FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Testování javascript aplikací
Student:	Milan Kriško
Vedoucí:	Ing. Josef Gattermayer, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2019/20

Pokyny pro vypracování

Cílem práce je analyzovat a popsat technologie určené k testování JavaScript aplikací. Součástí práce bude implementace unit, integračních a end-to-end testů existující aplikace. Hlavní cíl práce je implementace testů do existující aplikace.
1) Porovnejte a analyzujte technologie na testování klientské části aplikace.
2) Porovnejte a analyzujte technologie na testování serverové části aplikace.
3) Implementujte testy pro klientskou a serverovou část aplikace, která nebyla testy pokryta.
4) Implementujte end-to-end testy pro vybrané případy použití aplikace.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 12. února 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalárska práca

Testování javascript aplikací

Milan Kriško

Katedra softwarového inženýrství

Vedúci práce: Ing. Josef Gattermayer, Ph.D

15. mája 2019

Poďakovanie

Chcem sa poďakovať vedúcemu práce Ing. Josefovi Gattermayerovi, Ph.D za vedenie a spätnú väzbu počas písania mojej práce. Ďalej by som sa chcel poďakovať mojej rodine za podporu počas štúdia.

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval(a) samostatne a že som uviedol(uviedla) všetky informačné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona, v znení neskorších predpisov, a skutočnosť, že České vysoké učení technické v Praze má právo na uzavrenie licenčnej zmluvy o použití tejto práce ako školského diela podľa § 60 odst. 1 autorského zákona.

V Prahe 15. mája 2019 ……

České vysoké učení technické v Praze

Fakulta informačních technologií

© 2019 Milan Kriško. Všetky práva vyhradené.

Táto práca vznikla ako školské dielo na FIT ČVUT v Prahe. Práca je chránená medzinárodnými predpismi a zmluvami o autorskom práve a právach súvisiacich s autorským právom. Na jej využitie, s výnimkou bezplatných zákonných licencií, je nutný súhlas autora.

Odkaz na túto prácu

Kriško, Milan. *Testování javascript aplikací.* Bakalárska práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

Abstrakt

Táto bakalárska práca popisuje a porovnáva dostupné technológie na testovanie javascript webových aplikácií. Na základe porovnania sú ďalej v implementácií popísané testy aplikácie Vobo. Implementácia zahŕňa testovanie viacerých častí aplikácie a takisto rôzne druhy testov.

Klíčová slova automatizované testovanie, webová aplikácia Vobo, integračné testy, end-to-end testy, Node.js, React

Abstract

This bachelor thesis describes and compares available technologies for testing javascript web applications. Based on comparison are in the part of the implementation described tests of the application Vobo. The implementation consists of testing various parts of the application and various types of tests.

Keywords automated testing, web application Vobo, integration tests, end-to-end tests, Node.js, React

Obsah

Úvod **1**

1 Cieľ práce **3**

2 Analýza **5**
2.1 O projekte Vobo 5
2.2 Použité technológie na vývoj aplikácie 6
2.3 Metódy a koncepty použité pri vývoji 8
2.4 Databázové entity 9
2.5 Prípady použitia 9

3 Návrh **11**
3.1 Analyzované technológie na testovanie 11
3.2 Unit testy 13
3.3 Testovanie API 13
3.4 End-end-testy 15

4 Porovnanie technológií na testovanie **17**
4.1 Porovnanie Jest a Mocha 17
4.2 Porovnanie Puppeteer, Selenium a Cypress 22

5 Implementácia **31**
5.1 Unit testy 31
5.2 Testovanie API 32
5.3 End-to-end testy 35

6 Nasadenie **39**
6.1 Heroku 39
6.2 Priebeh nasadenia 39
6.3 Konfigurácia 40

7 Užívateľské testovanie **43**

8 Možnosti pre vylepšenie **45**
8.1 Implementácia ďalších testov 45
8.2 End-to-end testy pri nasadení 45
8.3 Pridanie testov pre React komponenty 45
8.4 Migrácia na plnohodnotný hosting 46

Záver **47**

Bibliografia **49**

A Zoznam použitých skratiek **53**

B Obsah priloženého DVD **55**

Zoznam obrázkov

2.1 Diagram prípadov použitia pri správe testov 10

4.1 Výstup z testovania pomocou frameworku Jest 20
4.2 Výstup z testovania pomocou frameworku Mocha 21
4.3 Výstup na konzoli z testovania pomocou Puppeteer 23

5.1 Okno dashboardu Cypress 37
5.2 Správa o úspešne vykonaných testov 38

Zoznam tabuliek

4.1 Funkcie vykonávané počas testovania 19

7.1 Užívateľské testy 44

Úvod

V súčasnej dobe webové aplikácie zaznamenali výrazný vývoj či už na strane frontendu alebo na strane backendu. S tým prichádza aj vývoj nových technológií na testovanie aplikácií. Práca poskytuje prehľad aktuálnych technológií pre webových vývojárov, ktorí vyberajú technológiu pre testovanie alebo začínajú s testovaním. V práci sú porovnávané frameworky, ktoré služia na spúšťanie a organizáciu testov a takisto aj technológie, ktoré testujú aplikáciu v rámci webového prehliadača. Motivácia pre písanie automatizovaných testov je zabezpečenie správnej funkčnosti vyvýjanej aplikácie. Avšak tento proces je často opomíjaný a aplikácia je testovaná len manuálne, čo je pri rozsiahlych aplikáciách veľmi časovo náročné. Ďalej tento prístup môže spôsobiť prehliadnutie chýb pri nepozornom testovaní.

Ďalej bakalárska práca popisuje súvisiace aspekty vývoja s testovaním ako je napríklad automatické nasadenie aplikácie na server, na ktorom bude aplikácia prístupná pre verejnosť alebo užívateľské testovanie. Hlavnou časťou práce je implementácia jednotlivých testov pre platformu Vobo, ktorá bola vyvynutá pomocou súčasných technológií používaných pre vývoj webových aplikácií.

Kapitola 1

Cieľ práce

Cieľom teoretickej časti práce je vysvetlenie základných pojmov pri problematike automatizovaného testovania webových aplikácií. Súčasťou teoretickej časti bude analýza technológií na testovanie, ktoré sú v súčasnosti najpoužívanejšie. Zároveň ide o technológie, ktoré sú už používané dlhší čas a nové technológie, ktoré vyšli v posledných rokoch. Na základe analýzy budú vybrané najvhodnejšie technológie z pohľadu programátora, ktoré budú v praktickej časti použité.

V praktickej časti práce budú implementované rôzne typy testov pre vybranú aplikáciu - Vobo (platforma pre učenie cudzích jazykov) a to od unit testov až po end-to-end testy, ktoré sa spúšťajú vo webovom prehliadači. Testované budú najčastejšie prípady použitia aplikácie z pohľadu učiteľov. Súčasťou práce bude aj konfigurácia automatizovaného nasadenia aplikácie na server po úspešnom vyhodnotení testov.

Kapitola 2

Analýza

2.1 O projekte Vobo

Vobo je webová aplikácia určená pre učiteľov a študentov cudzích jazykov. Aplikácia vznikla ako môj vedľajší osobný projekt, na ktorom pracujem sám.

Z pohľadu učiteľov cudzích jazykov je hlavným prínosom aplikácie možnosť vytvárať si vlastné testy s preddefinovanými správnymi odpoveďami. Učiteľ má na výber pri vytváraní testu z troch typov cvičení a to:

- preklad cudzích slovíčok,
- dopĺňanie chýbajúcich slov do viet,
- výber správnej odpovede na otázku z možností.

K cvičeniam je možné pridať Youtube video alebo sprievodný text. Cvičenia si učitelia môžu vytvárať vlastné alebo môžu použiť cvičenia vytvorené inými učiteľmi. Každý test po vytvorení získa unikátnu adresu, na ktorej žiaci test vyplnia. Testy vyplnené žiakmi aplikácia automaticky opraví a ušetrí tak učiteľom čas, ktorý by strávili manuálnym opravovaním testov. Učitelia po opravení testov získajú agregované štatistiky ohľadom úspešnosti jednotlivých odpovedí vďaka čomu uvidia, kde majú žiaci najväčšie rezervy. Okrem toho vďaka online testovaniu ušetria papier, čo je ekologicky a ekonomicky výhodnejšie. V tejto výhode by práve mohol byť veľký potenciál pre aplikáciu vzhľadom na to, že v súčasnosti svet rieši problém globálneho oteplovania a odlesňovanie je súčasťou problému.

Vobo môžu používať aj samotní žiaci pre svoju vlastnú potrebu. A to na prípravu pred testom z cvičení, ktoré vytvorili iní užívatelia alebo si môžu vytvoriť vlastné zoznamy cudzích slovíčok, ktoré sa chcú naučiť. Následne

môžu použiť mód na učenie alebo si skúsiť test nanečisto. V tejto oblasti je možné aplikáciu dosť výrazne rozšíriť o funkcionalitu.

