Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Mobilní aplikace pro výuku anglických slovíček

2019

Aleš Trunda

Vedoucí práce: Mgr. Jiří Zacpal, Ph.D.

Studijní obor: Aplikovaná informatika, kombinovaná forma

Bibliografické údaje

Autor:	Aleš Trunda
Název práce:	Mobilní aplikace pro výuku anglických slovíček
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2019
Studijní obor:	Aplikovaná informatika, kombinovaná forma
Vedoucí práce:	Mgr. Jiří Zacpal, Ph.D.
Počet stran:	57
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliograhic info

Author:	Aleš Trunda
Title:	Mobile application for learning English vocabulary
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2019
Study field:	Applied Computer Science, combined form
Supervisor:	Mgr. Jiří Zacpal, Ph.D.
Page count:	57
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

Bakalářská práce pojednává o vývoji aplikace pro výuku anglických slovíček. Pro tento účel je vyvíjena jak mobilní tak webová aplikace. Aby vývoj nebyl příliš náročný, využívá se externích služeb dostupných přes internet. Jsou navrženy různé postupy, které lze pro vývoj použít. Pomocí vhodného řešení je vytvořena webová a mobilní aplikace.

Synopsis

This bachelor's thesis deals with developing an application for learning English vocabulary. For that purpose mobile and web applications are developed. To prevent the work from getting too complex, online external services are used. Various solutions are presented. Using appropriate methods web and mobile applications are developed.

Klíčová slova: mobilní aplikace; webová aplikace; studium angličtiny; slovník; javascript; react; react native

Keywords: mobile application; web application; learning English; vocabulary; javascript; react; react native

Rád bych věnoval poděkování Mgr. Jiřímu Zacpalovi, Ph.D. za vedení a dozor nad celým projektem a také open source komunitě, jejíž software v práci hojně využívám.

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

1 Úvod **8**

2 Rozbor zadání **8**
2.1 Průzkum podobných řešení . . . 9
2.2 Možná řešení . . . 9
2.2.1 Webová aplikace . . . 10
2.2.1.1 Serverově renderovaná aplikace . . . 10
2.2.1.2 Single Page Application . . . 11
2.2.2 Mobilní aplikace . . . 11
2.2.2.1 Nativní vývoj . . . 12
2.2.2.2 Multiplatformní vývoj . . . 12
2.2.3 Progressive Web App . . . 13

3 Zvolené řešení **13**
3.1 HTML/CSS . . . 14
3.1.1 Absence programových prvků . . . 14
3.1.2 Složitý zápis . . . 14
3.1.3 Globální rozsah selektorů . . . 15
3.2 JavaScript . . . 16
3.2.1 React . . . 16
3.2.2 Flux . . . 18
3.2.3 React Native . . . 18
3.3 Vzhled . . . 19
3.4 Výuka . . . 20
3.4.1 Slovník . . . 20
3.4.2 Forgetting Curve . . . 20
3.4.3 Spaced Repetition . . . 21
3.4.4 Vybraná funkcionalita . . . 21

4 Výuka **21**
4.1 Přihrádky . . . 22
4.2 Seznamy . . . 23
4.3 Lekce . . . 23

5 Průběh vývoje **24**
5.1 Návrh vzhledu . . . 24
5.1.1 Wireframes . . . 25
5.1.2 Uživatelské rozhraní . . . 25
5.2 Workflow . . . 26
5.3 Podpůrné nástroje a knihovny . . . 28
5.3.1 Balíčky . . . 28
5.3.2 Sestavení kódu . . . 29
5.3.3 Ladění . . . 30

5.4 Programová část . . . 31
5.4.1 Správa dat . . . 31
5.4.2 Komponenty . . . 32
5.4.3 Externí služby . . . 32
5.5 Server . . . 33
5.6 Mobilní aplikace . . . 33
5.6.1 Styled Components . . . 33
5.6.2 Podpora zařízení . . . 34
5.6.3 Publikování . . . 35
5.7 Webová aplikace . . . 35
5.7.1 HTML a CSS . . . 35
5.7.2 Podpora webových prohlížečů . . . 36
5.7.3 Publikování . . . 36
5.8 Testování . . . 36
5.9 Další vývoj . . . 37

Závěr **38**

Conclusions **39**

A Uživatelská příručka **40**
A.1 Použití . . . 40
A.2 Menu . . . 40
A.3 Ukládání uživatelských dat . . . 41
A.4 Slovník . . . 42
A.5 Přidat slovíčko . . . 43
A.6 Vyhledávání . . . 44
A.7 Nastavit hodnocení slovíčka . . . 45
A.8 Přidat frázi . . . 46
A.9 Vytvořit seznam slovíček . . . 47
A.10 Nastavit slovíčka v seznamu . . . 47
A.11 Vytvořit lekci . . . 48
A.12 Spustit lekci . . . 49

B Obsah přiloženého CD/DVD **52**

Seznam zkratek **54**

Literatura **55**

Seznam obrázků

1 Podpora *Web App Manifest* v čase psaní této práce 13
2 Flux architektura 18
3 Forgetting curve 20
4 Leitner System 21
5 Detail slovíčka „Bear“ 22
6 Seznam „Animals“ 23
7 Detail lekce „Animal audio“ 24
8 Vyhodnocení po skončení lekce 25
9 Obrázková lekce, špatná odpověď 26
10 Obrázková lekce s nápovědou 27
11 Lekce typu fráze s obrázkovou nápovědou 28
12 Wireframes - úvodní strana 29
13 Wireframes - strana slovíčka 30
14 Kroky při publikování webové aplikace 31
15 Menu 41
16 Přihlášení na Google Drive 42
17 Stažení slovníku 43
19 Přidání slovíčka do oblíbených 44
20 Vyhledávání slovíčka 45
22 Nastavení hodnocení slovíčka 46
23 Přidání fráze ke slovíčku 47
24 Seznam slovíček 47
26 Nastavení slovíček v seznamu 48
27 Vytvoření lekce 49
30 Spuštění a průběh lekce 51

Seznam tabulek

Seznam vět

Seznam zdrojových kódů

1 CSS 15
2 SASS 15
3 CSS 16
4 JSX 17
5 JavaScript 17
6 CSS 34
7 CSS 34
8 SASS 35

1 Úvod

Cílem práce je vytvoření aplikace pro výuku anglických slovíček. K zajištění široké dostupnosti pro uživatele budu vytvářet jak mobilní tak webovou aplikaci. Vytvořením mobilní aplikace získám dostupnost na mobilních zařízeních a to i v režimu offline. Obecnou dostupnost pak zajistí webová aplikace. Co se týče mobilní verze cílím na operační systém Android a aplikaci budu publikovat na Google Play. Jelikož uživatel může využívat jak mobilní tak webovou aplikaci, budu se také zabývat jejich vzájemnou synchronizací.

Pro potřeby výuky bude aplikace využívat slovník s definicemi slovíček a obsahovat různou funkcionalitu pro pohodlnou práci se slovíčky a frázemi. Vhodnými metodami výuky budu usilovat o to, aby studium slovíček bylo pro uživatele efektivní.

Vzhledem k velikosti slovníku není reálné, aby byl součástí webové aplikace. Proto uvažuji o použití veřejně dostupné služby, která by sloužila jako zdroj dat pro slovník.

Aplikaci chci vyvíjet tak, aby byla moderní a robustní. Také bych se zaměřil na to, aby byla jednoduše udržovatelná a do budoucna rozšiřitelná. Robustnost zajistím správným návrhem s prostorem pro další vývoj. Vzhledem k množství technologií, které přichází v úvahu, budu předpokládat, že ty populární jsou prověřené a tedy budou dobrá volba.

V následujících kapitolách se snažím shrnout typické postupy používané při vývoji aplikací tohoto typu. Vybranému řešení se věnuji podrobně. Vzhledem k rychlému vývoji, kterým oblast webů a i mobilních aplikací prochází, zaměřuji se na ty technologie a postupy, které jsou moderní a perspektivní. V dalších částech práce pak dokumentuji konkrétní kroky vývoje a použité metody výuky.

2 Rozbor zadání

V první řadě bude rozumné volit technologie tak, aby se vývoj mobilní a webové aplikace pokud možno překrýval, protože po stránce funkcionality a vzhledu by aplikace měly být prakticky totožné. Pokud bych takového překrytí dosáhl, samotný vývoj by byl o to jednodušší. Samozřejmě ale musím přihlížet k tomu, aby tím zvolené řešení neztratilo na kvalitě.

U aplikací tohoto typu hraje podstatnou roli kvalitní uživatelské rozhraní. Na vývoji aplikací se obyčejně podílí tým několika lidí s různou specializací. Návrh uživatelského rozhraní a designu obecně bývá v režii grafika, tedy pro mě z pohledu programátora bude rozumné v tomto směru hledat nějaké usnadnění.

Další otazník pak přináší nutnost synchronizace webové a mobilní části. Vyvíjená aplikace by mohla běžet na mobilním telefonu, respektive ve webovém prohlížeči a data pro slovník by si aplikace nezávisle čerpaly z nějaké veřejně dostupné služby. Ovšem pro potřeby synchronizace bude třeba ještě další část, která bude sloužit jako prostředník mezi aplikacemi. Tato část bude muset zajistit perzistentní uložení uživatelských dat a také autentizaci uživatelů. To je

ale již komplexní problém, jelikož je třeba řešit zabezpečení uživatelských dat. Nabízí se tudíž využít nějaké veřejně dostupné úložiště třetí strany, kde by tato funkcionalita již byla připravena.

Co se týče výuky budu cílit na pokročilejší uživatele, kteří mají zájem o procvičení či zdokonalení své slovní zásoby. Pro tyto potřeby bude aplikace umožňovat plnohodnotnou správu slovíček a možnost rychlého procvičování, jelikož především v tomto směru vidím u existujících aplikací nedostatky.

2.1 Průzkum podobných řešení

Existuje obrovské množství webových či mobilních aplikací pro výuku slovní zásoby. Některé z nich jsou velice robustní, fungují již řadu let a za jejich vznikem stojí celé týmy vývojářů. Ovšem v žádné z testovaných aplikací jsem nenašel přímo požadovanou funkcionalitu a to, aby si uživatel mohl slovíčka zopakovat velice rychle prakticky pouhým pohledem.

Typický příklad: jsem v angličtině pokročilý, mám vybraný okruh slovíček, ve kterých bych se chtěl zdokonalit. Když mám volnou chvíli, chci si aplikaci otevřít a v rychlosti si procvičit několik slovíček.

Ty jednoduché aplikace nenabízejí plnohodnotnou správu slovní zásoby a ty složité mají výuku příliš komplikovanou – nutí uživatele odpovídat na kvízy, dopisovat slovíčko do vět, splnit celý okruh výuky a podobně, což je časově náročné. Samozřejmě pro začátečníka je tento styl výuky dobrý, ovšem pro pokročilého uživatele, který má zájem si v krátkém čase zopakovat pár vybraných slovíček, je to nevhodné. Ten je schopen letmým pohledem okamžitě rozhodnout, zda slovíčko ovládá nebo ne a tedy vyplňování nějakých kvízů a procházení celého okruhu slovíček je pro něj častokrát pouze zbytečné zdržení.

V rámci této práce nelze konkurovat rozsáhlým aplikacím jako je například *Duolingo* (webová i mobilní aplikace) nebo *Vocabulary.com* (webová aplikace). Ovšem právě obsáhlost těchto aplikací často snižuje jejich přehlednost a použitelnost – uživatel musí provést několik kroků než se dostane do požadované části aplikace. Proto se také chci zaměřit na to, aby vyvíjená aplikace byla jednoduchá, přehledná a aby obsluha byla co nejrychlejší. Tedy žádné úvodní stránky a složité proklikávání k lekcím, chci aby hned na první stránce byl seznam slovíček, které si uživatel může okamžitě začít procvičovat.

Neopomeňme také, že ty obsáhlé aplikace často využívají reklam a placeného obsahu, čemuž bych se chtěl ve své aplikaci pokud možno vyhnout.

2.2 Možná řešení

Jak oblast webů tak mobilních aplikací je velice populární a rychle se vyvíjí. Jednoduše lze ztratit přehled v obrovském množství frameworků a knihoven [1], které neustále vznikají a pouhá volba vhodného programovacího jazyka je nelehký úkol.

Budu volit perspektivní a moderní technologie. Ideálně takové, které je možné

použít pro vývoj webových i mobilních aplikací. Které konkrétně by přicházely v úvahu? To můžeme zjistit na nejpopulárnějším portálu pro vývojáře *Stack Overflow*, konkrétně z každoročního průzkumu[1] mezi desítkami tisíc vývojářů.

Z programovacích jazyků je patrná dominance JavaScriptu [2], podrobnější informace ohledně oblíbených frameworků a dalších nástrojů tohoto jazyka shrnuje například *The State of JavaScript*[2].

2.2.1 Webová aplikace

V posledních letech lze pozorovat velký rozkvět JavaScriptu. Byly vydány nové verze a z druhořadého programovacího jazyka se stal jeden z nejpoužívanějších. Objevily se různé frameworky a knihovny pro vývoj tzv. *Single Page Application* [3] (zkráceně SPA). Tento koncept přinesl nový pohled na vývoj webových aplikací a velice rychle se rozšířil.

Jak již výraz *Single Page Application* naznačuje, jedná se o aplikaci, která obsahuje pouze jednu stránku – princip spočívá v tom, že obsah stránky se vykresluje dynamicky pomocí JavaScriptu. Dojde tedy pouze k načtení první stránky a dále už k *reload* v prohlížeči nedochází. Veškerý další obsah se překresluje v rámci této jedné stránky.

Vytvoření HTML kódu je tedy řešeno na straně uživatele, čili na SPA můžeme nahlížet jako na šablonový systém běžící v prohlížeči. Před vznikem SPA se tento krok přípravy HTML řešil výhradně na straně serveru, který pro každou stránku uživateli připravil kompletní HTML kód. Pro toto serverové zpracování HTML se používají různá označení jako *tradiční aplikace*, *serverově renderovaná aplikace* nebo *Multiple Page Application*. Rozhodl jsem se používat pojem *serverově renderovaná aplikace*.

2.2.1.1 Serverově renderovaná aplikace

Vykreslování na straně serveru je tradiční metoda pro zobrazení webových stránek. Doposud to byla jediná metoda. Pro vykreslování na straně klienta je totiž nutné rozsáhlé využití JavaScriptu. Ten byl dlouhou dobu považován za podprůměrný jazyk a tudíž mu nebyla věnována přílišná pozornost. Upřednostňovaly se kvalitnější jazyky na straně serveru jako *php*, *.NET* nebo *Java*. Tento pohled se změnil s masivním rozvojem JavaScriptu v posledních letech.

Princip serverového vykreslování: jakmile server od klienta obdrží požadavek na zobrazení nějaké stránky, zpracuje kompletní HTML kód a ten klientovi pošle. Takto se zpracovává každá stránka, tedy pokud klient navštíví další stránku, opět dojde ke kompletnímu zpracování HTML kódu na straně serveru.

Stránky na jednom webu obyčejně nějaké části sdílejí, třeba hlavičku, proto by bylo výhodné překreslovat pouze ty části, které se mezi stránkami liší, nikoli vždy celou stránku. Takto ale serverové vykreslování nefunguje. Je tedy zřejmé, že v tomto směru existuje velký prostor pro zlepšení.

[1] https://insights.stackoverflow.com/survey/2018#technology
[2] https://2018.stateofjs.com/introduction

2.2.1.2 Single Page Application

U SPA je vytvoření HTML kódu řešeno na straně uživatele. Při úvodním načtení se stáhnou potřebné skripty a další zdroje, které aplikace využívá. Poté si už aplikace může ze serveru stahovat pouze syrová data, o zpracování dat a vytvoření HTML se postará prohlížeč. Díky tomu se snižují nároky na přenos dat i zátěž serveru.

