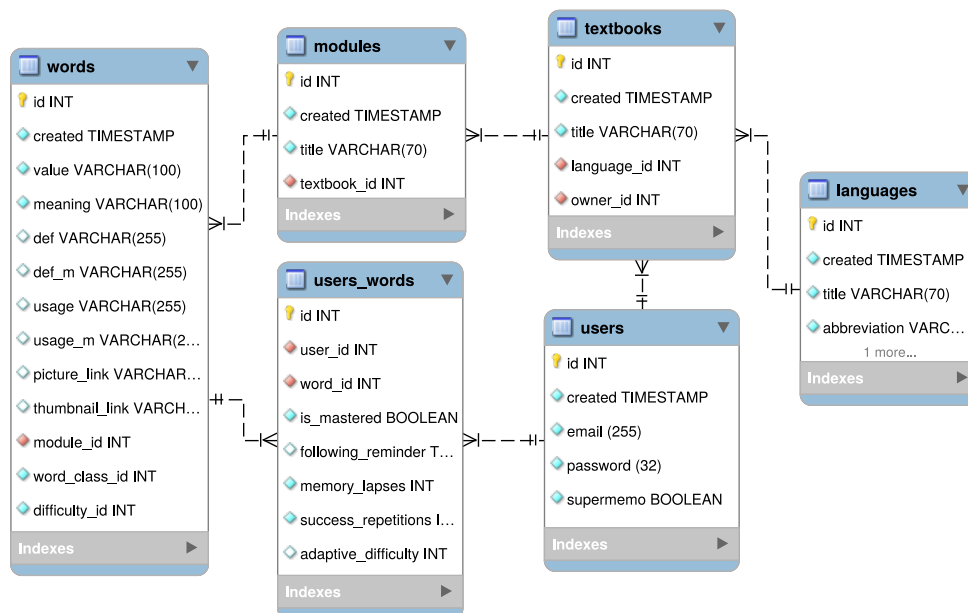
Django jak už bylo zmíněno nepovoluje přímý přístup k databázi. Ve frameworku existuje abstraktní vrstva ORM, která mapuje objekty v Pythonu na databázové dotazy. To přináší spoustu výhod, jedna z nich je vysoká optimalizace dotazů prováděné na databázi, další může být například podpora databázových migrací. Django si sám spravuje strukturu tabulek v databázi. Potom co se implementují modelové třídy, je ve frameworku připraven nástroj, který vytvoří v databázi tabulky s atributy, indexy a se vším, co se v modelových třídách definuje. Django rovněž podporuje propagaci změn na databázi v případě, že dojde k editaci modelových tříd [23].

### 5.4.1 Model

Následující obrázek 5 znázorňuje část modelu databáze, která je zodpovědná za správu učebnice, slovíček a uživatelů. Za zmínku stojí tabulka *words*, která ukládá informace o slovu - překlad, význam, definice, odkazy na náhledový obrázek, i ten v plné velikosti, globální obtížnost atd. Tabulka *users\_words* slouží k ukládání informací o slovu ke každému uživateli. V případě, že se uživatel v procvičování potká s daným slovem, vytvoří se záznam



v této tabulce. S každou další interakcí se změní obtížnost slova interpretovaná atributem *adaptive\_difficulty*. Tato tabulka rovněž ukládá informace o připomenutí slov. Atribut *memory\_lapses* představuje počet, kolikrát uživatel zapomněl slovo, *success\_repetitions* značí, kolikrát uživatel byl schopný si úspěšně vzpomenout na slovo a *following\_reminder* představuje čas, kdy by mělo dojít k dalšímu připomenutí.



Obrázek 5: Ukázka části databázového modelu zodpovědné za správu slovíček

## 5.5 Testování

Testování rozhraní bylo zaměřené pouze na správnou funkčnost API, tzn. kontrolu, zda jednotlivé dotazy vracejí správná data. Jednotkové testování krátkých logických funkcí nemělo příliš význam, jelikož v REST frameworku jde spíše o nastavení součástí a jejich spojení do jednotného celku, než o implementaci nízkourovňových logických bloků.