Čo sa týka zabezpečenia kvality a funkčnosti aplikácie tak doteraz bola aplikácie testovaná len manuálne, čo nie je do budúcnosti udržateľný stav pre rozširujúcu sa funkcionalitu aplikácie.

2.2 Použité technológie na vývoj aplikácie

Pre vývoj aplikácie bola použitá celá rada aktuálne používaných technológií na vývoj webových aplikácií. Väčšinou išlo o technológie vyvýjane ako opensource či už ide o technológie pre frontendovú alebo backendovú časť.

2.2.1 HTML

HTML (Hypertext Markup Language) je jazyk používaný na tvorbu webových stránok. HTML nie je programovací jazyk ale je to značkovací jazyk, čo znamená, že tvorí systém pre identifikovanie a popis rôznych komponent dokumentu ako napríklad hlavička, odstavec a zoznam. Tvorí samotnú štruktúru dokumentu. V súčasnosti sa používa verzia HTML5, ktorá prináša nové elementy pre prehľadnejšie a logickejšie štrukturovanie dokumentov. Medzi také elementy patria napríklad header, footer alebo article, ktoré svojím názvom lepšie popisujú obsah príslušnej časti kódu. [1]

2.2.2 CSS

CSS (Cascading Style Sheets) popisuje ako obsah stránky vyzerá po vizuálnej stránke. Fonty, farby, obrázky na pozadí, medzery medzi riadkami, rozloženie stránky atď. sú kontrolované kódom CSS. S verziou CSS3 je možné pridať špeciálne efekty a základné animácia na stránku. [1]

2.2.3 Javascript

Javascript je objektovo orientovaný jazyk. Najčastejšie je používaný vo webových prehliadačoch a v tomto kontexte je rozšírený o objekty, ktoré umožňujú interagovať s užívateľom, ovládať webový prehliadač a upravovať obsah dokumentu, ktorý sa zobrazuje v okne prehliadača. Okrem objektovo orientovaných prvkov jazyk javascript obsahuje aj funkcionálne prvky. [2]

2.2.4 React

React je knižnica na vývoj klientskej časti aplikácií, ktorý vyvýja firma Facebook ako opensource projekt. Nie je to MVC framework ale stará sa len o časť View klientskej časti aplikácie. Takže pre komplexné použitie sú potrebné ďalšie technológie. Základ vývoja v React-e spočíva v členení kódu do

jednotlivých komponent vďaka ktorým je kód prehľadne štrukturovaný. Dôležitým aspektom je aj automatické prekreslenie HTML elementov pri zmene stavu komponenty. Komponenty v React-e môžu byť dvoch typov a to stavové alebo bezstavové. Stavové komponenty obsahujú dáta, ktoré môžu byť v rámci komponenty menené. Bezstavové komponenty sú vhodné pre statické vykreslovanie. Dnes je framework React najpoužívanejším frameworkom pre vývoj klientskej časti. Pričom vedie pred frameworkami ako Angular alebo Vue.js. [3] Na vývoj aplikácie bola použitá verzia 16. [4]

2.2.5 Redux

Redux je knižnica, ktorá sa používa ako kontajner stavu pre Javascript aplikácie. Nie je závislá od žiadneho frameworku alebo ďalšej knižnice. Najčastejšie sa používa v kombinácií s knižnicou React na klientskej časti aplikácie. Pri aplikácií Vobo má táto knižnica na starosti celú datovú časť na strane klienta. [5]

2.2.6 Node.js

Node.js je technológia, ktorá umožňuje spúšťanie kódu v jazyku Javascript v konzoli vďaka čomu môžeme písať backend aplikácie v Javascripte. Ide o pomerne novú technológiu, ktorá vznikla v roku 2009. Je to alternatíva k Jave a PHP pre serverovú časť aplikácie voči, ktorým ponúka iný spôsob vykonávania požiadavkov, ktoré prídu na server. [6]

2.2.7 Express

Framework, ktorý beží na Node.js. Je veľmi jednoduchý a je vhodný na použitie pre vývoj backendového API. Okrem routovania (v tomto prípade ide o smerovanie použiadavkov na správnu funkciu) ponúka aj veľmi jednoduchý spôsob ako pridať do aplikácie vlastný middleware (v tomto prípade ide o kód, ktorý sa vykoná pri každom požiadavku na server). Ide momentálne o najpoužívanejší Node.js framework. [7]

2.2.8 MongoDB

V súčasnosti najpopulárnejšia NoSQL databáza. [8] Umožňuje horizontálne škálovanie a nepotrebuje pevne danú štruktúru kolekcií (tabuľky v MongoDB), čo je veľká výhoda keď je potrebné často meniť databázový model. To bol hlavný dôvod prečo bola technológia vybraná pre vývoj aplikácie Vobo nakoľko aplikácia je neustále vo vývoji a mení sa podľa spätnej väzby od užívateľov. [9]

2.2.9 Webpack

Webpack je bundler statických modulov. Prechádza kód balíka a vytvára graf závislostí. Na základe grafu vytvorí nový balík, ktorý obsahuje len nutné minimum požadovaných súborov. Výsledok je často samotný Javascript súbor, ktorý môže byť vložený do HTML súboru a použitý v aplikácií. [10]

2.2.10 NPM

NPM je balíkový systém pre moduly Node.js. Hostuje tisíce balíkov, ktoré boli vytvorené ako opensource inými programátormi alebo spoločnosťami. Veľká väčšina balíkov, ktoré boli použité pre vývoj aplikácie Vobo pochádzajú práve z tejto platformy. [11]

2.3 Metódy a koncepty použité pri vývoji

2.3.1 REST

REST (REpresentational State Transfer) je architektonický štýl pre tvorbu a organizáciu distribuovaných systémov definovaný Royom Fiedlingom. [12] Nie je štandardom a ani návodom. REST zahŕňa sadu obmedzení ako napríklad:

- architektúru klient-server,
- stateless,
- cacheable.

2.3.2 RESTful API

API (Application programming interface) je množina funkcií, ktoré slúžia na komunikáciu medzi aplikáciami alebo komponentami aplikácie. [13] RESTful API je API, ktoré dodržiava zásady REST. V prípade testovanej aplikácie ide o webové API, ktoré funguje ako backend.

2.3.3 Single page aplikácia

Single page aplikácia (SPA) je webová aplikácia, ktorá načíta jeden HTML dokument a dynamicky aktualizuje stránku, s ktorou užívateľ interaguje. Single page aplikácie používajú AJAX (technológia, ktorá umožňuje asynchrónnu aktualizáciu dát zo servera [14]) a HTML5 na vytvorenie webových aplikácií bez opakovaného načítavania stránok. Na tvorbu takýchto aplikácií existuje v súčasnosti viacero frameworkov. Jedným z nich je aj React, ktorý bol použitý pre vývoj aplikácie Vobo. [15]

2.4 Databázové entity

V tejto časti budú popísané jednotlivé entity v databázi aplikácie Vobo. Nebudú v tejto časti popisované komplet všetky entity použité v aplikácií ale len tie, ktoré budú testované v implementačnej časti.

2.4.1 User

Používanie aplikácie je podmienené vytvorením užívateľského konta, ktoré reprezentuje entita User (užívateľ). Entita užívateľa obsahuje údaje ako email, meno, jazyk atď. Užívateľské konto je možné vytvoriť pomocou emailu alebo pomocou Facebooku či Googlu.

2.4.2 Lesson

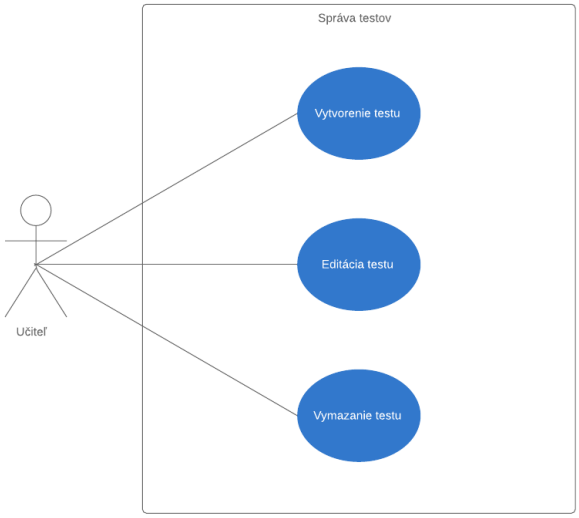
Entita pre jednotlivé cvičenia, ktorá obsahuje atribúty ako meno, typ cvičenia a autora cvičenia. Obsahuje v sebe hlavne údaje pre cvičenie ako je napríklad zoznam slovíčok, text alebo zoznam otázok s odpoveďami podľa typu cvičenia.

2.4.3 Test

Test je entita, ktorá v sebe drží zoznam cvičení, ktoré sú použité v danom teste. Okrem toho má v sebe informácie ako názov, autora a adresu, na ktorej je test prístupný pre žiakov.