Nicméně fakt, že je celá aplikace shrnuta do jedné stránky znamená, že to úvodní načtení může zahrnovat velký objem dat. Což především u větších aplikací znamená velkou prodlevu. Lze to řešit pomocí metody *Code Splitting* [4], tedy rozdělením kódu aplikace do více skriptů a načítáním dalších části až na vyžádání.

SPA obsahuje jednu vstupní HTML stránku, kde se z obsahového hlediska obyčejně nic nenachází, jelikož se vše generuje dynamicky. Aplikace zpravidla silně využívají technologii *AJAX*, díky které se načítá obsah ze serveru až podle akcí uživatele. Tento přístup ale velice komplikuje práci webovým vyhledávačům, jelikož ty při zobrazení HTML kódu vidí pouze prázdnou stránku. Procházet JavaScript je pro webové vyhledávače složité a také samozřejmě podstatně náročnější na zdroje oproti jednoduchému zobrazení HTML kódu. Nebudu se pouštět do široké problematiky *Search Engine Optimalization*, pro vyvíjenou aplikaci to není nijak důležité. Pouze bych zmínil, že existují nástroje pro vykreslování JavaScriptu na serveru [5] tak, aby odpověď pro klienta obsahovala odpovídající HTML kód.

Tento jednostránkový přístup má samozřejmě i světlé stránky. Celá aplikace se díky tomu může skládat pouze ze tří souborů: HTML stránka, soubor s JavaScriptem a soubor s CSS (při publikování aplikace jsou z důvodu optimalizace jednotlivé soubory spojeny do jednoho a minimalizovány).

Díky tomu, že SPA běží v prohlížeči, okamžitě reaguje na akce uživatele. Komunikace se serverem probíhá na pozadí, což je při práci s dynamickou aplikací značná výhoda oproti serverově renderovaným aplikacím, kde načtení nové stránky může trvat i několik sekund.

Nejpopulárnější SPA jsou: knihovna *React* [6], framework *Angular* [7] a framework *Vue* [8]. Pro lepší orientaci záměrně uvádím, zda se jedná o knihovnu nebo o framework, jelikož tyto pojmy jsou občas zaměňovány. Na serveru se k nim často používá kombinace *Node.js* a tzv. *NoSQL* databáze.

2.2.2 Mobilní aplikace

Především je třeba rozhodnout, na která zařízení, respektive na které operační systémy (zkráceně OS) se bude cílit. Prakticky stačí uvažovat pouze Android a iOS, na které dohromady připadá přibližně 99% podílu mobilních OS [9]. Tato aplikace je vyvíjena pro systém Android.

Dalším krokem je výběr programovacího jazyka. Variant je více, ty nativní jazyky jsou sice více robustní, ale tím také trpí vývoj, který je komplikovaný. Proto lze uvažovat různé alternativy, které se snaží vývoj zjednodušit – především

zaujmou nástroje umožňující multiplatformní vývoj [10].

Dokonce jsem narazil i na nástroje jako Dropsource[3] umožňující vytvořit aplikaci bez znalosti programování. Aplikace se jednoduše skládá v grafickém rozhraní pomocí *drag & drop*. Toto ale jistě nebude cesta pro vývoj robustní rozšiřitelné aplikace.

2.2.2.1 Nativní vývoj

Základní varianta je použít nativní jazyk, ten výchozí pro Android je Java [11], pro iOS pak Swift. Ale případně lze použít i jiné, například Objective C.

Nativní vývoj je plnohodnotné řešení, vzhledem k tomu, že je cílený na konkrétní platformu, mohou proběhnout různé optimalizace a obecně jsou takto vyvíjené aplikace velice rychlé. Vytknout lze závislost na OS a také především pro menší aplikace může být vývoj zbytečně složitý.

2.2.2.2 Multiplatformní vývoj

Je velice lákavé vyvinout jednu aplikaci, kterou by bylo možné s minimálním úsilím publikovat pro více OS. K tomuto účelu již vznikla řada nástrojů, většina z nich staví na webových technologiích (JavaScript, HTML a CSS). Funkcionalita zařízení je pak přístupná v JavaScriptu pomocí vnitřního API. Takové aplikace se označují jako hybridní [12]. Běží v jistém obalu označovaném jako *WebView*, který jim umožňuje chovat se jako nativní aplikace. Jedná se například o *Apache Cordova* nebo *Ionic Framework*.

Nicméně dělení nativní/hybridní není úplně přesné. Kupříkladu *React Native* [13] sice také využívá webové technologie, tedy bez HTML, nicméně neběží ve WebView a měl by se tudíž řadit mezi nativní. Vzhledem k podobnostem s hybridním vývojem je ale často označován také jako hybridní, nebo řekněme je v tomto směru na pomezí.

React Native, jak už název napovídá, je rozšíření k React, který byl zmíněn u vývoje SPA. I ostatní populární SPA frameworky/knihovny lze využít pro vývoj mobilních aplikací – pro Angular můžeme použít *NativeScript*, pro Vue pak *Weex*.

Pomocí zmíněných kombinací lze dosáhnout překrytí ve vývoji webové a mobilní aplikace. Především se nabízí možnost sdílení částí kódu – a to ne pouze JavaScript, ale i styly, knihovny apod. Webovou SPA nelze přímo použít jako mobilní, bude třeba počítat s jistými úpravami, případně vytvořením nadstavby. Nicméně princip samostatných komponent, který se v SPA používá, je z pohledu tohoto sdílení výhodný.

Ionic (nástroj pro vývoj hybridních aplikací, který staví na Angular) provedl mezi vývojáři průzkum[4], ze kterého je patrný odklon od nativního vývoje a preferování hybridních aplikací. Jako důvod vývojáři uvádějí možnost používat webové technologie.

[3] https://www.dropsource.com/

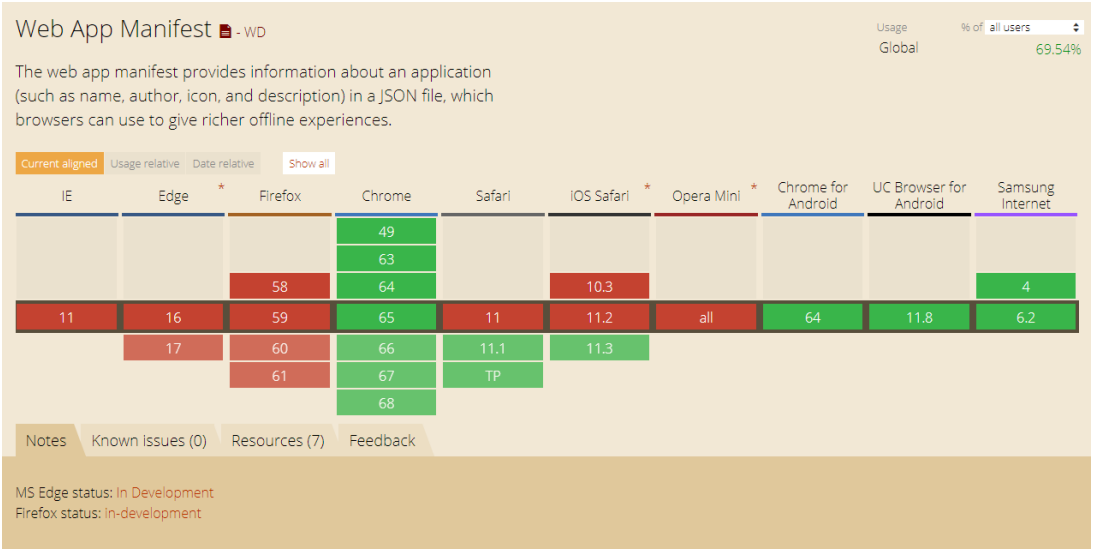
[4] https://ionicframework.com/survey/2017#trends

2.2.3 Progressive Web App

Progressive Web App [14] (zkratka PWA) je klasická webová aplikace rozšířená o funkčnost, kterou nabízí mobilní zařízení tak, že se PWA chová a vypadá jako nativní mobilní aplikace – lze ji instalovat, má přístup k vnitřnímu úložišti, dokáže pracovat i offline atd. Ovšem tento typ aplikací je stále relativně nový a funkčnost, kterou plnohodnotná PWA vyžaduje, jako *Service Worker* (umožňuje aplikaci fungovat i offline) nebo *Web App Manifest*, zatím není mezi prohlížeči a zařízeními plně podporována[5] (Obr. 1).

Nicméně již nyní lze narazit na názory[6], že PWA v budoucnu úplně vytlačí mobilní aplikace.

Výhody PWA jsou zřejmé: aplikace je dostupná na webu, čili indexovatelná webovými vyhledávači. Jedno řešení pro web i mobil. Není nutné ji umístit na služby typu Google Play. Lze přesně specifikovat její vzhled včetně responzivity. Je jednodušší na vývoj, publikování a vydávání nových verzí. Je rychlá a multiplatformní.



Obrázek 1: Podpora *Web App Manifest* v čase psaní této práce

3 Zvolené řešení

Pro webovou aplikaci jsem zvolil SPA za použití React a pro mobilní aplikaci React Native, vyvíjet tedy budu především v JavaScriptu.

JavaScript je jazyk primárně pro webový prohlížeč. Ovšem pouze samotným JavaScriptem žádnou rozumnou aplikaci vytvořit nelze, je nutné jej kombinovat s HTML a CSS. Čili pro vývoj smysluplných aplikací je nutné ovládat všechny

[5] https://caniuse.com/
[6] https://torquemag.io/2017/11/10-reasons-pwas-will-future-apps/

tyto technologie. A pro vývoj moderních a kvalitních aplikací je třeba ještě pracovat i se spoustou dalších technologií a nástrojů. Například si vezměme testy. Aplikaci lze vytvořit i bez testů, ale pokud se má jednat o kvalitní aplikaci, určitě by nějaké testy obsahovat měla. Pro testování existuje spousta frameworků: *Jest*, *Jasmine*, *Karma*, *Cypress*, *Chai*, *Mocha*, *Selenium*, *TestCafe*... Jeden se více hodí na použití s React, jiný na použití s Angular apod. Také záleží, zda máme zájem o nízkoúrovňové unit testy nebo end-to-end testy. Očividně je tedy možností více než dost.

Tímto stylem je kolem JavaScriptu vytvořena celá síť[7] různých frameworků, nástrojů a knihoven, o kterých vývojář musí mít přehled a alespoň s některými musí umět aktivně pracovat. Pro začínající vývojáře je potom velice náročné naučit se vše potřebné – v tomto smyslu se vžilo označení *JavaScript fatigue* [15].

3.1 HTML/CSS

Pro webovou aplikaci použiji HTML5, tam prakticky není čím více se zabývat. V React Native HTML použít nelze, ale k tomu až níže v sekci React Native.

Především se zde chci věnovat CSS, protože jeho naivní použití má několik citelných nedostatků, které je vhodné řešit. V základu se jedná o: absence programových prvků, složitý zápis, globální rozsah selektorů. Většinu z těchto problémů řeší CSS preprocesory, tím nejpopulárnějším je *SASS* [16], což je i moje preference.

CSS preprocesory jsou nástroje, které přinášejí novou logiku do vytváření CSS. Mají vlastní syntaxi velice podobnou CSS, pro SASS je to *SCSS* (Sassy CSS). Ta se následně kompiluje do CSS. Díky tomu je možné řešit zápis stylů podstatně robustněji a přehledněji.

3.1.1 Absence programových prvků

V moderním CSS je sice možné používat proměnné, ale další funkcionalita jako smyčky nebo rozhodovací direktivy k dispozici nejsou. Opět je možné řešit preprocesorem – SASS obsahuje *SassScript*, který je podstatně jednodušší než klasické programovací jazyky, ale základní konstrukce obsahuje: cykly, seznamy, funkce...

3.1.2 Složitý zápis

Především nemožnost vnořovat selektory je při větším objemu stylů omezující, dochází k častému opakování kódu. Tento problém řeší CSS preprocesory se svou vlastní robustnější syntaxí.

Uvažujme tento CSS kód:

[7] http://roadmap.sh

```
.button {
  color: #666;
}
.button.active {
  color: red;
}
.button.active:hover {
  background-color: green;
}
```

Zdrojový kód 1: CSS

Pomocí SASS jej můžeme zapsat podstatně efektivněji a čitelněji:

```
.button {
  color: #666;

  &.active {
    color: red;

    &:hover {
      background-color: green;
    }
  }
}
```

Zdrojový kód 2: SASS

Standardní zápis CSS také obsahuje spoustu bílých znaků, které pro prohlížeč nejsou potřeba, jejich odstraněním lze ušetřit desítky procent z původní velikosti. Odstranění nepotřebných znaků a případné optimalizování zápisu se označuje jako *minimalizace*. K tomuto účelu je možné použít různé nástroje jako *cssnano*.

3.1.3 Globální rozsah selektorů

Největší problémy v CSS plynou z faktu, že rozsah selektorů není nijak omezen. Styly ovlivňují všechny prvky na stránce, chovají se jako globální proměnné. Nevhodně strukturované CSS dříve nebo později způsobí, že styly nebudou kaskádovat tak, jak bylo zamýšleno, ale místo toho se budou aplikovat nějaká jiná nechtěná pravidla s vyšší vahou selektoru. Logické řešení je zvýšit váhu selektoru tak, aby došlo k přepsání těch nechtěných pravidel. Toto přepisování váhy se ale i v menších projektech rychle stane neudržitelné. Nejprve citelně klesá kvalita kódu a později to ústí až k nutnosti kompletního refaktorování.

Aby tyto problémy nevznikaly, je třeba CSS správně strukturovat. K tomuto účelu vznikly různé metodiky, např. *OOCSS*, *SMACSS* nebo *SUITCSS*. Já jsem

si oblíbil metodiku *BEM*[8]. Zkratka se skládá z trojice *Block*, *Element* a *Modifier*, což je vzorec pro pojmenování selektorů. V praxi by zápis selektorů podle BEM mohl vypadat takto:

```
.user {
  background-color: #666;
}
.user__avatar {
  width: 100px;
  height: 100px;
}
.user__avatar--compact {
  width: 75px;
  height: 75px;
}
```

Zdrojový kód 3: CSS

3.2 JavaScript

JavaScript v posledních letech prošel obrovským vývojem [17]. Vznikají nové standardy jazyka, především verze ES6 (ECMAScript 2015) z roku 2015 byla přelomová a přinesla spoustu vylepšení. Další verze ES8 pak obsahuje *async/await*, což citelně ulehčuje práci s asynchronními událostmi. Neustále vznikají nové frameworky, knihovny a nástroje, z těch populárních bych zmínil například: *npm*, *webpack*, *React*, *Angular*, *Vue*, *Material-UI*...

JavaScript se úspěšně používá i mimo hranice webového prohlížeče – v této práci vyvíjím mobilní aplikaci za použití JavaScriptu, dále jej lze použít také pro vývoj desktopových aplikací (např. nástroj *Electron*) nebo také na straně serveru, kde je populární systém *Node.js*.

Za tímto vším můžeme vidět světlou budoucnost JavaScriptu. Výběr tohoto programovacího jazyka je tedy jistě perspektivní volba.

3.2.1 React

Před prací s knihovnou React je třeba rozumět principům jejího fungování, jelikož špatně sestavená aplikace může být pomalá a neefektivní. Níže se věnuji několika pojmům, se kterými je vhodné se seznámit.