Pro testování správné funkčnosti API byla využita třída `APITestCase` poskytovaná REST frameworkem. Testy jsou napsány na každou metodu každé API. Kontrola jedné metody API spočívala v ověření stavového kódu HTTP, dále v počtu vrácených dat, a pokud se jednalo o listy objektů, v náhodné kontrole hodnot objektu. K provádění dotazů posloužil objekt třídy `APIClient`, který umožňoval jednoduše editovat hlavičky dotazu pro celou testovací sadu.

Užitečným nástrojem pro testování jsou tzv. *fixtures*. Jsou to soubory ve formátu JSON, které obsahují databázová data. Na začátku každé testovací sady je vytvořeno nové dočasné schéma v databázi, které má identickou strukturu s produkčním schématem v databázi. Dále jsou na začátku testu předány jednotlivé *fixtures*, které pro účely testování naplní databázi požadovanými daty.

### 5.5.1 Uživatelské testy aplikace

Na závěr vývoje aplikace bylo provedeno testování uživateli (akceptační testy), kteří nebyli součástí vývoje. Aplikace pro účely testování byla spuštěna na produkčním serveru, aby imitace reálného prostředí byla autentická. Dále byla do aplikace naimportována sada slovíček z učebnice anglického jazyka s názvem Headway Intermediate. Pro testování byli zvoleni dva dobrovolníci - žákyně základní školy, která aktuálně dochází do šesté třídy Základní školy Lesní v Liberci a dospělý člověk zabývající se podnikáním, který simuloval roli učitele v aplikaci. Během testování bylo objeveno několik drobných chyb, které byly ihned po testování opraveny. Z důvodu časového omezení nebylo otestováno externími uživateli připomínání naučených slov.

Žákyně ihned ze začátku testování narazila na problém, že intuitivně neporozuměla konceptu testů. Aby si mohla začít procvičovat slova, musela totiž vytvořit test. Tato funkce je ale pouze spíše doplňkovou. Standardně vyučující pro žáky vytvoří testovací sady. Po vytvoření sady neměla žákyně žádný problém se spuštěním testu ani s testováním. Uživatelské rozhraní aplikace hodnotila převážně pozitivně s drobnými výtkami na design aplikace, který pro ni byl až příliš strohý a málo barevný.

Testování fiktivním učitelem spočívalo ve vytvoření editací a mazání učebnice, slovíček, modulů, tříd a testovacích sad. Orientace v aplikaci probíhala bez větších problémů. Nesrozumitelnou částí pro něj byla sekce pro řazení testů k jednotlivým třídám. Po drobné nápovědě se podařilo testy přiřadit k třídám.

## 6 Závěr

V rámci této diplomové práce vznikla aplikace pro učení slovíček cizího jazyka. Jedná se o webovou službu, která umožňuje efektivně rozvíjet slovní zásobu u studentů libovolného věku. Pro zlepšení techniky učení jsou v aplikaci využity přizpůsobené algoritmy rozloženého opakování. V části zabývající se procvičováním slov je využitý základ Leitnerova systému. Po procvičení slova následuje fáze připomínání slov, ve které je využit ověřený algoritmus SuperMemo. Generování slovíček v procvičování je přizpůsobováno v závislosti na úrovni uživatele, jenž je získána postupně z jeho předchozího odpovídání.

Aplikace umožňuje každému uživateli vytvořit vlastní učebnice se slovíčky. Slova lze importovat do aplikace v podobě formátu CSV. Ke skupinám slov v jednotlivých modulech lze v aplikaci jednoduše a rychle získat zvukovou a obrázkovou interpretaci. Z důvodu možného nevhodného obsahu obrazové formy, je uživatel nucen vybrat z navržených ten vyhovující, a tudíž proces nemůže být plně automatizován. Aplikace rovněž umožňuje vytvářet skupiny uživatelů, ve kterých lze sdílet celé učebnice.