2.5 Prípady použitia

Prípady použitia popisujú jednotlivé akcie vykonávané v rámci aplikácie užívateľmi. Prípady použitia môžeme v rámci aplikácie rozdeliť na dve časti a to na akcie učiteľov a žiakov. Medzi hlavné akcie učiteľa patrí správa testov. Rovnaké akcie môže učiteľ vykonávať v rámci správy cvičení, z ktorých sú tvorené samostatné testy. Z pohľadu študenta môžeme spomenúť prípady použitia ako napríklad vyplnenie testu (adresu testu, kde je test prístupný pre žiakov získa učiteľ po vytvorení testu) alebo prezeranie svojich výsledkov. Prípady použitia pre správu testov sú zobrazené v diagrame.



Obr. 2.1: Diagram prípadov použitia pri správe testov

Kapitola 3

Návrh

3.1 Analyzované technológie na testovanie

3.1.1 Jest

Testovací framework Jest je vyvýjaný spoločnosťou Facebook. Patrí medzi najpopulárnejšie frameworky na testovanie javascript aplikácií. Okrem iných je používaný firmami Twitter, Airbnb, Spotify a Instagram. Autor frameworku uvádza ako výhodu oproti ostatným frameworkom rýchlosť. Rýchlosť testovania zabezpečuje paralelné testovanie a automatická reorganizácia poradia testov. Pričom ako prvé sa pri opakovanom spustení testov spustia testy, ktoré v predchádzajúcom spustení zlyhali. [16]

3.1.2 Mocha

Mocha je jeden z najpoužívanejších frameworkov na testovanie. [17] Nie je komplexným riešením na testovanie na roziel od frameworku Jest nakoľko potrebuje pre svoju funkčnosť asertovaciu knižnicu prípadne knižnicu pre generovanie správy o pokrytí kódu testami. Pre aserty je možné použiť knižnicu `chai` alebo Node modul `assert`.

3.1.3 Chai

Chai je asertovacia knižnica pre Node a prehliadač, ktorú je možné použiť v kombinácií, s ktorýmkoľvek testovacím frameworkom. Najčastejšie sa však používa s frameworkom Mocha nakoľko framework Jest má túto funkcionalitu implementovanú a nepotrebuje ďalšie balíky. [18]

3.1.4 Selenium

Technológia, ktorá sa dá okrem testovania použiť aj na automatizovanie ľubovoľnej činnosti vo webovom prehliadači ako napríklad prehľadávanie dát na webe. Je môžné ho použiť s rôznymi programovacími jazykmi, čo je jeho veľkou výhodou oproti ostatným technológiám, ktoré budú v práci porovnávané. Ďalej dokáže pracovať s viacerými prehliadačmi vrátane najpoužívanejších prehliadačov Chrome a Firefox. [19]

3.1.5 Puppeteer

Puppeter je Node knižnica, ktorá poskytuje API na kontrolu prehliadača Chrome alebo Chromium. Môže bežať headless. Teda bez spusteného okna prehliadača alebo môže spúštať aj plnú verziu prehliadača Chrome alebo Chromium. Umožňuje napríklad:

- generovanie screenshotov stránok,
- automatizovanú správu formulárov,
- testovanie doplnkov prehliadača Chrome.

[20]

3.1.6 Cypress

Framework Cypress slúži na testovanie webových aplikácií. Je to opensource framework vyvýjaný firmou Cypress.io. Umožňuje písanie unit, integračných a end-to-end testov. Nie je postavený na frameworku Selenium a nevyžaduje žiadne ďalšie knižnice pre svoju funkčnosť. Je postavený na knižnici Mocha a Chai, ktoré sa starajú o organizáciu testov a aserty. [21]. Poskytuje dashboard, v ktorom sa dajú spúšťať samostatné testy a zároveň sledovať stav vykonávaných testov.

Medzi jeho silné stránky patrí rýchlosť a vytváranie snapshotov počas testovania, čo výrazne zjednodušuje hľadanie chýb. Ďalej je to automatické spúštanie testov pri zmene kódu, čo je veľmi užitočná vlasnosť pre programátora, ktorý hneď vidí prevedené zmeny kódu. Jednou z najväčších výhod frameworku je automatické čakanie na vykreslenie elementov v prehliadači, ktoré ešte neboli vykreslené a to z dôvodu:

- DOM sa ešte nenačítal,
- prebiehajúca animácia,
- nedokončený požiadavok na server.

Cypress umožňuje napísať si v súbore `commands.js` vlastné príkazy, ktoré môžeme používať ako metódy objektu `cy`. Vďaka tomu nemusíme svoje naprogramované časti kódu importovať do jednotlivých súborov, ktoré obsahujú samotné testy.

3.2 Unit testy

Unit testy sú testy, ktoré testujú funkčnosť jednotlivých častí alebo komponent aplikácie. . [22] Pri unit testoch nemusia byť odhalené veďlajšie efekty spôsobené zavolaním testovanej časti kódu. Preto nemôžu zabezpečovať komplexné testovanie aplikácie.

3.2.1 Testovanie pomocných funkcií

Testovanie funkcie wordSimiliarity, ktorá porovnávať nakoľko sa slová zadané študentom zhodujú so správnou odpoveďou. Funkcia vracia hodnoty v intervale od 0 do 1.

3.3 Testovanie API

Testované budú endpointy /users, /lesson a /test, ktoré sú kľúčové pre fungovanie celej aplikácie. Pri každom endpointe bude testovaná implementácia následujúcich metód GET, POST, PUT a DELETE. Jednotlivé metódy spĺňaju pri REST API nasledujúce funkcie:

- GET - získanie záznamu alebo zoznamu záznamov
- POST - vytvorenie nového záznamu
- PUT - úprava existujúceho záznamu
- DELETE - vymazanie záznamu

3.3.1 Testy pre endpoint /users

Endpoint určený pre správu užívateľov bude testovaný v následujúcich situáciách:

- vytvorenie užívateľa
- úprava užívateľa
- získanie užívateľa
- vymazanie užívateľa

Jednotlivé API metódy a endpointy pre operácie popísané vyššie sú:

- POST /users/sign-up

- PUT /users/me
- GET /users/:id
- DELETE /users/:id

3.3.2 Testy pre endpoint /lesson

Endpoint určený pre správu cvičení bude testovaný v následujúcich situáciách:

- vytvorenie cvičenia
- úprava cvičenia
- získanie zoznamu cvičenií užívateľa
- získanie cvičenia
- vymazanie cvičenia

Pre tieto operácie su definované tieto endpointy API:

- POST /lesson
- PUT /lesson/:id
- GET /lesson
- GET /lesson/:id
- DELETE /lesson/:id

3.3.3 Testy pre endpoint /test

Endpoint určený pre správu testov bude testovaný v následujúcich situáciách:

- vytvorenie testu
- úprava testu
- získanie zoznamu testov užívateľa
- získanie testu
- vymazanie testu

Jednotlivé API metódy a endpointy pre operácie popísané vyššie sú:

- POST /test
- PUT /test/:id
- GET /test
- GET /test/:id
- DELETE /test/:id

3.4 End-end-testy

End-to-end testy majú simulovať reálne používanie aplikácie užívateľom. V prípade webovej aplikácie ide o simuláciu udalostí v prehliadači, v ktorom je spustená testovaná aplikácia. V prehliadači budú testované následujúce prípady použitia.

- vytvorenie testu s jedným cvičením,
- vyhľadanie a uloženie cvičenia [1].

Nakoľko budú v tejto časti testované viaceré komponenty aplikácie tak pôjde zároveň aj o integračné testy. Integračné testy testujú viacero častí aplikácie naraz. Účel testov je odhaliť chyby pri interakcií medzi jednotlivými časťami. [23]

3.4.1 Vytvorenie testu s jedným cvičením

Tvorba testu pre žiakov je základom aplikácie. Test môže vytvoriť len zaregistrovaný užívateľ, ktorý si zvolil ako typ účtu učiteľský účet pri registrácií. Súčasťou testu sú následujúce kroky:

- Vytvorenie užívateľského účtu.
- Prechod na stránku s formulárom pre vytvorenie testu.
- Tvorba cvičenia s výberom možností.
- Uloženie testu.
- Otvorenie detailu testu.

3.4.2 Vyhľadanie a uloženie cvičenia

Užívatelia aplikácie môžu prehľadávať vytvorené cvičenia inými užívateľmi, ktoré môžu použiť vo svojich testoch. Testovanie správnosti tejto funkcionality pozostáva z týchto krokov:

- Vytvorenie užívateľského účtu.
- Prechod na stránku s vyhľadávaním cvičení.
- Spustenie vyhľadávania.
- Uloženie cvičenia.
- Kontrola prítomnosti cvičenia v zozname cvičení.

[1] V rámci aplikácie je zabezpečená existencia cvičenia

Kapitola 4

Porovnanie technológií na testovanie

Táto kapitola je porovnaním technológií z teoretickej časti. Rozoberané budú v prvej časti frameworky Jest a Mocha a ďalšie technológie potrebné pre plnohodnotné testovanie. V druhej časti budú popísané frameworky pre end-to-end testovanie Selenium, Puppeter a Cypress.