Uživatelské rozhraní se staví pomocí *komponent*, které lze skládat ve formě stromu. Komponenta je samostatný zapouzdřený celek, který má svůj vnitřní stav. Koncepčně se jedná o funkci. Vstupní data jsou komponentě předány pomocí parametrů. Komponenta implementuje metodu *render*, která vrací obsah

[8] http://getbem.com/

k vykreslení ve formě React komponenty. Tento obsah se obyčejně popisuje pomocí *JSX* syntaxe, což je syntaxe podobná *XML*. Při vykreslování React transformuje JSX do HTML. Tedy například JSX zápis:

```
<MyButton color="blue" shadowSize={2}>
  Click Me
</MyButton>
```

Zdrojový kód 4: JSX

je převeden na:

```
React.createElement(
  MyButton,
  {color: 'blue', shadowSize: 2},
  'Click Me'
)
```

Zdrojový kód 5: JavaScript

Přímá manipulace s *Document Object Model* (zkráceně DOM, což je stromová reprezentace HTML dokumentu) je výpočetně náročná operace, proto se ji React snaží provádět pouze v nutných případech. K tomu používá *Virtual DOM*, což je zjednodušená kopie DOM, se kterou se ale pracuje podstatně rychleji, protože Virtual DOM na rozdíl od DOM není navázán na prohlížeč, nevykresluje se. Proto React provádí změny ve Virtual DOM, ten následně porovná s DOM a až v případě, že narazí na rozdíl, upraví na potřebných místech DOM.

Překreslování komponent chceme provádět pouze pokud je to nutné, ovšem není jednoduché rozhodnout, zda by se měl objekt překreslit nebo ne. Objekt by se musel hloubkově projít a každou jeho hodnotu porovnat s tou předchozí, což by bylo výpočetně velice náročné. Proto se používá princip *Immutability*. Ten udává, že pokud chceme objekt měnit, nebudeme jej upravovat, namísto toho vytvoříme kopii tohoto objektu a tam zaneseme požadované úpravy. Porovnání takovýchto objektů je potom velice snadné – stačí porovnat jejich odkazy.

JavaScript je dynamicky typovaný jazyk, což může přispívat ke vzniku chyb, na druhou stranu absence typů zase dává větší volnost programátorovi. Existují nástroje jako *TypeScript*[9] nebo *Flow*[10], které zavádí typy do JavaScriptu. Tedy záleží na konkrétním projektu a na preferencích programátora, zda chce typy používat nebo ne. Jsou zastánci jak dynamického tak statického typování. Například Angular, jeden z těch nejpopulárnějších frameworků pro vývoj SPA, používá TypeScript. Ale samotný React nic takového nepoužívá, nicméně nabízí

[9] http://www.typescriptlang.org/
[10] https://flow.org/

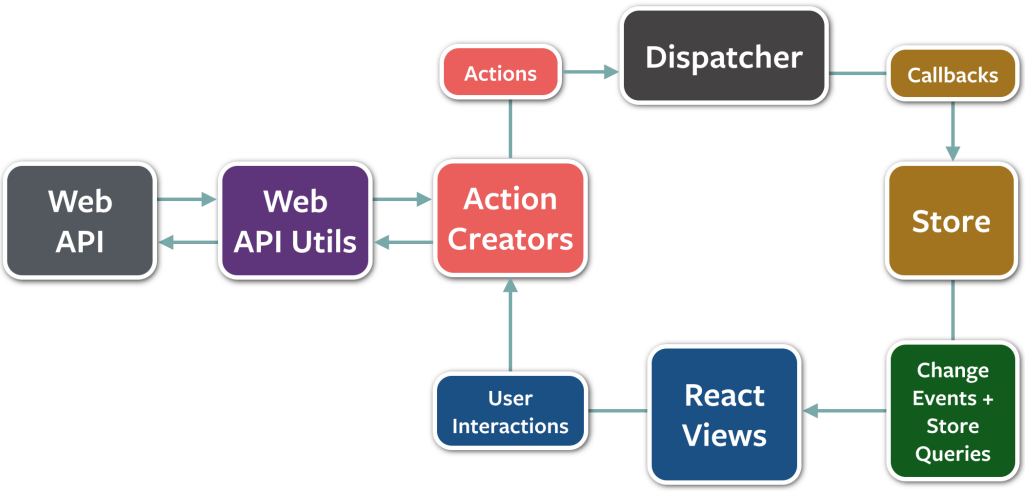
alternativu v podobě *PropTypes*. Pomocí PropTypes se kontrolují typy parametrů předávaných komponentě. Tedy je to takové částečné typování. Tuto variantu s PropTypes jsem se rozhodl použít.

3.2.2 Flux

Architektura MVC není vhodná pro aplikace typu SPA kvůli obousměrnému toku dat. V takovém případě by totiž změna v jedné komponentě mohla vyvolat změnu v jiné komponentě, ta by následně vyvolala další změny atd. Bylo velice složité aplikaci porozumět a nějak ji debugovat. U menších aplikací to nemusí být problém, ale ty větší by nebylo možné tímto stylem efektivně vyvíjet, proto byla navržena architektura *Flux* [18], která vymezuje tok dat pouze v jednom směru.

Jádro architektury Flux se skládá ze 4 částí (Obr. 2): *Action* (událost), *Dispatcher* (zpracování události), *Store* (uložení dat) a *View* (zobrazení). Jakmile uživatel provede nějakou akci ve View, vyvolá se událost Action, kterou zpracuje Dispatcher a podle potřeby upraví Store. Změna ve Store následně vyvolá překreslení odpovídajících komponent ve View.

Velice populární implementací Flux pro React je *Redux*, ten jsem se rozhodl použít.



Obrázek 2: Flux architektura

3.2.3 React Native

React Native má mezi vývojáři velkou popularitu a fakt, že jej používají i velké firmy jako *Airbnb* [19], značí, že je to rozumná volba.

React Native je velice podobný knihovně React, které jsem se již věnoval výše. Zde se proto zaměřím na rysy týkající se přímo React Native.

React Native nepoužívá HTML, místo toho obsahuje komponenty, které HTML elementy nahrazují, jsou to např. *Text* nebo *View*. HTML element *div* by bylo možné reprezentovat pomocí komponenty *View*, pro element *ul* by přicházela v úvahu komponenta *FlatList* a podobně.

Stejně tak nelze použít klasické kaskádové styly, respektive používají se pouze částečně – CSS musí být součástí JavaScriptu, což vyžaduje mírně odlišný zápis od syntaxe CSS. V porovnání se SASS tento přístup není příliš přehledný. Proto je často preferované řešení pomocí *Styled Components* [20], které umožňují robustnější práci s CSS v JavaScriptu. Právě Styled Components jsem se rozhodl použít pro práci se styly v mobilní aplikaci. Takto zapsané styly nejsou globální, ovlivňují pouze cílové komponenty a tedy v tomto případě ani není nutné zvažovat použití CSS metodik.

CSS v JavaScriptu není výsadou jen React Native, Styled Components bych mohl použít i v React při vývoji webové aplikace.

Další důležitá odlišnost týkající se CSS je v rozložení prvků, jelikož React Native se vymezuje pouze na použití Flexbox. Ovšem za tím nelze vidět nic negativního, jelikož Flexbox je moderní a robustní řešení, snad jen pokud bychom chtěli použít ještě pokročilejší řešení v podobě Grid. A také nejsou k dispozici některé další prvky z CSS jako třeba animace. Ty je možné realizovat použitím nějaké další knihovny.

Vzhledem k tomu, že je aplikace cílena jen na systém Android, možnostem multiplatformního vývoje se věnovat nebudu.

Posledním krokem ve vývoji je publikování. Webovou aplikaci stačí nahrát na webový server, ale u mobilní aplikace je to složitější. Ve výchozím nastavení OS ani neumožňuje instalovat aplikace pocházející z neověřeného zdroje a stejně chci, aby aplikace byla veřejně dostupná, v každém případě tedy bude nutné zajistit publikování na Google Play.

3.3 Vzhled

Z pohledu programátora usuzuji, že pouštět se do výroby vlastního designu nebude to pravé. Naštěstí povaha aplikace nevyžaduje design na míru na rozdíl kupříkladu od firemní aplikace, proto využiji veřejně dostupné knihovny a materiály.

Pro vytvoření uživatelského rozhraní jsou velice populární knihovny *Material-UI* a *React-Bootstrap* (druhá zmíněná staví na *Bootstrap*, což je obecně nejpopulárnější HTML/CSS/JS knihovna). Pro barevnou paletu vypadá zajímavě nástroj *Color Hunt*, ale možností je spousta: *Coolors*, *Paletton*... A pro rozložení prvků v aplikaci lze najít inspiraci například na *Mobile Patterns* nebo *Pttrns*.

Jelikož nemám stanoveny konkrétní požadavky na vzhled, předpokládám, že budu různě experimentovat. Před samotným vývojem je vhodné k navrženému vzhledu také získat odezvu od potencionálních uživatelů.

3.4 Výuka

Na aplikaci nelze nahlížet pouze po technické stránce, očekává se také, že bude mít odpovídající přínos pro uživatele. Pro výuku existuje spousta metod [21]. Jelikož cílem aplikace je výuka slovní zásoby, zaměřil jsem se především na techniky uložení slovíčka do dlouhodobé paměti uživatele.

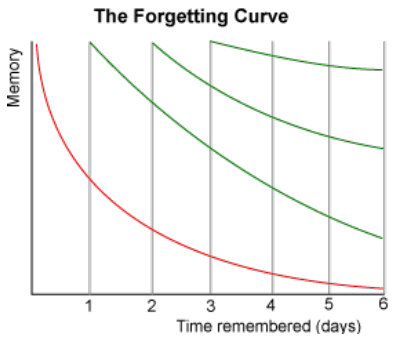
3.4.1 Slovník

Pro studium slovíček jsou vhodné jednojazyčné slovníky. Tyto slovníky obsahují popis významu slovíčka včetně výslovnosti, gramatiky a použití ve větě. Z mého pohledu jsou pro studium tyto slovníky podstatně efektivnější než překladové slovníky, které obsahují často i pouze jednoslovné překlady. Takový překlad jistě nemusí dostačovat pro pochopení významu slovíčka.

Při hledání vhodného jednojazyčného slovníku jsem objevil řadu možností, nicméně jelikož vyvíjená aplikace nemá k dispozici žádný rozpočet, je třeba omezit se na ty slovníky, které je možné využívat zdarma. Velice se mi líbí online služby jako *Oxford Dictionaries*[11], které poskytují podrobnou definici slovíčka včetně audio ukázky s výslovností, možnost vyhledávat v databázi nebo třeba i nalezení synonym. Tyto služby jsou sice placené, ale obyčejně nabízejí i nějakou základní verzi zdarma, která by mohla být dostačující. Druhá možnost je použít nějakou databázi slovíček, nejkvalitnější mi přijde *WordNet*[12].

3.4.2 Forgetting Curve

Hypotéza psychologa Hermanna Ebbinghause označovaná jako *Forgetting Curve* [22] pojednává o ztrátě informace v paměti vzhledem k času uplynulému od uložení nebo zopakování této informace. H. Ebbinghaus se věnoval tomu, jak je vhodné informaci opakovat, tak aby byla uložena v paměti po dlouhou dobu. Z obrázku 3 je zřejmé, že s vyšším počtem opakování je míra ztráty naučených informací menší.



Obrázek 3: Forgetting curve

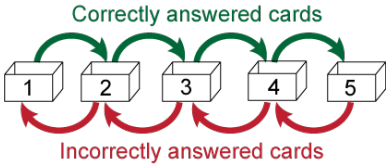
[11] https://www.oxforddictionaries.com/
[12] https://wordnet.princeton.edu/

3.4.3 Spaced Repetition

H. Ebbinghaus zavedl jev označovaný jako *Spacing Effect* [23], který se vztahuje k efektivitě učení. Pojednává o tom, že učení je efektivnější, když je rozprostřeno do delšího časového úseku a v jistých intervalech se opakuje.

Na této myšlence staví technika učení označovaná jako *Spaced Repetition*. Je to technika, kdy se studovaná látka následně v určitých intervalech opakuje. Intervaly opakování mohou být: 20 min, 1 hodina, 9 hodin, 1 den, 2 dny, 31 dní.

Leitner System [24] je populární metoda výuky, která využívá Spaced Repetition a kartiček (*Flashcard*, což je pomůcka k učení, kde na jedné straně kartičky je otázka a na druhé straně odpověď). Princip metody spočívá v rozdělování kartiček do přihrádek (Obr. 4). Přihrádek může být libovolné množství, například pět. Pokud je kartička zodpovězena správně, posouvá se do vyšší přihrádky, pokud je zodpovězena špatně, posouvá se do nižší přihrádky. Čím nižší přihrádka, tím častěji se její kartičky znovu procvičují.



Obrázek 4: Leitner System

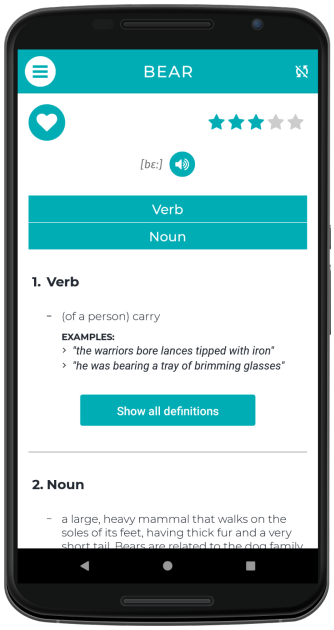
3.4.4 Vybraná funkcionalita

Na základě zmíněných faktů jsem se rozhodl v aplikaci implementovat tuto funkcionalitu:

- jednojazyčný slovník
- uložení slovíčka do oblíbených
- seskupování slovíček do seznamů
- hodnocení slovíčka
- dělení slovíček do přihrádek z hlediska hodnocení i času
- metoda kartiček
- použití slovíčka ve větě

4 Výuka

Základní forma výuky, kterou aplikace nabízí je vyhledání definice slovíčka. Data z Oxford Dictionaries, což je primární zdroj pro slovník, typicky obsahují: definice, slovní druh, výslovnost, audio a příklad použití ve větě (Obr. 5).



Obrázek 5: Detail slovíčka „Bear“

Věřím, že když je na začátku aplikace prázdná, je pro uživatele složité s ní začít pracovat. Vyřešil jsem to tak, že při otevření prázdné aplikace se uživateli nabídne několik připravených seznamů se slovíčky, které si může do svého slovníku importovat a hned tak začít s procvičováním. Převážně se jedná o volně dostupné seznamy nejpoužívanějších slovíček v angličtině.

Jinak ale ponechávám naplnění slovníku na uživateli. Předpokládám, že si sám nastaví, která slovíčka by chtěl procvičovat. Stejně tak nechci uživatele tlačit do studia pomocí pevných rozvrhů a při nesplnění jej upozorňovat notifikacemi a podobně. Smyslem aplikace je, aby si uživatel ve chvíli volna mohl procvičit pár slovíček.

4.1 Přihrádky

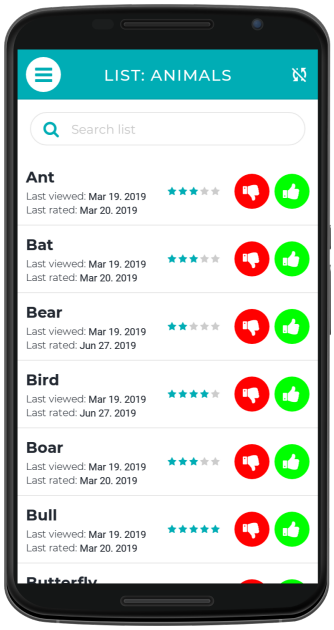
Vychází se z hodnocení slovíčka, což je hodnota v rozmezí 1-5, kde 5 je nejvyšší. Uživatel si hodnocení může nastavovat sám, nebo se hodnocení nastavuje automaticky během lekce v závislosti na správně/špatně zodpovězené otázce. V případě, že slovíčko ještě nebylo hodnoceno, rozlišuji i hodnocení 0. Čili podle hodnocení jsou slovíčka rozdělena do šesti přihrádek. Uživatel se může zaměřit na konkrétní přihrádku, nebo si třeba seznam slovíček seřadit podle hodnocení a zaměřit se více na ta s nižším hodnocením.