Aplikace umožňuje vytvářet seznamy slov. Tyto seznamy mohou být sdíleny vlastníkem skupiny mezi ostatní uživatele. Vyučující cizích jazyků tak mohou lépe žáky připravit na nadcházející lekci a zajistit tím její souvislejší průběh.

Do aplikace jsou také zařazeny prvky, které by mohly žáky motivovat k procvičování. Zda splňují funkci motivace, bude ověřeno až praktickým používáním. Během testování tuto funkci nebylo možné ověřit, jelikož je třeba více statistických údajů, což vyžaduje více uživatelů používajících aplikaci. Rovněž bylo složité otestovat, zda aplikace nedovolí uživateli zapomenout slovíčko, jelikož intervaly mezi jednotlivými připomenutími postupně narůstají až k týdnům a měsícům. Pochopitelně žákyně, která testování prováděla, neměla pro tak dlouhé intervaly dostatek času.

Aplikace by si v budoucím vývoji zasloužila řadu rozšíření, zejména po designové stránce. Vzhled by bylo dobré předat profesionálnímu návrhářovi, který by přizpůsobil aplikaci tak, aby zaujala především mladé žáky. Aplikace by také mohla být rozšířena o další možnosti importu slovíček v podobě formátů XLS a PDF. V neposlední řadě rozšířením by mohlo být i zapracování členů přímo do struktury slovíčka, což by umožňovalo přesněji vyhodnocovat odpovědi.

## Použitá literatura

- [1] Terasoft a.s. *Terasoft - Výukové programy* [online] 2002-10-07. [cit. 2016-12-15]. Dostupné z: [http://www.terasoftware.cz/czpages/cd\\_aj15.htm](http://www.terasoftware.cz/czpages/cd_aj15.htm).
- [2] LangSoft s.r.o. *Language Teacher* [online]. [cit. 2016-12-07]. Dostupné z: <http://www.langsoft.cz/teacher.htm>.
- [3] Duolingo Inc. *Duolingo* [online]. [cit. 2016-12-08]. Dostupné z: <https://cs.duolingo.com/info>.
- [4] VESSELINOV Roumen, GREGO John. *Duolingo Effectiveness Study* [online] 2002. [cit. 2016-12-08]. Dostupné z: <https://s3.amazonaws.com/duolingo-papers/other/vesselinov-grego.duolingo12.pdf>.
- [5] LanguageCourse S.L. *Vocabulary Trainer* [online] 2016. [cit. 2016-12-08]. Dostupné z: <http://vt-api.com.es/vocabulary-trainer.php>.
- [6] WOZNIAK Piotr A. *Repetition spacing algorithm used in SuperMemo 2002 through SuperMemo 2006* [online] 2006-04-02. [cit. 2016-12-26]. Dostupné z: <https://www.supermemo.com/english/algsm11.htm>.
- [7] KREJČOVÁ Lenka. *Psychologické aspekty vzdělávání dospívajících*. 1. vyd. Praha: Grada Publishing, 2011 [cit 2016-12-18]. ISBN 978-80-247-3474-3.
- [8] BIEMILLER Andrew, BOOTE Catherine. *An effective method for building meaning vocabulary in primary grades*. Vol 98(1), Journal of Educational Psychology, 2006 [cit 2016-12-18].
- [9] LOZANOV Georgi. *Suggestology and Outlines of Suggestopedy*. 1. vyd. Gordon and Breach, 1978 [cit 2016-12-18]. ISBN 0-203-39282-5.
- [10] I. S. P. Nation *Learning Vocabulary in Another Language*. Cambridge University Press, 2001 [cit 2016-12-18]. ISBN 0-521-800927.
- [11] Kwantlen Polytechnic University. *Spaced Repetition: Remembering What You Learn* [online] 2003. [cit. 2016-12-19]. Dostupné z: [https://www.kpu.ca/sites/default/files/Learning%20Centres/Think\\_SpacedRepetition\\_LA.pdf](https://www.kpu.ca/sites/default/files/Learning%20Centres/Think_SpacedRepetition_LA.pdf).
- [12] SVOBODOVÁ Jana, SVOBODOVÁ Diana, KULDANOVÁ Pavlína, GEJGUŠOVÁ Ivana, ROSOVÁ Milena, NOVÁK Radomil. *Lexikologie* [online] 2003. [cit. 2016-12-19]. Dostupné z: <http://www.osu.cz/fpd/kcd/dokumenty/cestinapositi/lexikologie.htm>.
- [13] Wikimedia Foundation, Inc. *Levenshtein distance* [online] 2016-12-20. [cit. 2016-12-21]. Dostupné z: [https://en.wikipedia.org/wiki/Levenshtein\\_distance](https://en.wikipedia.org/wiki/Levenshtein_distance).