4.1 Porovnanie Jest a Mocha

Pri porovnávaní frameworkov Jest a Mocha budú brané do úvahy charakteristiky ako:

- konfigurácia,
- písanie testov,
- rýchlosť,
- generovanie správy o pokrytí kódu testami.

4.1.1 Konfigurácia

Pri integrácií akejkoľvek knižnice alebo frameworku do existujúcej aplikácie je pre programátora dôležitá náročnosť konfigurácie danej technológie. Rozdiely môžu byť v niektorých prípadoch veľmi veľké a niekedy zdĺhavá a náročná konfigurácia môže byť dôvodom prečo danú technológiu nepoužiť.

4.1.1.1 Konfigurácia Jest

Pred použitím je nutné najskôr nainštalovať potrebné závislosti na testovanie. V tomto prípade ide o balíky `jest` a `supertest`, čo je balík, ktorý umožnuje

posielať požiadavky na backendové API [24]. Tento balík je možné použiť aj pri testovaní pomocou Mocha.

Pre spustenie testovania stačí použiť príkaz `jest`. Je potrebné umožniť spúšťanie aplikácie v testovacom prostredí, čo zabezpečí nastavenie premennej `NODE_ENV` na hodnotu `test`. Vďaka tomu sa aplikácia pripojí na testovaciu databázu. Pridané riadky v súbore package.json budú vyzerať takto:

```
"scripts": {
    "jest-test": "NODE_ENV=test jest",
    ...
}
"jest": {
    "testEnvironment": "node",
    "testRegex": "./jest-tests/.*test.js$",
    "moduleNameMapper": {
        "\\.(jpg|jpeg|png|gif|eot|otf|webp|svg|ttf|woff|
            woff2|mp4|webm|wav|mp3|m4a|aac|oga)$":
            "<rootDir>/__mocks__/fileMock.js",
        "\\.(css|less|scss)$": "identity-obj-proxy"
    }
},
...
```

Hodnota parametru `testRegex` určí, kde bude spúšťač testov hľadať kód s testami. Táto konfigurácia ďalej zabezpečuje mockovanie[2] statických súborov, ktoré môžu byť importované do súboru s kódom a pri testovaní nie sú potrebné. Túto vlastnosť zabezpečuje hodnota atribútu `moduleNameMapper`.

4.1.1.2 Konfigurácia Mocha

Požadované npm balíky pre testovanie pomocou Mocha sú `mocha` a `chai`. Pre spustanie testov pomocou npm skriptov je potrebné pridať do súboru package.json kód:

```
"scripts": {
  "mocha-test": "mocha --require babel-core/register
    --require babel-polyfill --require ignore-styles
    --exit mocha-tests",
  ...
}
```

[2]Nahradenie závislostí falošnými dátami. [25]

Balíky, ktoré sú vyžadované pri spustení testovania zabezpečujú funkčnosť nových prvkov moderného javascriptu. Vďaka tejto konfigurácií bude možné spustiť testovanie z konzoli pomocou príkazu `npm run mocha-test`. Tento príkaz spustí testy v priečinku mocha-tests súbory s koncovkou `.spec.js` alebo `.test.js`.

4.1.1.3 Záver porovnania konfigurácie

Obe technológie boli nakonfigurované bez väčších problémov a obtiažnosť bola na rovnakej úrovni. Pri frameworku Mocha bolo potrebné pridať rôzne balíky Babel [3], ktoré zabezpečili transpiláciu kódu ES6. Pri frameworku Jest bolo potrebné nastaviť mockovanie statických súborov, ktoré neboli potrebné pri testovaní a spôsobovali chyby pri behu.

4.1.2 Písanie testov

Samotné písanie testov pri používaní frameworkov Jest a Mocha je rozdielne pri jednotlivých asertoch nakoľko Jest používa vlastné funkcie a Mocha používa balík Chai. Okrem toho sú inak pomenované funkcie, ktoré slúžia na organizáciu testov. Ide o funkcie:

Čas zavolania	Jest	Mocha
Pred všetkými testami	beforeAll	before
Pred každým jedným testom	beforeEach	beforeEach
Po každom jednom teste	afterEach	afterEach
Po skončení všetkých testoch	afterAll	after

Tabuľka 4.1: Funkcie vykonávané počas testovania

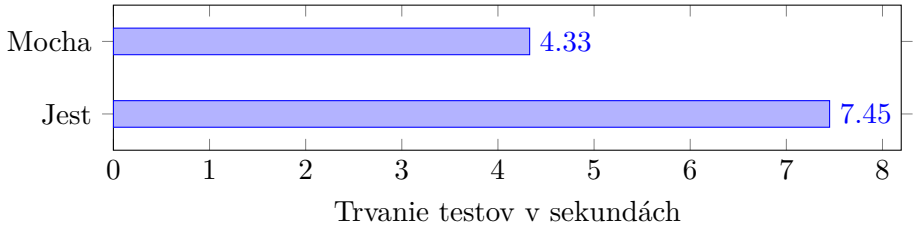
4.1.3 Rýchlosť

S veľkosťou aplikácie rastie aj čas potrebný na testovanie funkcionality. Doba trvania testov nie je zanedbateľný faktor, ktorý by sme mohli ignorovať. Firma Airbnb v minulosti migrovala z frameworku Mocha na Jest a znížili vďaka tomu čas na nasadenie z 12 minút na 4 minúty a 30 sekúnd. Prispel k tomu hlavne fakt, že Jest spúšťa testy paralelne. [26]

Pre porovnanie rýchlosti frameworkov Mocha a Jest bude použíta linuxová utilita `time`[4]. Testy pre každý framework boli spustené 5-krát a následne bol vybraný medián, aby sme dostali presnejší výsledok. Súčasťou porovnania bude jeden unit test pomocnej funkcie `wordSimiliarity` a tri endpointy API.

[3] https://babeljs.io/
[4] https://linux.die.net/man/1/time

Bližšie popísané testy jednotlivých endpointov sa nachádzajú v kapitole Návrh. Pri testovaní rýchlosti boli vynechané testy GET pre zoznam kolekcií pre endpointy /lesson a /test. Endpoint /users pre zoznam užívateľov nie je v aplikácií implementovaný. Testy boli prevedené pri zapnutom IDE, webovom prehliadači Chrome a nástroji Robomongo. Počítač, na ktorom boli prevedené testy má 16 GB pamäte RAM a dvojjadrový procesor Intel Core i7 2620M. Výsledky testovania vyzerajú následovne:



Mediánová hodnota pre framework Jest bola 7.45 sekundy. Priemerný čas trvania testov bol 8.11 sekundy.

```
minho@minho-HP-ProBook-6560b:~/Documents/projects/vobo$ time yarn test
yarn test v0.27.5
$ NODE_ENV=test jest
 PASS  test/api/lessons.test.js (5.497s)
 PASS  test/unit/wordSimiliarity.test.js
 PASS  test/api/tests.test.js (5.836s)
 PASS  test/api/users.test.js (5.867s)

Test Suites: 4 passed, 4 total
Tests:       17 passed, 17 total
Snapshots:   0 total
Time:        6.312s, estimated 7s
Ran all test suites.
Done in 7.19s.

real    0m7.450s
user    0m18.572s
sys     0m1.184s
```

Obr. 4.1: Výstup z testovania pomocou frameworku Jest

Mediánová hodnota pre framework Mocha bola 4.33 sekundy. Priemerná doba testov bola 4.73 sekundy.

Framework Mocha v tomto testovaní dopadol výrazne lepšie ako Jest. Prebehnutie testov Mocha trvalo približne polovicu času oproti Jest-u. Je dôležité povedať, že pri menšom počte testov sa nemusia prejaviť výhody paralelného testovania.

```
minho@minho-HP-ProBook-6560b:~/Documents/projects/vobo$ time yarn mocha-test
yarn mocha-test v0.27.5
$ NODE_ENV=test mocha --require babel-core/register --require babel-polyfill --require ignore-styles --exit mocha-tests/*


  Lessons endpoint
    ✓ creates lesson
    ✓ returns lesson
    ✓ updates lesson
    ✓ deletes lesson

  Tests endpoint
    ✓ creates test (38ms)
    ✓ returns test
    ✓ updates test
    ✓ deletes test

  Users endpoint:
    ✓ creates user (69ms)
    ✓ returns error when creating user with same email
    ✓ returns user
    ✓ updates user
    ✓ deletes user

  Word similarity function
    ✓ apple - orange -> returns 0
    ✓ apple - aple -> returns 0.5
    ✓ television - televison -> returns 0.75
    ✓ house - house -> returns 1


  17 passing (831ms)

Done in 4.07s.

real    0m4.333s
user    0m4.432s
sys     0m0.280s
```

Obr. 4.2: Výstup z testovania pomocou frameworku Mocha

4.1.4 Generovanie správy o pokrytí kódu testami

Code coverage popisuje aký podiel kódu je pokrytý testami. Pre vygenerovanie správy o pokrytí kódu testami stačí pri frameworku Jest pridať parameter `--coverage`. Takže príkaz na spustenie testov bude upravený na:

```
"jest-test": "NODE_ENV=test jest --coverage",
```

Po skončení testovania Jest automaticky vytvorí priečinok coverage a v ňom pre každý súbor v aplikácií HTML súbor s grafickou vizualizáciou a štatistickými údajmi o pokrytí kódu testami.