Pro doplnění jsem přidal i přihrádky pro rozdělení slovíček z pohledu času. Zde už se tedy nejedná o striktní rozdělení, jelikož jedno slovíčko může spadat do více přihrádek. Rozlišuji akce *zobrazení slovíčka* a *přidání slovíčka* a k nim časové úseky 1 týden, 1 měsíc a 3 měsíce.

4.2 Seznamy

Především u obsáhlejšího slovníku je výhodné seskupovat si slovíčka do seznamů. Uživatel se tak může jednoduše zaměřit na konkrétní část slovíček. Například všechna slovíčka zvířat si může zařadit do seznamu *Animals* (Obr. 6).

Aplikace obsahuje také výchozí seznam se všemi slovíčky a automaticky generované seznamy ze zmíněných přihrádek.



Obrázek 6: Seznam „Animals“

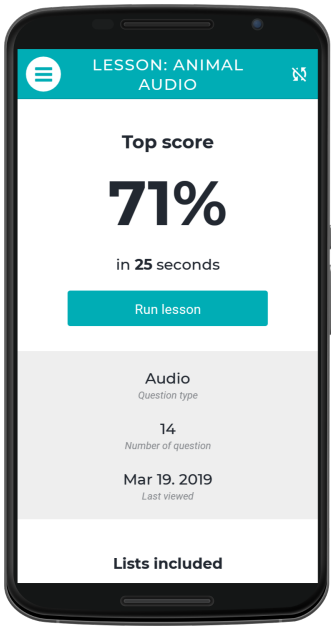
4.3 Lekce

Uživatel si může procvičovat procházením seznamů, nebo pro efektivnější studium si může vytvořit lekci. U lekce se specifikuje typ otázek, počet otázek a seznamy, ze kterých se budou náhodně vybírat slovíčka. Smyslem lekce rozumím sledování vývoje znalostí v čase, proto se u lekcí zaznamenává nejvyšší skóre, jak je vidět v detailu lekce (Obr. 7).

Rozhodl jsem se přidat i možnost anonymních lekcí, kde si uživatel jako seznamy vybere přihrádky. Může si tedy spustit lekci například pouze se slovíčky, která mají nižší hodnocení než 3. Anonymní lekce se neukládají, protože slovíčka v přihrádkách se velice často mění a nejvyšší skóre by tak nebylo objektivní.

Procvičování je založeno na metodě kartiček, kde na jedné straně je otázka, na druhé odpověď, což je procvičované slovíčko. Ve výchozím nastavení dojde k vyhodnocení až po skončení lekce (Obr. 8), nebo si lze přepnout na vyhodnocování po každé otázce (Obr. 9). K dispozici jsou tyto typy otázek: audio, obrázek, fráze, překlad, slovíčko a vlastní poznámka.

Na obrázku 11 je vidět otázka typu fráze, na které si lze dobře procvičovat použití slovíčka ve větě.



Obrázek 7: Detail lekce „Animal audio“

Některé typy otázek nemusí mít jednoznačnou odpověď. Uvažujme obrázek brouka, jako správnou odpověď bychom akceptovali „beetle“ i „bug“. To ale komplikuje proces vyhodnocení, protože pokud má uživatel ve slovníku jen jedno z těchto slovíček, nalezení toho druhého je problematické. Abych tyto nejednoznačnosti eliminoval, přidal jsem k otázce možnost jednoduché nápovědy. Na výběr je několik možností, například zobrazení počtu písmen ve slovíčku (Obr. 10).

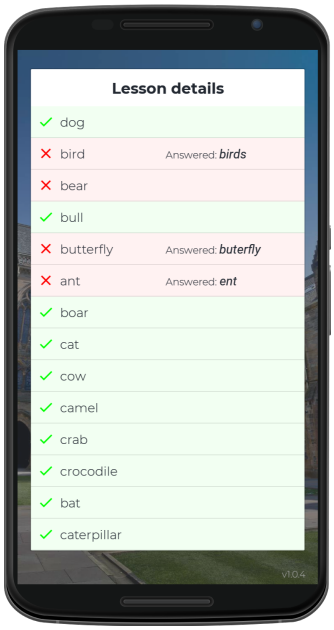
5 Průběh vývoje

V této kapitole dokumentuji jednotlivé kroky vývoje, které jsou víceméně společné pro webovou i mobilní část. Rozdílným částem se věnuji v sekcích „Webová aplikace“ a „Mobilní aplikace“.

Nejprve jsem vytvořil webovou aplikaci, jelikož vývoj v prostředí webového prohlížeče je pohodlnější a umožňuje lepší debugování. Následně jsem podle téměř finální webové aplikace vytvářel tu mobilní.

5.1 Návrh vzhledu

Volil jsem již zpracované volně dostupné materiály a standardní vzhled [25]. Myslím, že nevhodný vzhled může aplikaci citelně uškodit, proto jsem dal přednost průměrnému designu, než se pouštět do vypracování vlastního s nejasným výsledkem.



Obrázek 8: Vyhodnocení po skončení lekce

5.1.1 Wireframes

Jako první krok se obyčejně zpracují jednotlivé stránky pomocí *wireframes*, jimiž se velice jednoduše bez stylizace znázorní rozložení stránek.

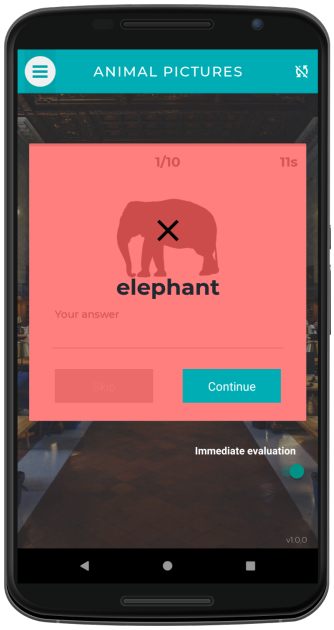
Pro jednoduchost jsem se zaměřil na nástroje dostupné online a zpracoval jsem si několik stran v *Mockflow*[13]. Při práci s těmito nástroji lze jednoduše narazit na problém, se kterým se vývojář nesetkává příliš často a to používání placených nástrojů. Díky velké komunitě programátorů, kteří se podílejí na vývoji open source softwaru, je totiž možné pohodlně vyvíjet s volně dostupnými nástroji. Jak jsem zjistil oblast designu je v tomto směru odlišná – většina nástrojů je zpoplatněná. Mnohdy je sice nabízena i varianta zdarma, ale ta obyčejně umožňuje jen velice omezené použití, které nedostačuje ani na jednodušší projekt. Verze zdarma je tedy opravdu pouze k vyzkoušení nástroje, nikoli k praktickému použití.

5.1.2 Uživatelské rozhraní

Přípravu designu můžeme rozdělit do tří kroků: výběr barevné palety, rozložení prvků, stylizace prvků. Pro výběr barev jsem použil *Color Hunt*[14], pro rozložení prvků jsem si různě našel inspiraci na *Mobile Patterns*[15].

V uživatelském rozhraní jsem použil běžné prvky jako tlačítko, zaškrtávátko, přepínač..., které jsou přítomny prakticky v každé aplikaci, tudíž existuje spousta

[13]https://www.mockflow.com/
[14]https://colorhunt.co/
[15]http://mobile-patterns.com/



Obrázek 9: Obrázková lekce, špatná odpověď

open source knihoven, kde jsou tyto prvky v nějakém základním vzhledu zpracovány. Stačí pouze sladit barevnou paletu. Pro webovou aplikaci jsem si vybral velice populární knihovnu *Material-UI*[16] a pro mobilní aplikaci knihovnu *React Native Elements*[17].

5.2 Workflow

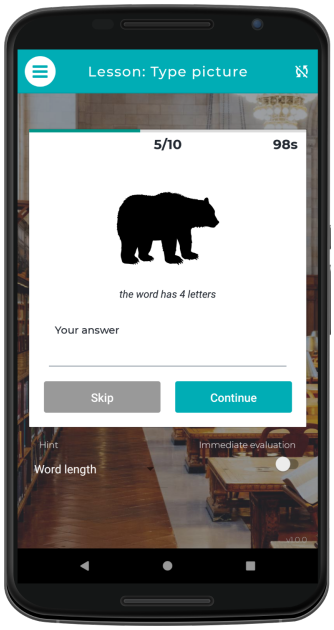
Na vývoji podobných aplikací se obyčejně podílí tým několika lidí. Jejich práci je nutné vhodně organizovat.

Mezi programátory je rozšířeno více stylů zápisu kódu, jedná se například o středníky (v JavaScriptu nejsou povinné) nebo o velikost odsazení. Je složité přesně rozhodnout, který zápis je správný a který je špatný, proto vzniklo několik standardů, které definují korektní zápis kódu. Při spolupráci více vývojářů je pro čitelnost kódu velice důležité udržovat napříč všemi stejný standard. Naštěstí samotný programátor se tím příliš zabývat nemusí, stačí správně nastavit nástroje jako *ESLint* a *Prettier*, které zajistí korektní zápis a automaticky kód upraví podle zvoleného standardu.

Jistě bych se mohl věnovat také procesu vývoje, velice populární jsou agilní metodiky jako *scrum* nebo *kanban*. Také by přicházelo v úvahu třeba *code review* apod. Ale nic takového jsem nepoužil. Pro jednotlivce a v tomto typu projektu to příliš nedává smysl.

Rád bych se ale věnoval *Continous integration* (průběžné začlenění změn, zkratka CI) a *Continuous deployment* (průběžné publikování). Jak webovou tak

[16] https://material-ui.com/
[17] https://github.com/react-native-training/react-native-elements



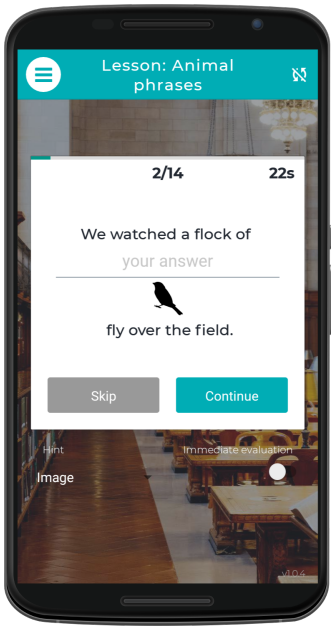
Obrázek 10: Obrázková lekce s nápovědou

mobilní aplikaci je nutné před publikováním sestavit. Často se aplikace publikuje už během vývoje, aby se k uživatelům dostala co nejdříve i jen částečně funkční verze. A samozřejmě po publikování se následně objevují různé chyby, které je třeba průběžně opravovat. Což ústí v to, že se aplikace může sestavovat a publikovat i vícekrát za den. Tento krok je rozumné zautomatizovat.

Obrázek 14 znázorňuje kroky, které provádím při publikování webové aplikace. Po provedení změn v kódu aktualizuji lokální repozitář pomocí *git commit*. V tomto kroku se vyvolá nástroj *Husky*, který před samotným *git commit* spustí nástroj *ESLint*, který zkontroluje, zda se v kódu nevyskytují syntaktické chyby. Pokud se narazí na chybu, *git commit* je odmítnut. Poté pomocí *git push* aktualizuji vzdálený repozitář, používám *GitHub*[18]. Na ten je napojen nástroj *Travis CI*[19], který jakmile zjistí, že došlo ke změně v repozitáři, spustí automatické testy. Pokud testy selžou, informuje mě o tom emailem. Aplikaci mám hostovanou na platformě *Heroku*[20]. Poté, co úspěšně proběhne CI, Heroku spustí skript pro sestavení aplikace, který vytvoří novou verzi aplikace a aktualizuje soubory na webovém serveru, čímž se nová verze dostává k uživatelům.

V případě serverové části je to jednodušší, ta není tolik složitá. Pro vzdálený repozitář používám *Bitbucket*[21] a pro hosting platformu *Microsoft Azure*[22]. Opět mám vytvořené napojení na repozitář, tedy při změně v repozitáři si MS Azure automaticky aktualizuje soubory a restartuje server.

[18] http://github.com
[19] https://travis-ci.org
[20] https://www.heroku.com/
[21] https://bitbucket.org/
[22] https://azure.microsoft.com/cs-cz/



Obrázek 11: Lekce typu fráze s obrázkovou nápovědou

U mobilní aplikace automatické publikování nemám, aplikaci na Google Play aktualizuji ručně. Nicméně pro vývoj jsem použil nástroj *Expo*[23], který umožňuje průběžné publikování aplikace na jejich vlastní servery, které slouží jako zdroj pro *over-the-air* aktualizace. Čili takto je možné vydat novou verzi, aniž bych aktualizoval záznam na Google Play.

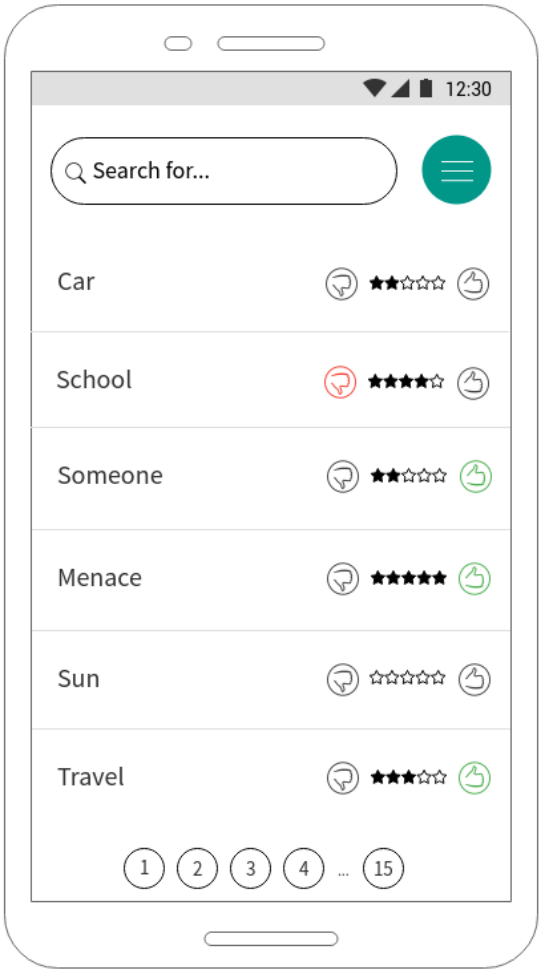
5.3 Podpůrné nástroje a knihovny

5.3.1 Balíčky

Pro JavaScriptové projekty je typická silná závislost na externích knihovnách. Pro webovou aplikaci jsem jich použil více než 20. Správce balíčků pro JavaScript je *npm*, obecně je to nejrozsáhlejší systém open source knihoven [26].

Je tedy pravděpodobné, že problém, na který narazíme již řešil někdo před námi a svoje řešení publikoval na npm. Použitím takového řešení ve vlastní aplikaci si ušetříme spoustu práce, navíc ty populární balíčky jsou také velice dobře odladěné. Ovšem má to i své mínusy, občas můžeme narazit na nekompatibilitu a pak především u déle trvajícího projektu se dostáváme do smyčky aktualizací, kdy opětovně zapracováváme nové aktualizace balíčků a nijak se neposunujeme ve vývoji. Nové verze balíčků vychází poměrně často a občas se jedná o dramatické změny. Potom při aktualizaci může být nutné přepracovat i větší část aplikace.