- [14] Google, Inc. *Google Cloud Platform* [online]. [cit 2016-12-21]. Dostupné z: <https://cloud.google.com/>.
- [15] Hermann Ebbinghaus *Memory: A Contribution to Experimental Psychology*. New York, 1913 [cit 25-12-2016]. ISBN 978-16-142-7166-6.
- [16] TAKADA Mikito. *Single page apps in depth* [online] 2014. [cit. 2016-12-27]. Dostupné z: <http://singlepageappbook.com/>.
- [17] KOPPERS Tobias a spol. *Webpack 2 Documentation* [online] 2016. [cit. 2016-12-26]. Dostupné z: <https://webpack.js.org/configuration/>.
- [18] SUSCAK Marek a spol. *Learn ES2015* [online] 2016. [cit. 2016-12-27]. Dostupné z: <https://babeljs.io/learn-es2015/>.
- [19] Ecma International. *ECMAScript® 2015 Language Specification* [online] 2015 [cit. 2016-12-27]. Dostupné z: <http://www.ecma-international.org/ecma-262/6.0/index.html>.
- [20] Facebook Inc. *JSX In Depth* [online] 2016 [cit. 2016-12-28]. Dostupné z: <https://facebook.github.io/react/docs/jsx-in-depth.html>.
- [21] Facebook Inc. *React Components and Props* [online] 2016 [cit. 2016-12-28]. Dostupné z: <https://facebook.github.io/react/docs/components-and-props.html>.
- [22] WESTSTRATE Michel a spol. *MobX Core Concepts* [online] 2016 [cit. 2016-12-28]. Dostupné z: <https://mobxjs.github.io/mobx/index.html>.
- [23] Django Software Foundation. *Django documentation* [online] 2015 [cit. 2016-12-28]. Dostupné z: <http://www.docs.djangoproject.com/en/>.
- [24] CHRISTIE Tom. *Django REST framework* [online] 2015 [cit. 2016-12-28]. Dostupné z: <http://www.django-rest-framework.org/>.
- [25] Open Web Application Security Project. *Cross-Site Request Forgery* [online] 2016-11-02 [cit. 2016-12-29]. Dostupné z: [https://www.owasp.org/index.php/Cross-Site\\_Request\\_Forgery\\_%28CSRF%29](https://www.owasp.org/index.php/Cross-Site_Request_Forgery_%28CSRF%29).
- [26] Open Web Application Security Project. *Cross-site Scripting (XSS)* [online] 2016-11-02 [cit. 2016-12-29]. Dostupné z: <https://www.owasp.org/index.php/XSS>.
- [27] Auth0 Inc. *Introduction to JSON Web Tokens* [online] 2016-11-02 [cit. 2016-12-29]. Dostupné z: <https://jwt.io/introduction/>.