Pri knižnici Mocha je potrebné použiť balík tretej strany. Na tento účel bude použitý balík nyc [5], ktorý sa štandardne nainštaluje ako npm balík. Pre použitie treba upraviť npm skript. A to tak, že pridáme časť `nyc --reporter=html`, ktorá zároveň zabezpečí vygenerovanie pokrytia do HTML súborov, čo nie je východzia možnosť. Východzou možnosťou je textový výstup na konzoli.

```
"mocha-test": "NODE_ENV=test nyc --reporter=html mocha
    --require babel-core/register --require babel-polyfill
    --require ignore-styles --exit mocha-tests/*",
```

[5] https://istanbul.js.org/docs/tutorials/mocha/

4.1.5 Záver porovnania Jest a Mocha

Obe knižnice sú vhodnými nástrojmi na testovanie aplikácie. Jest ponúka komplexné riešenie na rozdiel od frameworku Mocha, ktorá potrebuje pre svoju funkčnosť asertovaciu knižnicu a pre generovanie správy o pokrytí kódu testami ďalšiu knižnicu. Na druhej strane pri teste rýchlosti mal navrch framework Mocha a to s viditeľným rozdielom. Pri písaní testov a používaní sa neobjavili medzi týmito dvoma frameworkami zásadne rozdiely, ktoré by boli dôvodom pre uprednostnenie jednej knižnice pred druhou. Pre ďalšie testovanie bude použitá kombinácia Mocha a Chai a to hlavne pre svoju rýchlosť, ktorá sa prejaví hlavne pri opakovanom spúšťaní testov počas vývoja.

4.2 Porovnanie Puppeteer, Selenium a Cypress

V tejto časti budú porovnávané technológie pre end-to-end testovanie a to frameworky Puppeteer, Selenium a Cypress, ktoré boli spomenuté v teoretickej časti.

4.2.1 Konfigurácia

Rovnako ako v predchádzajúcej časti bude porovnaná aj náročnosť konfigurácie jednotlivých frameworkov od inštalácie až po spustenie testov. Pre testovanie aplikácie v prehliadači je nutné najskôr aplikáciu spustiť. Na tento účel bude slúžiť príkaz `npm run start-test`, ktorý získame pridaním následujúceho riadku do časti scripts do súboru package.json:

```
"start-test": "NODE_ENV=browser-test gulp",
```

Vďaka nastaveniu premennej `NODE_ENV` bude spustená aplikácia napojená na testovaciu databázu.

4.2.1.1 Konfigurácia Puppeteer

Pre testovanie bude použitá knižnica Mocha. Ako prvý krok je potrebné pridať závislosť `puppeteer`. Konfiguračný súbor package.json v časti scripts je potrebné doplniť o následujúci riadok:

```
"mocha-puppeteer-test": "mocha --timeout 30000
    --exit puppeteer-tests/*",
```

Jednoduchý test s knižnicou Mocha, ktorý otvorí úvodnú stránku aplikácie v prehliadači, nájde na stránke text nadpisu a skontroluje text vyzerá následovne:

```
const puppeteer = require('puppeteer')
const { expect } = require('chai')

describe('Landing page test', () => {
    it('renders page', async () => {
        const browser = await puppeteer.launch()
        const page = await browser.newPage()
        await page.goto('http://localhost:3000')
        // Vyhladanie nadpisu na uvodnej stranke
        const titleElement = await page.$('.landing-quote')
        expect(titleElement).to.exist
        const text = await page.evaluate(el =>
            el.textContent, titleElement)
        expect(text).to.equal('Your vocabulary book')

        await browser.close()
    })
})
```

Testy sa vo východzom nastavení spustia v režime headless. Teda bez otvoreného okna prehliadača. Výstup z testovania:

```
minho@minho-HP-ProBook-6560b:~/Documents/projects/vobo$ yarn mocha-browser-test
yarn mocha-browser-test v0.27.5
$ mocha --timeout 10000 --exit puppeteer-tests/*


  Landing page test
    ✓ renders page (1946ms)


  1 passing (2s)

Done in 3.30s.
```

Obr. 4.3: Výstup na konzoli z testovania pomocou Puppeteer

Pre supstenie testovania s otvoreným oknom prehliadača je potrebné upraviť riadok kde sa spúšťa puppeteer na:

```
const browser = await puppeteer.launch({ headless: false })
```

4.2.1.2 Konfigurácia Selenium

Balík, ktorý je potrebný pre testovanie pomocou Selenium sa nazýva `selenium-webdriver`. Pre jednoduchšie spúšťanie v konzoli je potrebné pridať v súbori package.json riadok:

```
"mocha-selenium-test": "mocha --timeout 30000
    --exit selenium-tests/*",
```

Okrem toho je potrebné nainštalovať `geckodriver`, čo je WebDriver pre Firefox. Ide o balík, ktoré je potrebné nainštalovať do systému. Kód testu, ktorý skontroluje vykreslenie úvodnej stránky aplikácie:

```
const { expect } = require('chai')
const { Builder, By } = require('selenium-webdriver')

describe('Landing page test', () => {
    it('renders page', async () => {
        let driver = await new Builder().forBrowser('firefox')
                                        .build()
        try {
            await driver.get('http://localhost:3000')
            const title = await driver.findElement(
                By.className('landing-quote')
            ).getText()
            expect(title).to.equal('YOUR VOCABULARY BOOK')
        } finally {
            await driver.quit()
        }
    })
})
```

Výstup testu vyzerá identicky ako pri frameworku Puppeteer nakoľko je použitý framework Mocha.

4.2.1.3 Konfigurácia Cypress

Prvý krok je inštalácia balíka `cypress`. Testovanie pomocou Cypress nevyžaduje inštaláciu ďalších balíkov. Pre jednoduchšie spúšťanie je vhodné upraviť súbor package.json nasledovne:

```
    "scripts": {
        "cypress-open": "./node_modules/.bin/cypress open",
        "cypress-run": "./node_modules/.bin/cypress run",
        ...
    }
```

Prvý príkaz otvorí okno, kde je možné spustiť jednotlivé testy alebo všetky testy naraz. Druhý príkaz je pre spustenie testovania bez otvoreného okna prehliadača s výstupom na konzoli.

Kód testu úvodnej stránky pomocou frameworku Cypress vyzerá následovne:

```
describe('Landing page', function() {
    it('renders page', function() {
        cy.visit('http://localhost:3000')
        cy.get('.landing-quote')
          .should('contain', 'Your vocabulary book')
  })
})
```

4.2.1.4 Záver porovnania konfigurácie

Najjednoduchšie bolo nakonfigurovať framework Cypress a to hlavne preto, že nevyžadoval inštaláciu žiadneho ďalšieho balíka. Pozitívnym prínosom je aj to, že je možné bez konfigurácie Babelu používať všetky funkcionality ES6. Pri frameworku Cypress je písanie testov najpriamočiarejšie. Samotný kód testu vyšiel len na pár riadkov a je dobre čitateľný aj pre programátora, ktorý nevidel dokumentácia daného frameworku.

4.2.2 Používanie pri testovaní

V tejto časti budú popísané jednotlivé aspekty pri testovaní pomocou frameworkov a to:

- hľadanie elementov,
- čakanie na vykreslenie elementov,
- rýchlosť.

4.2.2.1 Hľadanie elementov

Pri testovaní je základ vyhľadávanie elementov v HTML dokumente a kontrola či obsahujú očakávaný obsah. V tejto časti budú vypísané metódy, ktoré poskytujú jednotlivé balíky na testovanie pre vyhľadávanie elementov.

Puppeteer ponúka pre hľadanie elementov metódy ako napríklad:

- page.$(selector) - vyhľadávanie podľa CSS selektoru,
- page.$x(selector) - vyhľadávanie podľa XPATH. [27]

Selenium používa na hľadanie elementov metódy findElement alebo findElements, ktoré sú súčasťou objektu drive. Ako parameter prijíma navratovú hodnotu jednej z metód objektu By:

- By.className(name),
- By.css(selector),
- By.id(id),
- By.xpath(id).

[28]

Príklad použitia v kóde:

```
driver.findElement(By.className('popup-wrapper-close'))
```

Cypress pre túto funkcionalitu ponúka metódu get, ktorá príjima ako parameter selektor. Okrem toho je možné ďalej použiť metódy ako:

- children() - potomok elementu,
- first() - prvý nájdený element,
- last() - posledný nájdený element,
- eq(n) - n-tý element.