[23] https://expo.io/tools



Obrázek 12: Wireframes - úvodní strana

5.3.2 Sestavení kódu

Sestavení mobilní aplikace jsem řešil pomocí nástroje Expo. Nutnost sestavení je zřejmá – je třeba zpracovat kód v JavaScriptu do instalačního souboru. V případě webové aplikace jsem použil *webpack* [27]. V tomto případě se ale nabízí otázka, proč vůbec sestavovat JavaScript, když aplikace běží ve webovém prohlížeči, který samozřejmě je schopen JavaScript zpracovat.

Základní problém je v použití knihoven, protože ty si uvnitř obyčejně načítají další knihovny atd. Výsledný objem aplikace by ale takto byl příliš velký. Například knihovny standardně obsahují svoje vlastní testy, ale ty do naší aplikace nepotřebujeme začleňovat. Webpack si vytváří graf závislostí mezi moduly, čímž je schopen zpracovat aplikaci tak, aby obsahovala jen ty nezbytně nutné moduly.

Další přední problém je v tom, že jak se JavaScript za poslední roky hodně vyvinul, vývojáři se naučili používat nové prvky jazyka, protože jim usnadňují práci. Ovšem ne všechny webové prohlížeče znají tyto nové verze jazyka. Za prvé

Obrázek 13: Wireframes - strana slovíčka

to nějakou dobu trvá, než vývojáři zapracují nový standard jazyka do prohlížeče[24] a za druhé je třeba podporovat i ty prohlížeče, které jsou i několik let staré a stále používané. Tento problém řeší nástroj *babel*, který převede prvky z nového standardu jazyka do staršího.

5.3.3 Ladění

Webovou aplikaci lze pohodlně ladit v prostředí webového prohlížeče. Do prohlížeče Google Chrome, ale i do jiných existuje rozšíření *React Developer Tools*[25] a *Redux DevTools*[26], díky nimž lze aplikace velice dobře debugovat.

S mobilní aplikací je to trochu složitější. Expo umožňuje spustit aplikaci v simulátoru nebo přímo v mobilním telefonu. Simulátor si můžu spustit přes

[24] https://kangax.github.io/compat-table/es6/

[25] https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi

[26] https://chrome.google.com/webstore/detail/redux-devtools/lmhkpmbekcpmknklioeibfkpmmfibljd

Obrázek 14: Kroky při publikování webové aplikace

Android Studio, nicméně příliš se mi tato varianta nelíbila, protože odezva aplikace byla velice špatná. Ani zvýšení výkonu simulovaného zařízení nepřineslo větší zlepšení, přestože výkon počítače by pro to měl být dostatečný.

Spuštění aplikace na mobilním telefonu je prosté: do mobilního telefonu nainstalujeme aplikaci Expo z Google Play. Dále zajistíme, aby počítač a telefon byly připojeny ve stejné síti. V mobilním telefonu do aplikace Expo načteme QR kód, který vygeneroval nástroj Expo v počítači, tím dojde ke spuštění aplikace. Takto spuštěná aplikace byla podstatně rychlejší než spuštění přes simulátor, proto jsem tuto variantu používal. Co se týče debugování Expo pro to využívá vývojářské nástroje v prohlížeči, ovšem je to podstatně méně pohodlné než debugování webové aplikace. Z tohoto důvodu jsem se rozhodl vytvořit nejprve webovou aplikaci a poté tu mobilní.

5.4 Programová část

5.4.1 Správa dat

Pro práci s daty jsem použil Redux, který implementuje zmiňovanou architekturu Flux. Místo, kde Redux ukládá data se nazývá *Store* a je to tedy vnitřní stav aplikace. Data ve Store lze rozdělit na perzistentní a neperzistentní. Neperzistentní informace je například číslo aktivní stránky ve stránkovaném seznamu, jinak většina dat je perzistentních.

Před rozhodnutím, kam data ukládat, je nutné zjistit kolik paměti asi bude potřeba. V aplikaci ukládám pouze uživatelská data, čili například ID oblíbených slovíček, hodnocení slovíček, vytvořené lekce, nejlepší skóre atd. Samotné definice slovíček neukládám, podmínky použití online služby pro slovník to ani neumožňují. Předpokládám, že průměrný uživatel bude mít v aplikaci stovky slovíček a jednotky až desítky seznamů a lekcí. Abych mohl odhadnout, kolik je to přibližně dat, vytvořil jsem si jednoduchý generátor, který náhodně generuje tato uživatelská data v zadaném objemu. Při objemu 5000 slovíček, 500 lekcí, 500 seznamů a 5 frází na slovíčko, tedy podstatně více než kolik reálně odhaduji, dostávám přibližně 5MB dat.

U mobilní aplikace tato data ukládám do vnitřní paměti zařízení. U webové aplikace používám *Local Storage*, tedy úložiště v rámci webového prohlížeče. Velikost tohoto úložiště je hodně limitovaná, záleží na prohlížeči, ale mělo by být k dispozici minimálně 5MB [28].

V mobilní aplikaci jsem implementoval také možnost stažení slovníku do zařízení a mít tak možnost slovník používat i offline. Jedná se o slovník WordNet, nicméně preferovaná varianta je použití online služby Oxford Dictionaries, která

oproti WordNet obsahuje i výslovnost, audio a další informace. Velikost slovníku WordNet je přijatelných 35MB.

Pro externí úložiště jsem použil Google Drive. Díky němu lze sdílet data mezi mobilní a webovou aplikací. Jelikož se jedná o osobní aplikaci, nepředpokládám použití webové a mobilní aplikace zároveň – synchronizace v reálném čase by byla netriviální záležitost.

5.4.2 Komponenty

Pro strukturování komponent je vhodné použít dělení na *obalové* a *prezentační*, případně se označují jako *chytré* a *hloupé*. Prezentační (hloupé) komponenty jsou určeny pro vizuální zobrazení, nemají vnitřní stav a nejsou napojeny na Store. Může se jednat například o komponentu *Tlačítko*. Obalové (chytré) komponenty jsou určeny pro ztvárnění logiky, na vizuálním zobrazení by se měly podílet minimálně, mohou mít vnitřní stav a mohou být napojeny na Store. Obalová komponenta může být třeba *Vyhledávací formulář*, která bude obsahovat *Textové políčko* a *Tlačítko*. Obsah textového políčka je její vnitřní stav a po odeslání formuláře pošle odpovídající akci do Store.

Důležité jsou také komponenty vyšších řádů (High Order Components, případně jen HOC), pomocí kterých je možné sdílet funkcionalitu napříč různými komponentami. HOC se předává komponenta jako parametr, tato komponenta se rozšíří o novou funkcionalitu a vrátí se jako výstup HOC.

Komponenty se skládají stejně jako HTML elementy, tvoří tedy strom. Při vývoji je možné postupovat od té nejvyšší úrovně (označováno jako *top-down*) nebo od té nejnižší (*bottom-up*). Obyčejně je pro jednodušší aplikace výhodnější top-down, pro větší bottom-up. Použil jsem top-down.

5.4.3 Externí služby

V aplikaci jsem použil tři služby třetích stran: *Oxford Dictionaries* pro slovník, *Microsoft Azure* pro překlad a *Google Drive* pro externí úložiště. Na straně serveru jsem použil *mLab*[27] pro hostování *MongoDB* databáze. Všechny tyto služby využívám zdarma. Oxford Dictionaries, Microsoft Azure a mLab mají pro bezplatnou verzi jistá omezení. Pokud by to znamenalo problém, lze to samozřejmě jednoduše řešit přechodem na placenou verzi.

Během práce s Oxford Dictionaries jsem zjistil, že jejich verze zdarma je až příliš omezená. Jedno z omezeních bylo maximálně 60 dotazů za minutu, což jsem byl i jako jednotlivec schopen vyčerpat. Nejvíce dotazů generoval formulář s funkcí *autocomplete*, kdy se s každým znakem odesílal požadavek na vyhledání. Abych se tomu vyhnul zvedl jsem interval čekání mezi stiskem klávesy až na 2s. Tedy až 2s poté, co uživatel zadal poslední znak, se teprve odešle dotaz. Přesto se mi ale limit této služby zdál příliš striktní, velice rychle by mohlo dojít k jeho vyčerpání. Proto jsem si na serveru, který slouží jako prostředník pro tyto

[27] https://mlab.com

dotazy, napojil i databázi WordNet. Tedy pokud dotaz na Oxford Dictionaries selže, vyhledává se ve WordNet.

Co se týče Google Drive je třeba zažádat o udělení přístupu k datům. Podle bezpečnosti jsou k dispozici různé úrovně přístupu. U webové aplikace využívám tu nejnižší, kdy je pro aplikaci vytvořen samostatný prostor mimo uživatelská data. U mobilní aplikace už využívám vyšší úroveň, aby bylo možné přímo z aplikace provést export na Google Drive. Nicméně stále se jedná o velice bezpečný přístup, aplikace sice může číst a zapisovat do uživatelského prostoru, ale zapisovat může pouze do kořenové složky a číst může pouze soubory, které si sama vytvořila. K souborům uživatele tedy nemám přístup.

Díky tomu, že se mi podařilo najít vhodné služby, které je možné využívat bezplatně, aplikaci můžu provozovat zdarma a bez reklam.

5.5 Server

Pro potřeby aplikací jsem vytvořil také serverovou část. Je to jednoduché REST API v Node.js, které zajišťuje tyto funkce:

- prostředník pro komunikaci s Oxford Dictionaries a MS Azure
- slovník WordNet
- logování chyb
- poskytování informací o připravených sadách slovíček

U aplikací tohoto typu, kde probíhá velké množství malých dotazů je výhodné použití *Node.js* [29], který díky neblokujícím operacím dokáže obsloužit velké množství požadavků.

K rozhodnutí realizovat serverovou část jsem dospěl poté, co jsem zjistil, že z webového prohlížeče nelze odesílat dotazy přímo na API Oxford Dictionaries. Je to z toho důvodu, že jejich server nepodporuje *CORS*[28], což znamená, že webový prohlížeč z důvodu bezpečnosti zakáže dotaz směřující na jinou doménu. Byl jsem tedy nucen řešit to použitím vlastního serveru jako prostředníka, kde mám nastavenou podporu CORS.

Výhodné je také logování chyb, jelikož samozřejmě chci být informován o kritických chybách, které způsobí pád aplikace, aniž bych musel čekat až mi to uživatelé nahlásí.

5.6 Mobilní aplikace

5.6.1 Styled Components

V React Native nelze používat klasické HTML a CSS, proto jsem použil nástroj *Styled Components*, který usnadňuje zápis stylů v JavaScriptu.

[28] http://tiny.cc/vxh78y

Na to, že se pro celý layout používá flexbox, jsem si zvykl rychle. Co mi ale dělalo problémy byly samotné styly. Delší dobu se věnuji vývoji v oblasti webů, CSS znám velice dobře a proto jsem byl nemile zaskočen faktem, že řada věcí z CSS v React Native nefunguje. V první řadě jsem zvyklý používat pseudoelementy *::after* a *::before*, což v React Native nelze. A dále některá CSS pravidla nefungují. Například vytvoření ohraničení by v CSS vypadalo takto:

```
1   border: 1px solid #000;
```

Zdrojový kód 6: CSS

Problém je v tom, že React Native nepodporuje jiný typ ohraničení než *solid*, proto se typ ohraničení ani nepíše a uvedený zápis tedy není syntakticky správně. Toto pravidlo je třeba rozložit na dvě pravidla takto:

```
1   border-width: 1px;
2   border-color: #000;
```

Zdrojový kód 7: CSS

Nefunguje ani box-shadow, animace, fixní pozice... čili pro vývojáře, který přechází z oblasti webů, nemusí být snadné přeorientovat se na jiná řešení, než na která byl zvyklý.

5.6.2 Podpora zařízení

Vyvíjel jsem pouze pro Android, i když zajištění podpory pro iOS by mělo být relativně jednoduché – téměř celý kód by měl být platformě nezávislý, pouze by bylo nutné ošetřit pár případů, kde to neplatí. V aplikaci můžeme zjistit o jaký OS se jedná, čili řešení by mělo být velice přímočaré. Podle detailů aplikace z dílny Airbnb [30] měli pouze 0.2% kódu specifického pro určitou platformu.

Pomocí flexboxu lze nastavit proměnlivou šířku sloupců, čili layout by se měl přizpůsobit různým velikostem obrazovky. Simuloval jsem aplikaci i na tabletu, jeví se v pořádku. Ovšem nějaké robustní řešení responzivity, jaké je k dispozici v CSS, chybí.

Přestože se tedy layout přizpůsobuje velikosti obrazovky, přepínání mezi *portrait* a *landscape* módem způsobovalo značné problémy. Nakonec jsem se rozhodl landscape mód zakázat.

V simulátoru jsem otestoval několik různých zařízení a verzí OS, aplikace by měla fungovat na všech mobilních telefonech s verzí Android 5.0 a vyšší.

5.6.3 Publikování

Proces publikování aplikace na Google Play proběhl dle očekávání. Pro vývojáře je připraveno příjemné administrační rozhraní. Někomu by případně mohl způsobit komplikace fakt, že některé věci je nutné řešit v angličtině. Pouze jsem byl trochu překvapen, že je požadován menší registrační poplatek pro účet vývojáře.

Přibližně po týdnu od publikování mi bylo oznámeno, že aplikace byla zakázána, protože porušuje podmínky použití služby Google Play. Jádro problému bylo v tom, že nástroj Expo, pomocí kterého jsem aplikaci vyvíjel, obsahuje i třeba napojení na sociální sítě, které si na straně uživatele ukládají sledovací kódy. Přestože jsem nic takového ve své aplikaci nepoužil, Expo tyto kódy vloží do výsledného balíčku. Při automatické kontrole Google Play narazil na tyto sledovací kódy, které se sice nikdy nevykonají, přesto vyžaduje, aby aplikace měla zpracované zásady ochrany osobních údajů. Na základě veřejně dostupných šablon jsem zpracoval tyto zásady, umístil online a požádal o přezkoumání. Během dvou dnů byla aplikace opět povolena.

Stejné záležitosti jsem řešil i při použití Google Drive API, přes které je možné přistupovat k soukromým souborům uživatele. Podával jsem tam žádost ohledně toho, k jakým uživatelským datům a z jakého důvodu vyžaduji přístup. Je únavné tyto věci řešit, na druhou stranu je velice pozitivní, že se vyvíjí nemalé úsilí k ochraně osobních údajů.

5.7 Webová aplikace

5.7.1 HTML a CSS

Pro zápis stylů jsem použil preprocesor SASS a metodiku ITCSS (pro organizaci selektorů používá metodiku BEM). Layout jsem sestavil pomocí vlastního gridu, který je postavený na flexboxu.

Aplikace je plně responzivní, v rozměru pro mobilní telefony vypadá téměř totožně jako mobilní aplikace. Responzivitu jsem zpracoval přístupem *mobile-first*, tedy vycházel jsem z rozměru pro mobilní telefony a následně jsem zobrazení upravoval pro větší obrazovky.

Pro responzivitu jsem použil tyto breakpointy:

```
1   xs: 340,
2   sm: 480,
3   md: 767,
4   bg-page-img: 960,
5   lg: 1024,
6   xl: 1200,
7   xxl: 1600,
8   xxxl: 1900
```

Zdrojový kód 8: SASS

5.7.2 Podpora webových prohlížečů

Podporou starších prohlížečů jsem se příliš nezabýval, cílil jsem na ty moderní. Nicméně použil jsem zavedené techniky, nic příliš převratného a také *babel* je nastaven pro podporu IE11+, tedy i takto starý prohlížeč by měl být podporován.