## Přílohy

### A Ukázky kódů

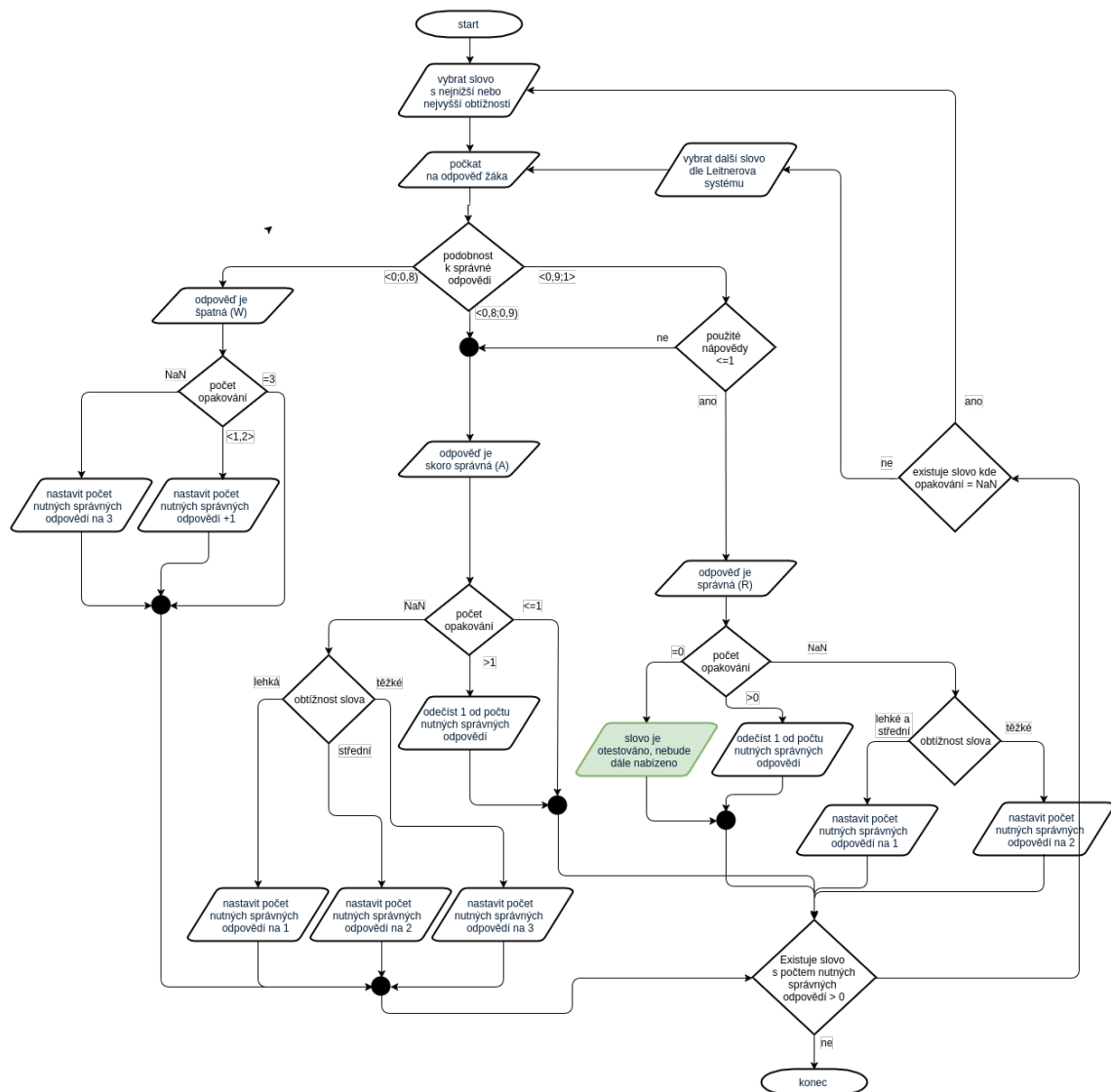
Kompletní zdrojové kódy obou aplikací jsou dostupné včetně instalačních návodů na:

- <https://daniel-madera.github.io/pijs/> - *frontend (klientská část)*
- <https://github.com/daniel-madera/pijs-api/> - *backend (serverová část)*

```
1 String.prototype.levenshtein = function(string, removeDiacritics = false,
  replace = {}) {
2   let a = this,
3       b = string + "",
4       m = [],
5       i, j, min = Math.min,
6       eq = false,
7       tmpB, tmpA
8   if (!(a && b)) return (b || a).length
9   if (a.length > b.length) {
10    var tmp = a
11    a = b
12    b = tmp
13  }
14  if (removeDiacritics) {
15    a = a.removeDiacritics()
16    b = b.removeDiacritics()
17  }
18  for (i = 0; i <= b.length; m[i] = [i++]);
19  for (j = 0; j <= a.length; m[0][j] = j++);
20
21  for (i = 1; i <= b.length; i++) {
22    for (j = 1; j <= a.length; j++) {
23      tmpA = a.charAt(j - 1)
24      tmpB = b.charAt(i - 1)
25      eq = tmpA === tmpB
26      if (replace && !eq) {
27        eq = replace[tmpA] ? (replace[tmpA] === tmpB) : (replace[tmpB] ===
          tmpA)
28      }
29      m[i][j] = eq ? m[i - 1][j - 1] : m[i][j] = min(
30        m[i - 1][j - 1] + 1,
31        min(m[i][j - 1] + 1, m[i - 1][j] + 1))
32    }
33  }
34  return m[b.length][a.length]
35 }
```

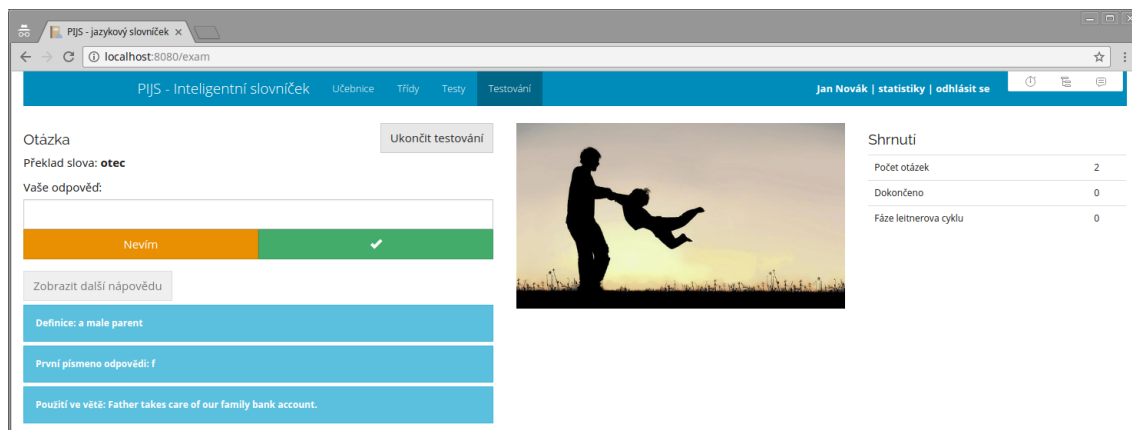
Ukázka kódu 9: Implementace algoritmu pro výpočet Levenshteinovy vzdálenosti