Niekedy môže byť problém zvoliť vhodný selektor pre hľadaný element. Cyprees ponúka nástroj Selector Playground, ktorý pomôže vybrať unikátny selektor pre element, ktorý budeme potrebovať pri testovaní.

4.2.2.2 Čakanie na vykreslenie elementov

Pri testovaní aplikácií v prehliadačí je často potrebné čakať na vykreslenie elementov. Tento problém rieši každý framework trochu odlišne, čo bude práve popísané v tejto časti.

Selenium má na tento problém objekt until. Jeho použitie vyzerá takto:

```
await driver.wait(
    until.elementLocated(By.className('class-selector')),
    5000
)
```

Druhý parameter metódy je maximálna doba čakania na element v milisekundách.

Puppeteer má metódu waitForSelector, ktorá je súčasťou objektu page.

```
await page.waitForSelector('#popup-positive')
```

Cypress tento problém rieši inak. Na element, ktorý nedokáže nájsť čaká automaticky. Je to užitočná vlastnosť a ušetrí niekoľko riadkov kódu. Hlavne ak sú v aplikácií animácie alebo časté asynchrónne volania na backend.

4.2.2.3 Programátorom definované príkazy

Puppeteer a Selenium neposkytuje API pre pomocné funkcie. Pre implementáciu vlastných príkazov je potrebné vytvoriť funkcie, ktoré budú prijímať objekt page alebo driver.

Cypress ponúka pre tento problém svoje vlastné riešenie. V súbore `commands.js` si programátor môže sám nadefinovať časti kódu, ktoré sa pri testoch budú opakovať. Súbor sa nachádza na ceste `cypress/support/commands.js`. Nie je potrebné ho vytvárať nakoľko je vytvorený automaticky počas inštalácie. Vytvorenie príkazu pre odhlásenie z aplikácie môže vyzerať takto:

```
Cypress.Commands.add('signOut', () => {
  cy.get('#profile-header-menu')
    .click()

  cy.get('#sign-out-link')
    .click()
})
```

Po napísaní vyššie uvedeného kódu už nie je potrebné ďalej importovať žiadny súbor do súboru s testami. Príkaz je prístupný ako metóda objektu cy. V teste sa použije jednoduchým zavolaním metódy `cy.signOut()`. V tejto oblasti ide o užitočnú vec, ktorá ušetrí programátorovi písanie kódu v porovnaní so Selenium a Puppeteer.

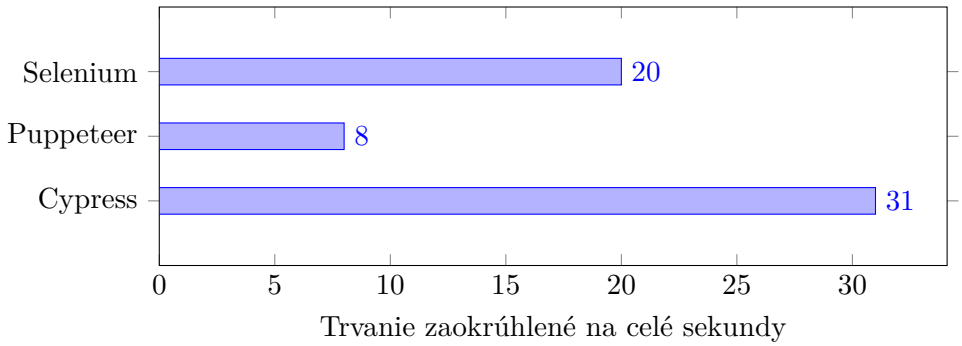
4.2.3 Rýchlosť

Pre porovnanie rýchlosti budú použité tieto dva testy:

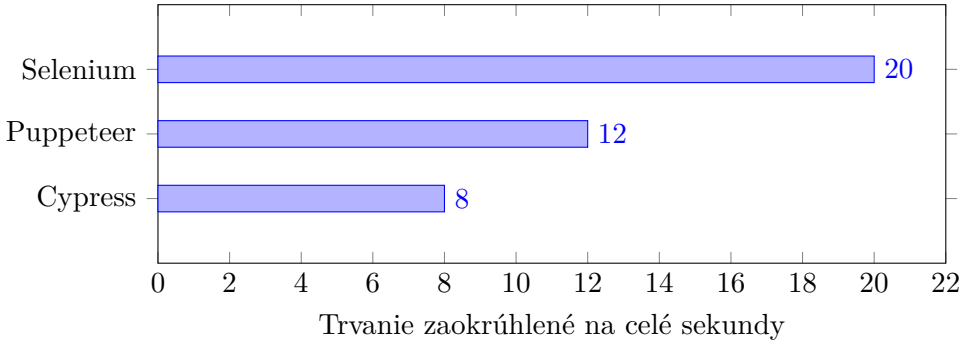
- vykreslenie úvodnej stránky
- vyhľadávanie cvičení

Pri testovaní bude vybraná znovu ako v predchádzajúcej časti mediánová hodnota z piatich meraní pre každý framework.

Výsledky testovania rýchlosti v režime headless:



V režime headless výrazne vedie Puppeteer s približne 8 sekundami. Najhoršie dopadol Cypress s 31 sekundami. Výsledky testovania rýchlosti s otvoreným oknom:



S otvoreným prehliadačovým oknom vyzerali výsledky testov v prípade Puppeteer a Cypress úplne inak. Selenium sa pohybovalo na rovnakých hodnotách ako v predchádzajúcom teste. Puppeteer bol približne o polovicu pomalší. Víťazom testu pri otvorenom okne bol Cypress. Treba však povedať, že Cypress pri otvorenom dashboarde umožňuje okamžité spúšťanie testov v poradí. To výrazne šetrí čas na spustenie samotného testovania a vďaka tomu vedie. Počas vývoja aplikácie je dôležitejší údaj s otvoreným oknom nakoľko programátor môže hneď vidieť kde test zlyhal.

4.2.4 Záver porovnania

Pre implementáciu end-to-end testov bude použitý framework Cypress. Hlavne pre pohodlné užívateľské prostredie, ktoré poskytuje pre testovanie. Automatické spúšťanie testov pri zmene súboru s kódom testov bez manuálnej konfigurácie je ďalšia výhoda, ktorá zefektívňuje prácu programátora. Užitočnou

vlastnosťou Cypress je možnosť prezerania priebehu testu smerom do minulosti a spätne prezerať tak každú interakciu v aplikácií či už je to vyhľadanie elementu na stránke, kliknutie na tlačidlo alebo vypĺňanie vstupu do formulára.

Kapitola 5

Implementácia

V tejto kapitole bude popísaná implementácia unit, integračných a end-to-end testov popísaných v kapitole Návrh. Pre spúšťanie testov bol vybraný framework Mocha s knižnicou Chai, ktorý vyšiel ako vhodnejšia voľba na základe porovnania v predchádzajúcej kapitole.

5.1 Unit testy

Kód súboru `wordSimiliarity.test.js` výzerá následnovne:

```
import { expect } from 'chai'
import similiarity from '../../src/common/lib/wordSimiliarity'

describe('Word similiarity function', () => {
    it('apple - orange -> returns 0', () => {
        expect(similiarity('apple', 'orange')).to.equal(0)
    })
    it('apple - aple -> returns 0.5', () => {
        expect(similiarity('apple', 'aple')).to.equal(0.5)
    })
    it('television - televison -> returns 0.75', () => {
        expect(similiarity('television', 'televison'))
            .to.equal(0.75)
    })
    it('house - house -> returns 1', () => {
        expect(similiarity('house', 'house')).to.equal(1)
    })
})
```

Tento jedoduchý test len skontroluje správne návratové hodnoty pre dvojice slov, ktoré dostane funkcia na vstupe. Funkcia nemá žiadne vedľajšie účinky a preto je vhodná na testovanie pomocou unit testu.

5.2 Testovanie API

Pre implementáciu testov backendového API bude použitý tiež framework Mocha a Chai. Ukážky kódu testov budú pre endpoint na správu užívateľského profilu. Pri testovaní API budú kontrolované aj stavové HTTP kódy, ktoré sú súčasťou odpovede backendového servera. V tomto prípade to budú kódy:

- 200 OK,
- 201 Created,
- 404 Not Found,
- 409 Conflict. [29]

5.2.1 Vytvorenie užívateľského konta

Okrem základných operácií bude testovaná aj chybová situácia ako napríklad, keď sa bude užívateľ pokúšať vytvoriť účet s emailom, ktorý už bol raz použitý pre vytvorenie účtu.

Pre účely testovania bude vytvorený účet s nasledujúcimi atribútmi:

```
const sampleUser = {
    name: 'users.test',
    email: 'users.test@test.com',
    password: 'mocha',
    role: roles.TEACHER,
    userLanguage: 'sk',
    learningLanguage: 'en',
    privacyPolicyAgreement: true,
}
```

Jednotlivé testy popísané nižšie budú obalené vo funkcií describe, ktorá združuje skupinu testov. Vytvorenie užívateľa prebehne odoslaním POST požiadavku na server s objektom popísaným vyššie. V odpovedi servera by mal byť vytvorený užívateľ s tokenom pre autorizáciu, ktorý sa uloží do premennej a bude sa používať pre ďalšie testy. Status kód servera by mal byť 201.