5.7.3 Publikování

SPA nemá prakticky žádné požadavky na hosting, stačí pouze hostování statických souborů. V případě, že se aplikace skládá z více stránek, je třeba vyřešit přesměrování. SPA se totiž fyzicky skládá pouze z jedné stránky, ovšem při navigaci dochází ke změně historie v prohlížeči, což vypadá, jako kdyby se uživatel skutečně přesunul na novou stránku. Pokud by ale tuto stránku v prohlížeči načetl znovu, dostal by se na neexistující adresu. Tedy je potřeba všechny tyto dotazy přesměrovat na vstupní stranu aplikace.

Rozumné je použít přímo hosting, který je orientovaný na SPA. Vybral jsem si platformu Heroku. Pro hosting je možné zvolit verzi zdarma a navíc Heroku nabízí spoustu dalších užitečných funkcí jako třeba propojení s GitHub repozitářem. Nevýhoda je, že hosting zdarma po určité době nečinnosti upadá do módu spánku. Následně trvá několik sekund než se obnoví, tedy úvodní načtení aplikace potom může trvat dost dlouho.

5.8 Testování

Pro testování je na výběr řada frameworků: *Mocha*, *Chai*, *Jest*, *Karma*... Vybral jsem *Jest*. Tento framework je dobrý pro unit a integrační testy. Umožňuje také zajímavou možnost testování a to pomocí *Snapshot*. Snapshot je řekněme fotografie komponenty. Princip tohoto testování spočívá v tom, že poté co komponentu otestujeme manuálně, pořídíme její Snapshot. Následně se při automatickém testování vytvoří nový Snapshot a porovná se s tím původním. Pokud se narazí na rozdíl, test je označen jako chybný. Podle nalezených rozdílů programátor rozhodne, jestli se skutečně jedná o chybu. Pokud se jedná o zamýšlené změny, dá pokyn pro aktualizaci Snapshotu.

Pro end-to-end testy jsem použil framework *Cypress*, který testuje aplikaci z pohledu uživatele. Pro tyto účely používá *headless browser*, což je zjednodušený webový prohlížeč bez grafického uživatelského rozhraní. S Cypress se mi pracovalo velice dobře. Pro akce používá časovače tak, aby se počkalo na dokončení asynchronních událostí, které se v SPA hojně využívají. Při simulování akce uživatele, například kliknutí na tlačítko, Cypress kontroluje také proveditelnost této akce. Akce selže v případě, že je tlačítko mimo obrazovku, zakryté jiným prvkem apod.

Původně jsem vyvíjel podle principu *Test Driven Development*, ale postupně jsem od toho upustil. Během vývoje jsem totiž ještě návrh aplikace hodně měnil. Často se mi stávalo, že poté co jsem napsal testy a dostal se k vývoji komponent, rozhodl jsem se pro jiné řešení a tak jsem ty připravené testy bez užitku

zahazoval.

5.9 Další vývoj

Z vývoje podobných aplikací se často stává dlouhodobý projekt. Pokud má aplikace potenciál a je o ni zájem, můžeme ji postupně rozšiřovat a vylepšovat a získávat tak další uživatele. Především uvažuji o rozšíření funkcionality, případně by se také dalo vylepšit UI, přidat více animací, vytvořit profesionální design atd.

Co se týče rozšíření funkcionality, v úvahu připadá řada možností. Mohl bych využít další informace, které poskytuje služba Oxford Dictionaries, což je například vyhledávání synonym. Myslím, že velice zajímavé by také bylo napojení aplikace na sociální sítě a nějaká možnost sdílení pokroku ve studiu, čímž by se uživatelé mohli navzájem k učení motivovat. Dále bych v aplikaci uvítal i nějakou odlehčenou formu výuky, třeba v podobě kvízů, doplňovaček nebo přímo nějakých her.

Pokud je uživatelův slovník prázdný, nabízím mu k importu několik připravených seznamů se slovíčky. Těchto připravených seznamů by mohlo být více a včetně třeba i frází a lekcí, aby si uživatel nemusel vše vytvářet sám. Ovšem k vytvoření takových materiálů by už byly potřeba profesionální znalosti angličtiny.

Pohlížím na aplikaci pouze z pohledu vývojáře, důležité je také získat zpětnou vazbu od uživatelů. Základ jsou uživatelské recenze, případně bych mohl udělat nějaký průzkum mezi vzorkem uživatelů a podobně. Zajímavá technika pro zkoumání chování uživatelů je A/B testování. Testování probíhá tak, že nějaký prvek nebo funkcionalita je poskytnuta jen vzorku uživatelů a u nich se sleduje, jak na to budou reagovat. Pokud se dojde k závěru, že by tato změna byla prospěšná, zapracuje se do aplikace.

Jelikož je React Native multiplatformní, nabízí se vydat aplikaci také pro iOS. Dále bych se mohl zabývat optimalizacemi, propagovat aplikaci na sociálních sítích, nebo použít nějakou formu reklamy. Pokud by se aplikace měla rozrůst, lze také uvažovat o přidání reklam nebo nějakého placeného obsahu, čímž by se mohl financovat další vývoj.

Závěr

Úspěšně jsem vytvořil webovou a mobilní aplikaci pro studium anglických slovíček. Forma výuky je připravena tak, aby byla pro uživatele efektivní. Obě aplikace jsou veřejně dostupné, mobilní aplikace byla publikována na Google Play. Pro potřeby aplikací jsem vytvořil i samostatnou serverovou část s databází. Aplikace také využívají několik volně dostupných služeb třetích stran.

Prozkoumal jsem možná řešení, především jsem se zaměřil na moderní technologie. Jako programovací jazyk jsem zvolil JavaScript. U moderního JavaScriptu jsem se setkal s velice rozsáhlým systémem technologií, které se rychle vyvíjí. Aplikace jsem vytvořil pomocí technologií React a React Native. Zdokumentoval jsem jejich základní koncepty, které je důležité si osvojit pro správný návrh aplikace.

Vývoj byl pohodlný, u obou aplikací jsem zpracoval požadovanou funkčnost. S webovou aplikací vytvořenou pomocí SPA a React jsem byl velice spokojený. S vývojem serveru v Node.js také, zde bych pouze případně zvažoval použití moderního GraphQL místo REST API. Dle mého názoru je React Native dobrá volba pro vývoj mobilních aplikací, nicméně uvažuji, že bych příště vyzkoušel jinou technologii.

Conclusions

I successfully created web and mobile applications for learning English words. The learning process is made to be effective for users. Both applications are public, the mobile application was published to Google Play. For the applications' needs I created an independent server with a database. Applications also use several public third-party services.

I examined development options focusing especially on modern technologies. For programming language I chose JavaScript. With modern JavaScript, I came across a huge system of technologies, that develop rapidly. I developed the applications using React and React Native. I described their basic concepts, which are important for creating a solid application.

Working with these technologies was a pleasant experience, both applications incorporate the required functionality. I was quite satisfied with using SPA and React for the web application. Developing the server in Node.js was pleasant as well, perhaps I would just consider using modern GraphQL instead of REST API. In my opinion React Native is a good choice for mobile application development however, I'm thinking next time I would probably try some other technology.

A Uživatelská příručka

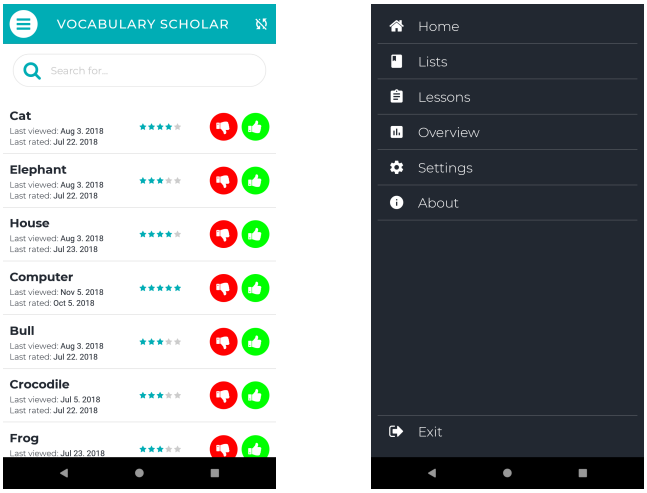
Náhledy pochází z mobilní aplikace, ale webová aplikace je prakticky totožná.

A.1 Použití

- Aplikaci můžete použít jako slovník.
- Pro učení nových slovíček si je můžete uložit - přidat si je do oblíbených, takže se k nimi budete moci kdykoli vrátit.
- Aby bylo učení pohodlnější, můžete si ke svým oblíbeným slovíčkům uložit různá data. Hodnocení bude pravděpodobně to nejdůležitější.
- Nyní byste měli být schopni efektivně procvičovat slovíčka. Aby bylo procvičování ještě lepší, můžete si pustit lekci (prozatím pouze anonymní, pro normální lekci bude potřeba nejprve vytvořit nějaké seznamy slovíček).
- Jak budete přidávat další a další slovíčka, slovník může být trochu nepřehledný - pro lepší organizaci můžete slovíčka seskupovat do seznamů.
- Jakmile máte nastaveny slovíčka a seznamy, můžete si spustit různé lekce a sledovat Vaše skóre.

A.2 Menu

- Otevřete menu kliknutím na tlačítko vlevo nahoře.
- (mobilní aplikace) Zavřete menu pomocí gesta *tažení vlevo* nebo ťuknutím na tlačítko *zpátky* na Vašem zařízení.
- (webová aplikace) Zavřete menu kliknutím mimo nebo na tlačítko menu.



(a) Menu zavřené (b) Menu otevřené

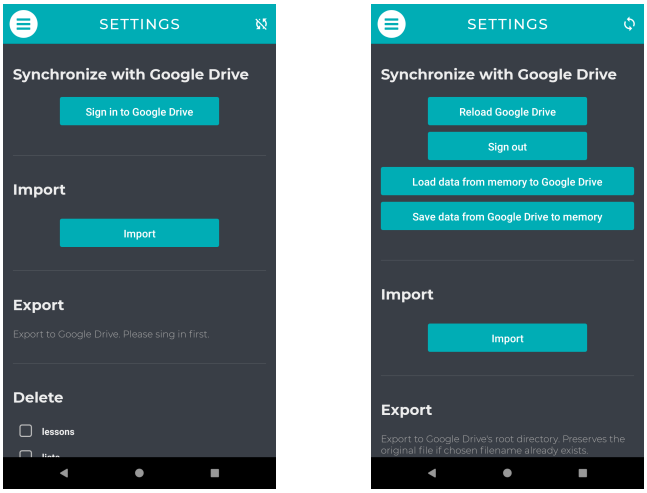
Obrázek 15: Menu

A.3 Ukládání uživatelských dat

Uživatelská data mohou být ukládána na dvou místech - vnitřní paměť a/nebo Google Drive.

Mobilní aplikace	**Webová aplikace**
- vnitřní paměť zařízení (výchozí) - Google Drive	- *Local Storage* ve webovém prohlížeči (výchozí) - Google Drive

- Pro ukládání na Google Drive se na straně *Settings* přihlaste na Google Drive.
- Až se přihlásíte, při příštím otevření aplikace se aplikace automaticky pokusí opětovně přihlásit.
- V aplikaci můžete mít takto uloženy dvě různé verze uživatelských dat - jednu v Google Drive a druhou uloženou ve Vašem zařízení/webovém prohlížeči. Když se přihlásíte můžete uložit data z Google Drive do zařízení/webového prohlížeče. A také naopak ze zařízení/webového prohlížeče do Google Drive.
- Aplikace ukládá velice malý objem dat, méně než 1MB pro přibližně 5000 slov.

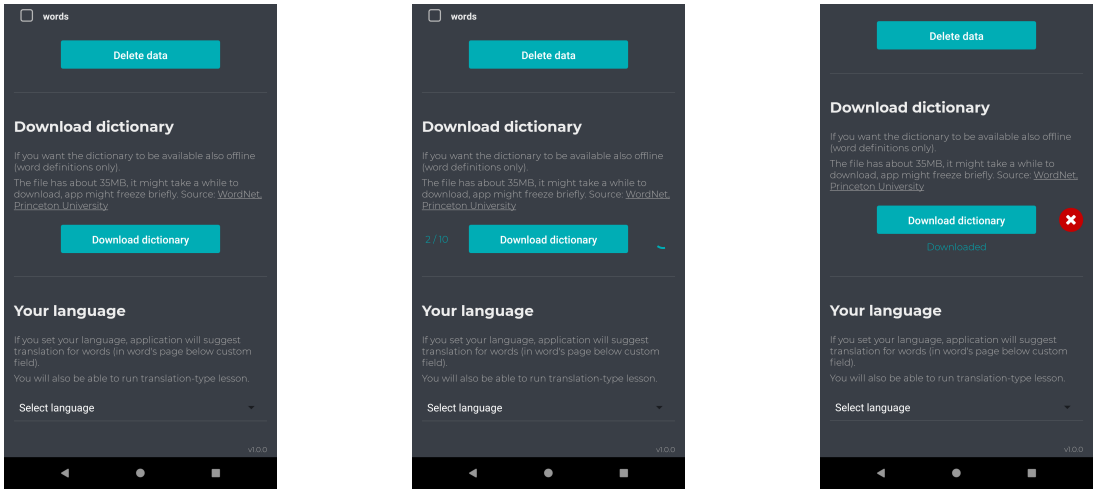


(a) Nepřihlášen (b) Přihlášen

Obrázek 16: Přihlášení na Google Drive

A.4 Slovník

- Aplikace používá online službu pro slovník. Hlavním zdrojem dat je Oxford Dictionaries, pokud není k dispozici (pravděpodobně protože byl vyčerpán limit pro použití zdarma), použije se Wordnet (Princeton University).
- (pouze mobilní aplikace) Bez připojení k internetu služba pro slovník nebude fungovat, takže pro tento případ si můžete stáhnout Wodnet do svého zařízení a použít jej i offline.
- (pouze mobilní aplikace) Na straně *Settings* je sekce *Download dictionary*, stáhněte slovník tam. Slovník má přibližně 35MB a je rozdělen do 10 částí. Bude to trvat nějaký čas než se stáhne. Během ukládání slov může vaše zařízení na chvíli zamrzat - ve slovníku je téměř 150 tisíc záznamů. Zrušení procesu stahování zanechá slovník neúplný.



(a) Stažení slovníku (b) Slovník se stahuje (c) Slovník stažen

Obrázek 17: Stažení slovníku

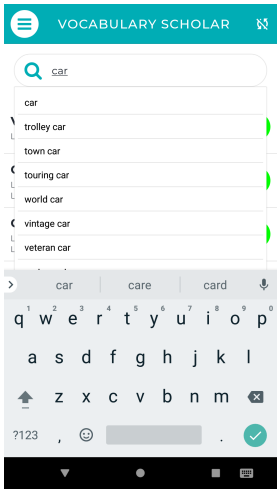
A.5 Přidat slovíčko

Ručně

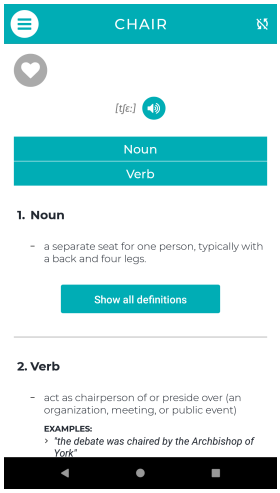
- Nejprve pomocí vyhledávacího pole na straně *Home* najděte slovíčko, které hledáte.
- Na straně *Word* nahoře se nachází tlačítko *přidat do oblíbených* (ikonka srdce), klikněte na něj abyste přidali slovíčko do slovníku, klikněte znovu pro odstranění.
- Jakmile je slovíčko nastaveno jako oblíbené, můžete k němu přidávat fráze, hodnocení, umístit jej do seznamů apod.