## B Diagramy

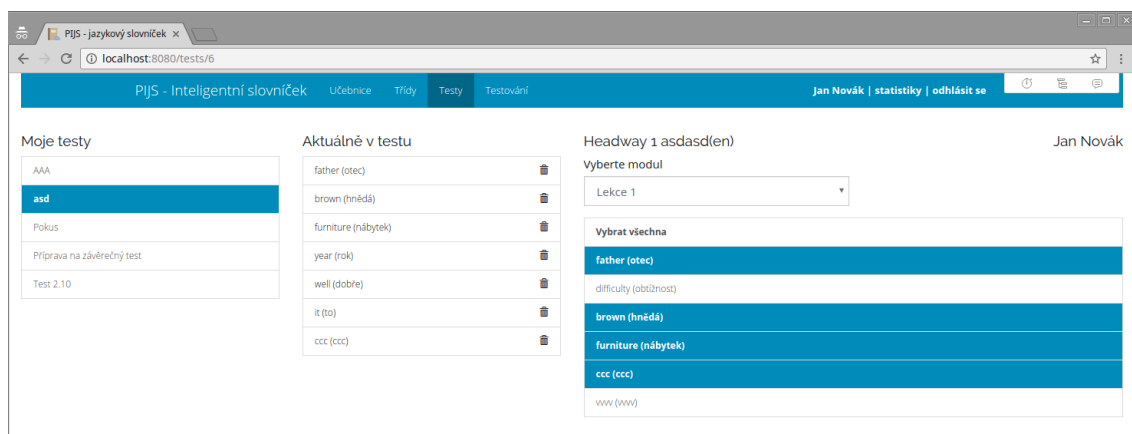


Obrázek 6: Vyhodnocování odpovědi v procvičování slov

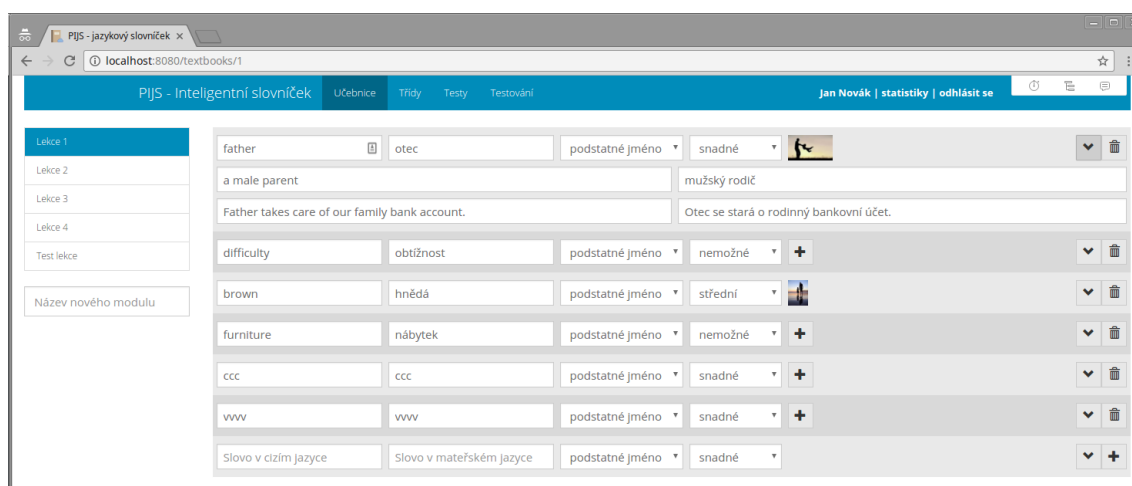
## C GUI



Obrázek 7: GUI testovací stránky



Obrázek 8: GUI výběr slov pro testování



Obrázek 9: GUI tvorby slov v učebnici



## D Obsah přiloženého média

/	
dp-madera-2017.pdf	Text diplomové práce ve formátu pdf
git-link.txt	Odkazy na Github repozitáře se zdrojovými kódy aplikací
assets	Obrázky a pdf použité v diplomové práci.
zadani.pdf	
db-model.pdf	
use-case.pdf	
...	
tex	L <sup>A</sup> T <sub>E</sub> Xové zdrojové kódy práce.
dp-madera.tex	
dp-titlepage.tex	
dp-prohlaseni.tex	