Email užívateľa musí byť pre každého unikátny preto by API nemalo umožniť vytvorenie účtu s rovnakým emailom. Okrem toho by malo API vrátiť status kód 409.

```
describe('Users endpoint:', () => {
    it('creates user', async () => {
        const res = await request(app).post('/api/v1/users/sign-up')
                                      .send(sampleUser)
        user = res.body.data
        expect(user.name).to.equal(sampleUser.name)
        expect(user.email).to.equal(sampleUser.email)
        expect(user.role).to.equal(sampleUser.role)
        expect(user.id).to.exist
        expect(user.tokens).to.exist
        expect(user.tokens.access).to.exist
        token = user.tokens.access
        expect(res.body.code).to.equal(201)
    })

    it('returns 409 when creating user with same email', async () => {
        const res = await request(app).post('/api/v1/users/sign-up')
                                      .send(sampleUser)
        expect(res.body.code).to.equal(409)
    })
    ...
```

5.2.2 Získanie užívateľského profilu

Vytvorený užívateľ dostane pridelený unikátny identifikátor. Na základe tohto indentifikátora môžeme poslať požiadavok na adresu /users/:id metódou GET a dostaneme užívateľa s daným identifikátorom. Test kontroluje či vrátený objekt obsahuje správne parametre a či bol vrátený správny status kód a to v tomto prípade 200. Túto funkcionalitu testuje následujúci test:

```
it('returns user', async () => {
    const res = await request(app)
        .get(`/api/v1/users/${user.id}`)
        .set('x-access-token', token)
    const userById = res.body.data
    expect(userById.id).to.equal(user.id)
    expect(userById.name).to.equal('users.test')
    expect(userById.userLanguage).to.equal('sk')
    expect(userById.tokens).to.not.exist
    expect(res.body.code).to.equal(200)
})
```

5.2.3 Úprava užívateľského profilu

V aplikácií môže užívateľ upravovať len svoj účet. Identifikácia užívateľa prebehne v aplikácií na základe odoslaného tokenu. Pri úprave záznamu by malo API vrátiť status kód 200:

```
it('updates user', async () => {
    const res = await request(app)
        .put('/api/v1/users/me')
        .set('x-access-token', token)
        .send({ userLanguage: 'de' })
    const updatedUser = res.body.data
    expect(updatedUser.id).to.equal(user.id)
    expect(updatedUser.name).to.equal('users.test')
    expect(updatedUser.userLanguage).to.equal('de')
    expect(res.body.code).to.equal(200)
})
```

5.2.4 Vymazanie užívateľského profilu

Pri mazaní užívateľa sa najskôr odošle požiadavok na zmazanie a vykonanie operácie sa overí požiadavkom GET na cestu pre vymazaného užívateľa. Požiadavok by mal vrátiť podľa implementácie API status kód 404.

```
it('deletes user', async () => {
    let res = await request(app)
        .delete(`/api/v1/users/${user.id}`)
        .set('x-access-token', token)

    user = res.body.data
    expect(user.name).to.equal(sampleUser.name)

    res = await request(app)
        .get(`/api/v1/users/${user.id}`)
        .set('x-access-token', token)

    expect(res.body.code).to.equal(404)
})
```

5.3 End-to-end testy

V tejto časti bude popísaná implementácia end-to-end testov v technológií Cypress. Technológia bola vybraná na základe porovnania v predchádzajúcej kapitole. Pre každý test bude na začiatku vytvorený užívateľský profil typu učiteľ. Pre túto činnosť bude v súbore commands.js vytvorený príkaz, ktorý bude možné použiť vo všetkých testoch.

```
Cypress.Commands.add('createTeacher', () => {
    const date = moment(new Date()).format('HH.mm.ss')
    const userEmail = `teacher-${date}@cypress.com`
    // Otvorenie aplikacie
    cy.visit('http://localhost:3000')
    cy.get('.start-button').click()
    // Vyber typu uctu a jazykov
    cy.contains('I am teacher').click()
    cy.contains('Continue').click()
    cy.contains('German').click()
    cy.get('#sign-up-user-language')
    ...
})
```

5.3.1 Vytvorenie testu pre žiakov

Na začiatku testu sa vytvorí užívateľský profil pre učiteľa. Test pre žiakov bude obsahovať len jedno. Nakoľko cieľom je otestovať samotnú tvorbu testov a nie samostatných cvičení. Následujucí kód otvorí formulár testu a vyplní jeho názov.

```
describe('Creates test', function() {
  it('creates successfully test', function() {
    cy.createTeacher()
    // Presmerovanie na formular s testom
    cy.get('#welcome-create-test')
      .click()
    // Skrytie informacneho popupu
    cy.get('#popup-positive')
      .click()
    // Zadanie nazvu testu
    cy.get('input')
      .type('Sample test')
    ...
```

Po vytvorení testu bude overená existencia testu v zozname testov užívateľa a prítomnosť vytvoreného cvičenia.

```
cy.get('.tests-list .test-info h3 a')
  .first()
  .click()
// Kontrola detailu testu
cy.get('.page-header h2')
  .should('contain', 'Sample test')
// Kontrola pritomnosti vytvoreneho cvicenia
cy.get('.test-lessons-list .blue-button span')
  .should('contain', 'ABC')
```

Na konci testu bude užívateľ odhlásený z aplikácie pomocou pripraveného prikázu `cy.signOut()`, ktorý už bol implementovaný počas porovnávanie frameworkov. Užívateľa je potrebné odhlásiť z aplikácie aby bolo zabezpečené, že ďalší test začne vo východzom stave a to na domovskej stránke bez prihláseného užívateľa nakoľko údaje o prihlásenom užívateľovi sú uložené v prehliadači.

5.3.2 Vyhľadanie a uloženie cvičenia

Učiteľ si nemusí vytvárať testy zo svojich cvičení ale môže použiť pre už vytvorené cvičenia od ostatných uživateľov prípadne cvičenia, ktoré sú súčasťou aplikácie. Ako aj pri predchádzajúcom teste aj v tomto prípade bude na začiatku vytvorený nový užívateľ. Začiatok testu bude vyzerať následovne:

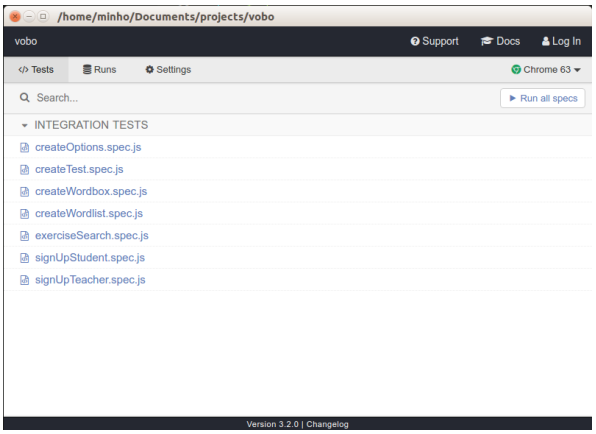
```
describe('Search exercise', function() {
  it('finds and saves exercise', function() {
    const lessonTitle = 'onboarding'
    cy.createTeacher()
    // Presmerovanie na stranku vyhladavania
    cy.get('#welcome-search-exercise')
      .click()
    cy.get('#popup-positive') // Skrytie informacneho popupu
      .click()
    cy.get('.search-input') // Zadanie mena cvicenia
      .type(lessonTitle)
    cy.get('#search') // Potvrdenie vyhladavania
      .click()
    ...
```

Po nájdení cvičenia test cvičenie uloží a presmeruje okno na stránku so zoznamom uložených cvičení užívateľa. Na tejto stránke skontroluje prítomnosť cvičenia a otvorí detail uloženého cvičenia. Na záver testu bude skontrolovaná zhoda názvu vyhľadávaného cvičenia. Na záver testu bude užívateľ odhlásený z aplikácie.

```
    ...
    // Prechod na stranku so zoznamom cviceni
    cy.get('#nav-link-lessons')
      .click()
    cy.get('#popup-positive') // Skrytie informacneho popupu
      .click()
    // Otvorenie detailu cvicenia
    cy.get('.item-box h3')
      .click()
    cy.get('.page-header h2') // Kontrola spravneho textu
      .should('contain', lessonTitle)
    cy.signOut()
  })
})
```

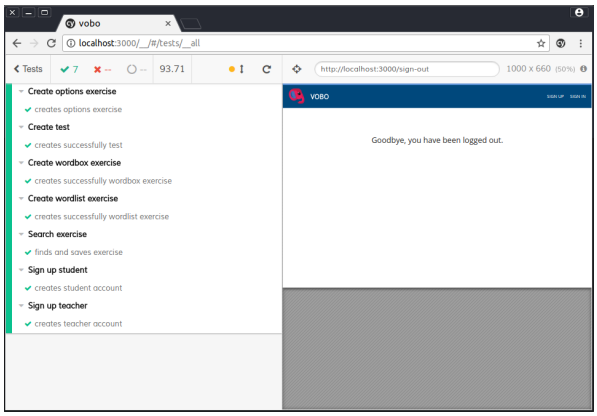
5.3.3 Spustenie testov

Pri testovaní s otvoreným dashboardom je možné jednotlivé testy spúšťať kliknutím na názov testu. Po kliknutí na test sa zobrazí okno s aplikáciou a spustí sa priebeh testu. Počas testovania sa v ľavom paneli zobrazuje stav vykonávania testov. Okrem toho programátor môže vidieť aktuálny stav aplikácie v prehliadači a má pristup aj ku konzole.