Import

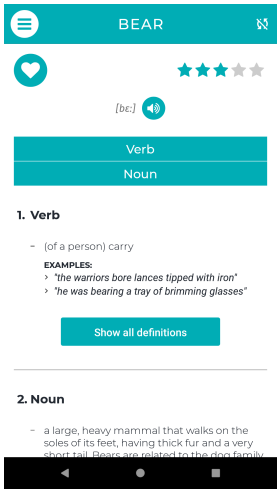
- Pokud je Váš slovník prázdný, běžte na stranu *Home*, kde se objeví ukázkové seznamy slovíček, které si můžete importovat.
- Na straně *Settings* si můžete importovat vlastní soubor.



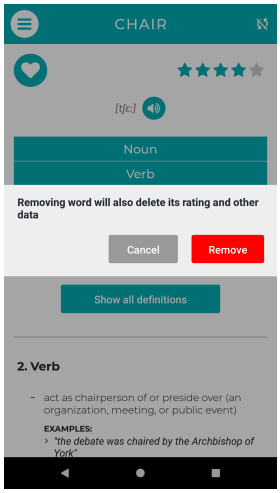
(a) Vyhledávání



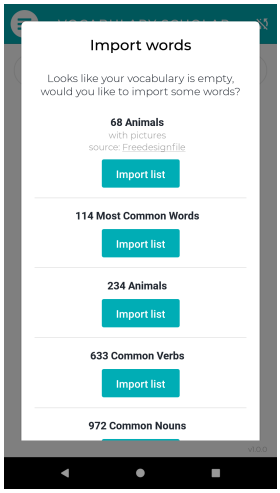
(b) Slovíčko není v oblíbených



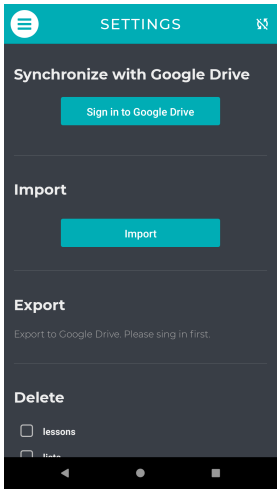
(c) Slovíčko přidáno do oblíbených



(a) Slovíčko odstranit z oblíbených



(b) Ukázkové seznamy pro import když je slovník prázdný

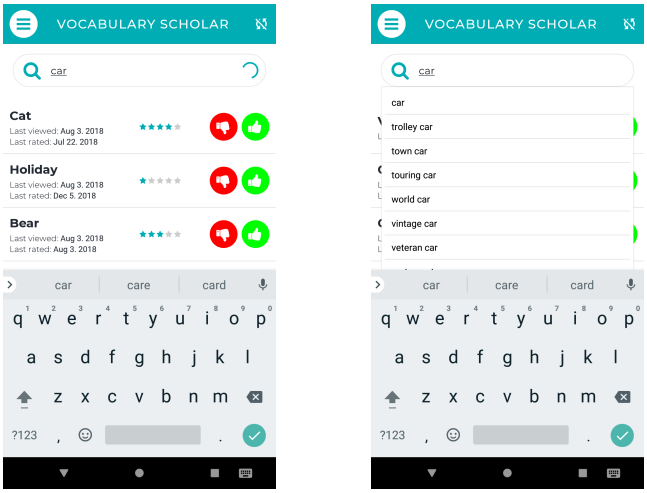


(c) Importovat ze souboru

Obrázek 19: Přidání slovíčka do oblíbených

A.6 Vyhledávání

- Na straně *Home* napište slovíčko, které hledáte do vyhledávacího pole. Pod políčkem se objeví průběžné návrhy, aby bylo Vaše hledání pohodlnější. Poté potvrďte hledání nebo klikněte na jeden z návrhů, dostanete se na stranu *Word*.
- Vyhledávací formulář je také na straně *List*, ale tento vyhledává pouze v rámci slovíček v aktuálním seznamu.



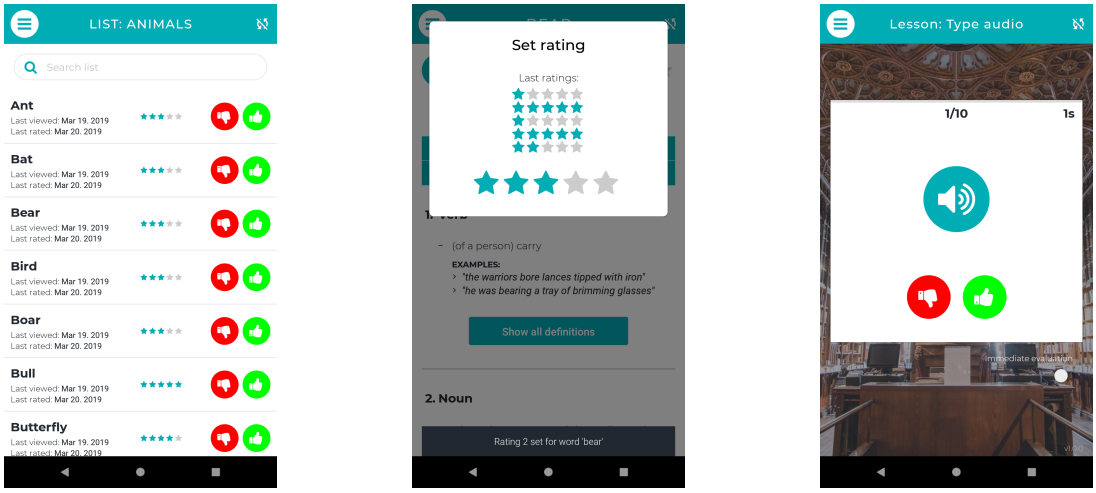
(a) Vyhledávací formulář

(b) Návrhy vyhledávání

Obrázek 20: Vyhledávání slovíčka

A.7 Nastavit hodnocení slovíčka

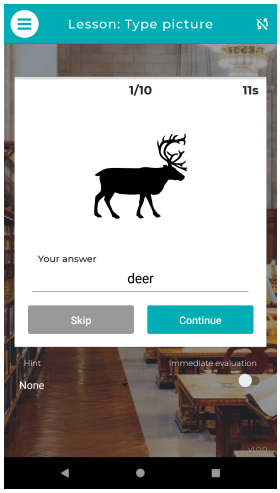
- Vy výpisu slovíček klikněte na hodnotící tlačítka, která jsou vpravo, to zelené (správně) tlačítko nastaví hodnocení 5 hvězd a to červené (špatně) 1 hvězdu.
- Nebo na straně *Word* klikněte na hvězdy, objeví se vyskakovací okno, klikněte na hvězdu, kterou chcete nastavit.
- Nebo pokud je spuštěna lekce, podle toho jak odpovíte, je slovíčkům nastaveno hodnocení 1 nebo 5.



(a) Nastavení hodnocení v seznamu slovíček

(b) Slovíčko nastavit hodnocení

(c) Hodnocení během spuštěné lekce

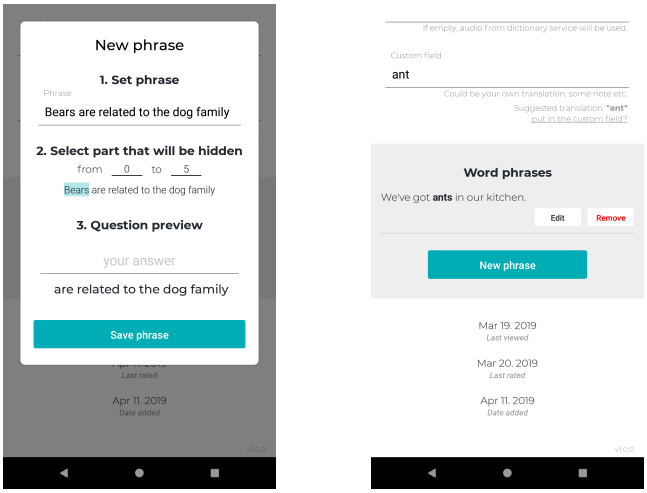


(a) Hodnocení během spuštěné lekce

Obrázek 22: Nastavení hodnocení slovíčka

A.8 Přidat frázi

- Fráze nemůže být samostatná, musí být spojena se slovíčkem.
- Na straně *Word* pod *Lesson settings* klikněte na *New phrase*, objeví se vyskakovací okno.
- Nejprve napište frázi, poté vyberte část fráze, která bude během lekce skryta. Cílem bude poskládat frázi zpátky.
- Frázi můžete vytvořit velice jednoduše, když si vybraný text zkopírujete. Potom bude políčko s frázi předvyplněno tímto textem automaticky a pokud se aktuální slovíčko nachází ve frázi, tato část fráze bude automaticky označena jako ta skrytá část. Vám pak bude stačit pouze kliknout na tlačítko *Save phrase*.

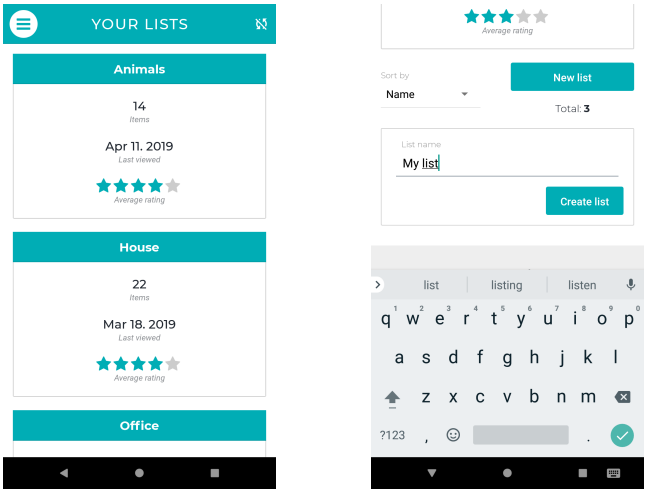


(a) Přidat frázi (b) Slovíčko fráze

Obrázek 23: Přidání fráze ke slovíčku

A.9 Vytvořit seznam slovíček

- Na straně *Lists* klikněte na tlačítko *New list*, objeví se formulář.
- Napište název seznamu a potvrďte.



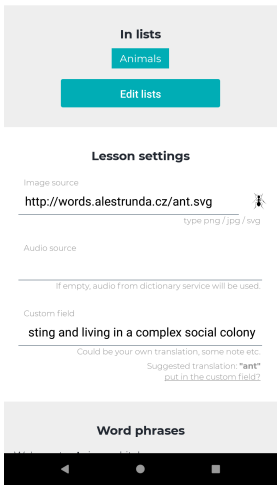
(a) Seznamy (b) Vytvoření seznamu

Obrázek 24: Seznam slovíček

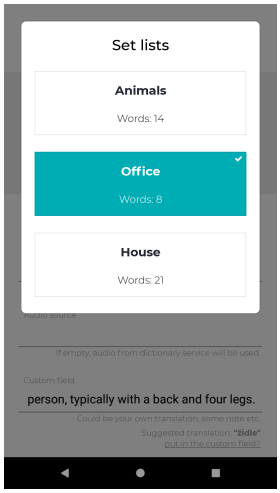
A.10 Nastavit slovíčka v seznamu

- Můžete je nastavit ze strany *Word* nebo ze strany *List*.

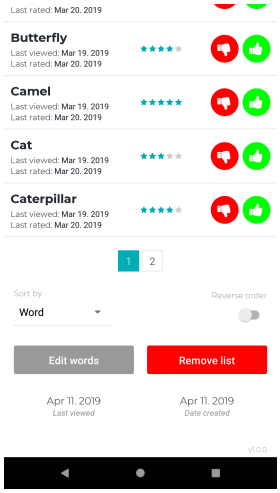
- Na straně *Word* klikněte na tlačítko *Edit lists*, objeví se vyskakovací okno, zatrhněte/zrušte seznamy jak potřebujete.
- Nebo na straně *List* klikněte na tlačítko *Edit words*, dostanete se do módu nastavení - zatrhněte/zrušte jak potřebujete, poté co jste skončili klikněte na *Done editing*.



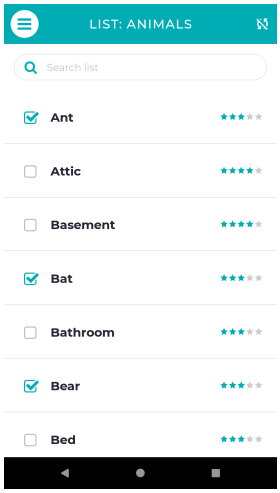
(a) Seznamy ve kterých je slovíčko



(b) Vyskakovací okno nastavení seznamů



(c) Seznam nastavení slovíček



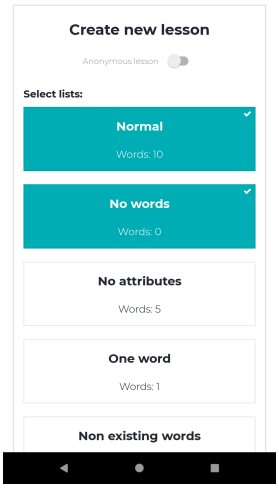
(a) Seznam mód nastavení

Obrázek 26: Nastavení slovíček v seznamu

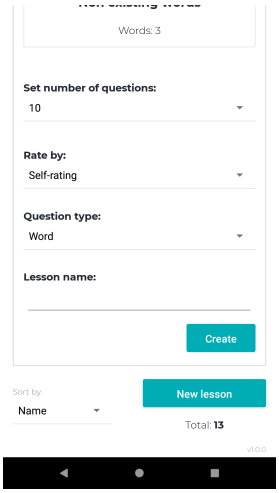
A.11 Vytvořit lekci

- Na straně *Lessons* klikněte na tlačítko *New lesson*, otevře se formulář pro novou lekci.

- Nejprve je třeba se rozhodnout, zda chcete normální nebo anonymní lekci.
  - Normální lekce je uložena, takže si ji můžete spustit znovu a sledovat svoje skóre. Můžete si vybrat pouze z vlastních seznamů slovíček, nikoli z výchozích seznamů, protože ty se příliš mění, čili sledování skóre by nebylo k užitku.
  - Anonymní lekce není uložena. Můžete si vybrat pouze z výchozích seznamů slovíček (seznamy jsou vytvořeny automaticky na základě aktuálních dat ve slovníku, například seznam obsahující pouze pětihvězdičková slovíčka).
- Vyberte seznamy jejichž slovíčka chcete mít v lekci.
- Poté co jsou seznamy vybrány, nastavte kolik otázek chcete v lekci mít.
- Vyhodnocení lekce můžete nastavit na vlastní vyhodnocování (objeví se hodnotící tlačítka, sami si rozhodnete, zda znáte odpověď nebo ne), nebo na vyhodnocování podle odpovědi, kterou napíšete do pole.
- Vyberte typ otázek.
- Nastavte název lekce (nikoli pro anonymní) a vytvořte lekci.



(a) Formulář vytvoření lekce



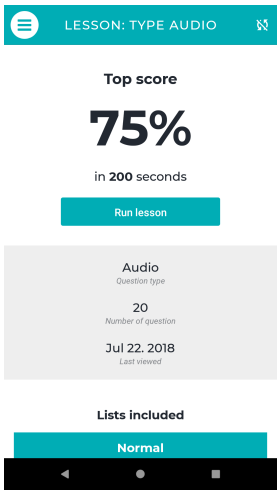
(b) Formulář vytvoření lekce

Obrázek 27: Vytvoření lekce

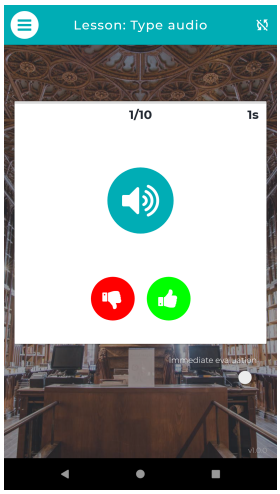
A.12 Spustit lekci

- Před spuštěním lekce se ujistěte, že máte pro tato slovíčka nastavena odpovídající políčka (strana *Word*). Jinak otázky typu *Image*, *Phrase* nebo *Custom field* nebudou fungovat.

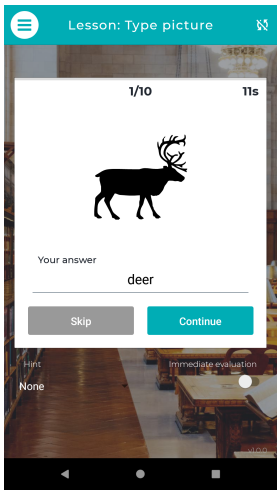
- Na straně *Lesson* klikněte na tlačítko *Run lesson*, nebo vytvořte anonymní lekci - ta se spustí okamžitě.
- Jakmile se načte otázka bude možné ji zodpovědět.
- Poté co zodpovíte všechny otázky uvidíte, které byly správně a které špatně.
- Nebo můžete zatrhnout *Immediate evaluation* - okamžitě uvidíte vyhodnocení Vaší odpovědi.
- Některé otázky nemusejí být zřejmé, v tom případě můžete použít nápovědu *Hint*. Vyberte si jaký typ nápovědy chcete, objeví se pod otázkou.
- Až zodpovíte všechny otázky uvidíte Vaše skóre.



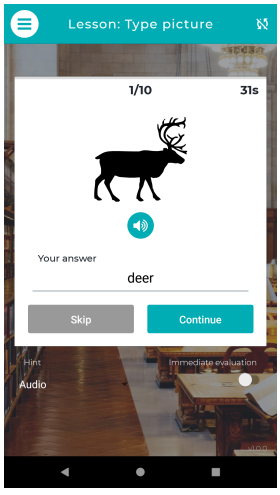
(a) Lekce



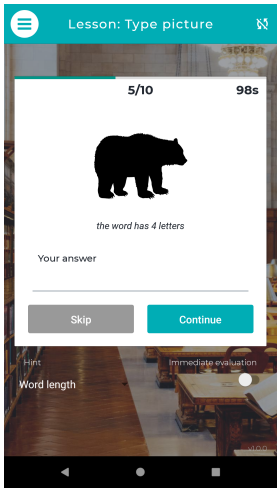
(b) Lekce nahrávka



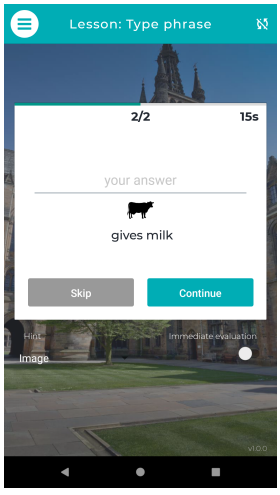
(c) Lekce obrázek



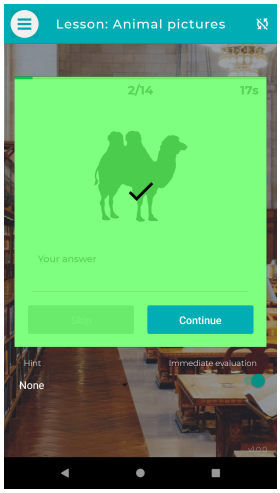
(a) Lekce obrázek, nápověda typu nahrávka



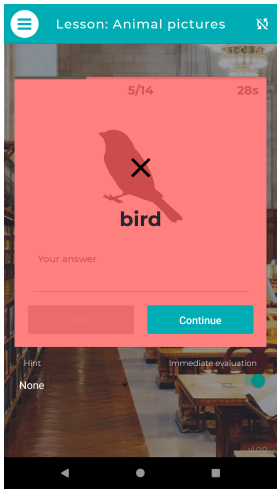
(b) Lekce obrázek, nápověda typu délka slova



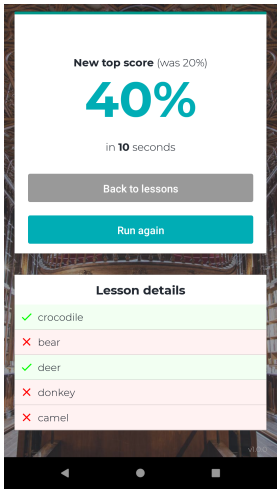
(c) Lekce fráze



(a) Lekce vyhodnocení - správně



(b) Lekce vyhodnocení - špatně



(c) Lekce ukončena

Obrázek 30: Spuštění a průběh lekce

B Obsah přiloženého CD/DVD

Na samotném konci textu práce je uveden stručný popis obsahu přiloženého CD/DVD, tj. jeho závazné adresářové struktury, důležitých souborů apod.

doc/
: Text práce ve formátu PDF, vytvořený s použitím závazného stylu KI PřF UP v Olomouci pro závěrečné práce, včetně všech příloh, a všechny soubory potřebné pro bezproblémové vygenerování PDF dokumentu textu, tj. zdrojový text, vložené obrázky, apod.

mobile/
: Kompletní adresářová struktura mobilní aplikace Vocabulary Scholar. Se všemi potřebnými (příp. převzatými) zdrojovými texty a dalšími soubory potřebnými pro bezproblémové sestavení aplikace pomocí nástroje Expo za použití veřejně dostupných knihoven získaných nástrojem npm. Nástroj Expo využívá pro sestavení vlastní servery dostupné online.

mobile/app.apk
: Instalační soubor mobilní aplikace Vocabulary Scholar pro instalaci aplikace do koncového zařízení nebo pro umístění na službu Google Play.

server/
: Kompletní adresářová struktura serverové části aplikace Vocabulary Scholar. Se všemi potřebnými (příp. převzatými) zdrojovými texty a dalšími soubory potřebnými pro zkopírování na webový server za použití veřejně dostupných knihoven získaných nástrojem npm.

server/dictionaries
: Datové soubory slovníku Wordnet pro stažení do mobilní aplikace a pro nahrání do databáze.

server/lists
: Datové soubory s připravenými seznamy slovíček, které se nabídnou pro import do aplikace Vocabulary Scholar pokud je prázdná.

web/
: Kompletní adresářová struktura webové aplikace Vocabulary Scholar. Se všemi potřebnými (příp. převzatými) zdrojovými texty a dalšími soubory potřebnými pro bezproblémové sestavení aplikace a zkopírování na webový server za použití veřejně dostupných knihoven získaných nástrojem npm.

web/build/
: Kompletní adresářová struktura sestavené webové aplikace Vocabulary Scholar. Se všemi potřebnými (příp. převzatými) zdrojovými texty, knihovnami a dalšími soubory potřebnými pro zkopírování na webový server.

web/sample-data-generator/
: Kompletní zdrojové texty pomocného programu pro generování testovacích dat pro aplikaci Vocabulary Scholar.

readme.txt
: Instrukce pro sestavení a nasazení webové aplikace Vocabulary Scholar na webový server, včetně všech požadavků pro její bezproblémový provoz. A webovou adresu, na které je aplikace nasazena pro účel testování při tvorbě posudků práce a pro účel obhajoby práce.
Instrukce pro sestavení mobilní aplikace Vocabulary Scholar, včetně všech požadavků pro její bezproblémový provoz. A webovou adresu, na které je aplikace publikována pro účel testování při tvorbě posudků práce a pro účel obhajoby práce.
Instrukce pro nasazení serverové části aplikace Vocabulary Scholar, včetně všech požadavků pro její bezproblémový provoz. A webovou adresu, na které je aplikace publikována pro účel testování při tvorbě posudků práce a pro účel obhajoby práce.

Navíc CD/DVD obsahuje:

data/
: Ukázková a testovací data použitá v práci a pro potřeby testování práce při tvorbě posudků a obhajoby práce. Tato data obsahují mimo jiné také odkazy na obrázky. Obrázky jsou v podsložce `images/`. Samotné odkazy vedou na můj server http://words.alestrunda.cz/ kde jsou tyto obrázky umístěny.

documentation/
: Kompletní dokumentace k aplikaci Vocabulary Scholar v angličtině. Vytvořeno jako webová stránka. Včetně zdrojových souborů pro její sestavení.

terms/
: Dodatečné informace pro aplikaci Vocabulary Scholar, konkrétně *licenční podmínky* a *ochrana osobních údajů*. V angličtině. Vytvořeno jako webová stránka. Včetně zdrojových souborů pro její sestavení.

user-guide/
: Kompletní uživatelský návod k aplikaci Vocabulary Scholar v češtině i v angličtině. Vytvořeno jako webová stránka. Včetně zdrojových souborů pro její sestavení.

U veškerých cizích převzatých materiálů obsažených na CD/DVD jejich zahrnutí dovolují podmínky pro jejich šíření nebo přiložený souhlas držitele copyrightu. Pro všechny použité (a citované) materiály, u kterých toto není splněno a nejsou tak obsaženy na CD/DVD, je uveden jejich zdroj (např. webová adresa) v bibliografii nebo textu práce nebo v souboru `readme.txt`.

Literatura

[1] HANNAH, John. *The Ultimate Guide to JavaScript Frameworks* [online]. [cit. 2019-3-7]. Dostupný z: ⟨http://tiny.cc/re6kaz⟩.

[2] SIMPSON, Kyle. *You Don't Know JS: Up & Going* [online]. First. 2015 [cit. 2019-3-7]. Dostupný z: ⟨https://github.com/getify/You-Dont-Know-JS⟩. ISBN 9781491924440.

[3] LIPSKI, Roman. *Single-page applications vs. multiple-page applications* [online]. [cit. 2019-3-7]. Dostupný z: ⟨https://blak-it.com/blog/spa-advantages/⟩.

[4] KONDOV, Alexander. *Lessons Learned: Code Splitting with Webpack and React* [online]. [cit. 2019-3-7]. Dostupný z: ⟨https://hackernoon.com/lessons-learned-code-splitting-with-webpack-and-react-f012a989113⟩.

[5] VEGA, Juan. *Client-side vs. server-side rendering: why it's not all black and white* [online]. [cit. 2019-3-7]. Dostupný z: ⟨http://tiny.cc/pg6kaz⟩.

[6] ALEX BANKS, Eve Porcello. *Learning React: Functional Web Development with React and Redux*. First. United States of America: O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472., 2017. 335 s. ISBN 978-1-491-95462-1.

[7] BOOTH, Joseph D. *Angular Succinctly* [online]. [cit. 2019-3-7]. Dostupný z: ⟨https://www.syncfusion.com/ebooks/angular-succinctly⟩.

[8] COPES, Flavio. *The Vue Handbook: a thorough introduction to Vue.js* [online]. [cit. 2019-3-7]. Dostupný z: ⟨http://tiny.cc/rh6kaz⟩.

[9] WIKIPEDIA. *Usage share of operating systems* [online]. [cit. 2019-3-7]. Dostupný z: ⟨http://tiny.cc/7i6kaz⟩.

[10] DOSSEY, Annie. *A Guide to Mobile App Development: Web vs. Native vs. Hybrid* [online]. [cit. 2019-3-7]. Dostupný z: ⟨https://clearbridgemobile.com/mobile-app-development-native-vs-web-vs-hybrid/⟩.

[11] DIMARZIO, J. F. *Beginning Android® Programming with Android Studio*. Fourth. Canada: John Wiley & Sons,Inc., 2017. 425 s. ISBN 978-1-118-70559-9.

[12] COWART, Jim. *What is a Hybrid Mobile App?* [online]. [cit. 2019-3-7]. Dostupný z: ⟨https://developer.telerik.com/featured/what-is-a-hybrid-mobile-app/⟩.

[13] LEBENSOLD, Jonathan. *React Native Cookbook: Bringing The Web To Native Platforms*. First. United States of America: O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472., 2018. 160 s. ISBN 978-1-491-99384-2.

[14] FARRUGIA, Kevin. *A Beginner's Guide To Progressive Web Apps* [online]. [cit. 2019-3-7]. Dostupný z: ⟨https://www.smashingmagazine.com/2016/08/a-beginners-guide-to-progressive-web-apps/⟩.

[15] AGUINAGA, Jose. *How it feels to learn JavaScript in 2016* [online]. [cit. 2019-3-7]. Dostupný z: ⟨https://hackernoon.com/how-it-feels-to-learn-javascript-in-2016-d3a717dd577f⟩.

[16] HUGO GIRAUDEL, Miriam Suzanne. *Jump Start Sass: Get Up To Speed With SASS In A Weekend*. 2016. 179 s. ISBN 978-0-9943470-1-5.

[17] SIMPSON, Kyle. *You Don't Know JS: ES6 & Beyond* [online]. First. 2015 [cit. 2019-3-7]. Dostupný z: ⟨https://github.com/getify/You-Dont-Know-JS⟩. ISBN 9781491905265.

[18] SALCESCU, Cristian. *An introduction to the Flux architectural pattern* [online]. [cit. 2019-3-7]. Dostupný z: ⟨https://www.freecodecamp.org/news/an-introduction-to-the-flux-architectural-pattern-674ea74775c9/⟩.

[19] PEAL, Gabriel. *React Native at Airbnb* [online]. [cit. 2019-3-7]. Dostupný z: ⟨https://medium.com/airbnb-engineering/react-native-at-airbnb-f95aa460be1c⟩.

[20] SPENCE, Scott. *Styled-components getting started* [online]. [cit. 2019-3-7]. Dostupný z: ⟨https://medium.com/styled-components/styled-components-getting-started-c9818acbcbbd⟩.

[21] EDUCATION, William & Mary School of. *A "Word" About Vocabulary* [online]. [cit. 2019-3-7]. Dostupný z: ⟨http://tiny.cc/0k6kaz⟩.

[22] JAAP M. J. MURRE, Joeri Dros. *Replication and Analysis of Ebbinghaus' Forgetting Curve* [online]. [cit. 2019-3-7]. Dostupný z: ⟨http://tiny.cc/il6kaz⟩.

[23] BOER, Viktor de. *Optimal Learning and the Spacing Effect: Theory,Application and Experiments based on the Memory Chain Model* [online]. 2003 [cit. 2019-3-7]. Dostupný z: ⟨http://www.few.vu.nl/~vbr240/publications/Scriptie.pdf⟩.

[24] GAO, Yunxia; TASIR, Zaidatun; HARUN, Jamalludin; JUMAAT, Nurul. Learning english vocabulary using web-based leitner box with social network. *Jurnal Teknologi*. 2015, vol. 77. Dostupný také z: ⟨http://tiny.cc/xm6kaz⟩.

[25] BABICH, Nick. *A Comprehensive Guide To Mobile App Design* [online]. [cit. 2019-3-7]. Dostupný z: ⟨http://tiny.cc/mo6kaz⟩.

[26] BROWN, Paul. *State of the Union: npm* [online]. [cit. 2019-3-7]. Dostupný z: ⟨https://www.linux.com/news/event/Nodejs/2016/state-union-npm⟩.

[27] SINGH, Ashish Nandan. *An intro to Webpack: what it is and how to use it* [online]. [cit. 2019-3-7]. Dostupný z: ⟨https://www.freecodecamp.org/news/an-intro-to-webpack-what-it-is-and-how-to-use-it-8304ecdc3c60/⟩.

[28] VIEIRA, Luis. *HTML5 Local Storage Revisited* [online]. [cit. 2019-3-7]. Dostupný z: ⟨https://www.sitepoint.com/html5-local-storage-revisited/⟩.

[29] MEAD, Andrew. *Learning Node.js Development*. 2018. 634 s. ISBN 978-1-78839-554-0.

[30] PEAL, Gabriel. *React Native at Airbnb: The Technology* [online]. [cit. 2019-3-7]. Dostupný z: ⟨http://tiny.cc/9o6kaz⟩.