Obr. 5.1: Okno dashboardu Cypress

Dashboard takisto umožňuje spustenie všetkých testov sekvenčne za sebou. Pri spustení testovania sa v hornom paneli zobrazuje správa o počte úspešných a neúspešých testoch. V prípade, že test zlyhá tak sa testovanie zastaví a už nepokračuje vo vykonávaní testov. Správa o chybe v aplikácií pri behu prostredia Cypress je dobre popísaná a vždy je vyznačené miesto kde chyba nastala. Testovanie pomocou grafického užívateľského rozhrania dosť uľahčuje testovanie a hľadanie chýb.



Obr. 5.2: Správa o úspešne vykonaných testov

Pri spustení testov v konzoli dostaneme jednoduchý výstup v podobe zoznamu testov, kde pri každom teste je doba trvania testu a údaj či bol test úspešný alebo nie. Pri vývoji testovanie pomocou Cypress tento spôsob testovania nie je veľmi vhodný nakoľko testovanie aplikácie trvá výrazne dlhšie v porovnaní s testovaním v prehliadačovom okne. Čo je ale potrebné spomenúť je, že pri zlyhaní testu Cypress automaticky vytvorí fotku obrazovky kde aplikácia zlyhala a uloží ju do priečniku `cypress/screenshots`. Táto funkcionalita je spustená vo východzom nastavení a nie je potrebné kvôli tomu meniť žiadny konfiguračný súbor.

Testované prípady použitia sú úplne najdôležitejšou funkcionalitou aplikácie a v kombinácií s testami backendového API pokrývajú väčšinu funkcionality aplikácie. Ďalšie prípady použitia sú implementované v prílohe. Testy zahŕňajú aj tvorbu samostatných všetkých troch typov cvičení, ktoré nemusia byť po vytvorení automaticky použité v teste.

Kapitola 6

Nasadenie

Po naprogramovaní zmien v aplikácií a po dopísaní testov na záver vývojového cyklu je potrebné aplikáciu nasadiť na server aby bola pristupná pre užívateľov. Pri nasadení bude použitá technika Continuous integration (CI), čo je proces pri, ktorom je každá integrácia skontrolovaná automatickým zostavením aplikácie a testovaním. [30] Pre tento proces bude použitá služba TravisCI, ktorú je možné používať bezplatne. [31]

6.1 Heroku

Heroku funguje ako PaaS (Platforma ako služba) a ponúka verziu zadarmo, ktorá bude použitá pre nasadenie aplikácie Vobo. V dashboarde služby sa dá pár kliknutiami nastaviť automatické nasadenie z repozitára Github. Takisto pridanie CI sa aktivuje jedným zakliknutím checkboxu. [32]

6.2 Priebeh nasadenia

Nasadenie aplikácie bude spustené automaticky pushnutím zmien programátorom do repozitára na službe Github, ktorý je prepojený so službami Heroku a TravicsCI. Okrem repozitára je možné ďalej špecifikovať aj konkrétnu vetvu v repozitári.

6.2.1 Proces integrácie TravisCI

Proces CI, ktorý sa spustí na službe TravisCI sa skladá z viacerých krokov a to:

- Naklonovanie repozitára.
- Inštalácia závislostí.
- Spustenie testov.

- Nasadenie na Heroku.

6.2.2 Proces nasadenia na Heroku

Po úspešnom prevedení všetkých krokov popísaných vyššie sa spustí automaticky proces nasadenia na platforme Heroku. Ten sa skladá takisto z viacerých krokov. Kroky, ktoré budú prevedené na platforme Heroku sú v následujúcom zozname:

- Naklonovanie repozitára.
- Inštalácia závislostí.
- Zostavenie aplikácie.
- Spustenie aplikácie.

6.3 Konfigurácia

V konfigurácií pre proces CI je potrebné pridať súbor .travis.yml. Konfigurácia zahŕňa informácie o použitom programovacom jazyku, potrebnej verzií, potrebných krokov pre zostavenie a otestovanie. Vďaka atribútu cache nie je potrebné zakaždým sťahovať npm balíky, čo ušetrí čas pri nasadení. Na záver je definované kde sa má nasadiť aplikácia.

```
branches:
  only:
  - dev
// konfiguracia jazyka a prostredia
language: node_js
// pre spravne fungovanie je potrebna aspon verzia 8
node_js:
  - "8.11.1"
cache: npm
jobs:
  include:
    script: gulp // spustenie zostavenia aplikacie
    script: npm run mocha-test // spustenie testov
  deploy:
    provider: heroku
    app: vobo-dev
    api_key:
      secure: "HEROKU_API_KEY" // Placeholder pre kluc
    on: dev
```

Súbor je potrebné uložiť do koreňového priečinka aplikácie a nahrať zmeny na repozitár. Tým bude konfigurácia celého procesu hotová.

Proces nasadenia je plne automatizovaný. Po pushnutí zmien do vetvy dev v repozitári Github sa automaticky spustí proces zostavenia a následne testovania aplikácie. Po úspešnom prebehnutí testov bude aplikácia nasadená na platforme Heroku. Celý proces nasadenia na Heroku a konfigurácia TravisCI boli vďaka prehľadnej dokumentácií jednoduché a bezproblémové. Pokiaľ sú testy aplikácie napísané dobre tak ide o bezpečný spôsob ako nasadiť aplikáciu na server vzhľadom k tomu že pri akejkoľvek chybe, ktorá sa pri nasadení objaví sa proces zastaví a pre uživateľov ostane dostupná pôvodná aplikácia bez prevedených zmien v kóde. Pri ručnom nasadení na server by hrozilo, že proces nasadenia zlyhá a aplikácia by bola úplne nedostupná pre užívateľov. Pre databázu aplikácie je použitá služba mLab[6], pre ktorú je len nakonfigurovaná cesta v rámci aplikácie takže nasadenie databázi nie je v tomto prípade potrebné riešiť.

[6]https://mlab.com/

Kapitola 7

Užívateľské testovanie

Pre overenie užívateľskej prívetivosti budú vykonané užívateľské testy so 4 užívateľmi. Každý užívateľ dostane za úlohu, vytvoriť jednoduchý test s cvičeniami, ktoré boli užívateľom dodané vo formáte textového dokumentu. Išlo o krátke a jednoduché zadanie pre predmet angličtiny.

Inštrukcie pre užívateľov boli následujúce:

- Zaregistruje sa pomocou emailu.
- Otvorte stránku na vytvorenie testu.
- Vo formulári vytvorte všetky tri typy cvičení.
- Uložte vytvorený test.
- Prejdite na stránku pre vyplnenie testu.
- Vyplňte test z pohľadu žiaka.
- Zobrazte vyhodnotené výsledky testu.

Testovacia skupina boli mladí ľudia, ktorí sú technicky zdatní čo sa týka používania moderných technológií. Aj keď nepatria do cieľovej skupiny aplikácie, testovanie by malo odhaliť prípadné chyby pri používaní nakoľko ide o oblasť, ktorá je každému dobre známa zo štúdia na základnej alebo strednej škole. Medzi účastníkmi testovaia boli:

- Užívateľ A: žena, študentka VŠ, 23 rokov.
- Užívateľ B: žena, pracujúca, 43 rokov.
- Užívateľ C: žena, študentka VŠ, 23 rokov.
- Užívateľ D: muž, pracujúci, 20 rokov.

Každý účastník testovania dostal na konci online dotazník kde mal číselne vyhodnotiť užívateľskú prívetivosť danej časti úlohy a to v rozmedzí 1-10. Kde 1 znamenalo, že sa mu nepodarilo dokončiť danú úlohu a 10 znamenalo, že úloha bola pre neho jednoduchá.

Aktivita/Užívateľ	A	B	C	D
Registrácia do aplikácie	9	10	10	10
Otvorenie formulára s testom	9	10	8	10
Vytvorenie cvičenia s možnosťami	8	10	8	10
Vytvorenie cvičenia s doplňovaním	7	10	6	7
Vytvorenie cvičenia s prekladom	9	10	9	9
Celkové vytvorenie testu	10	10	7	9
Vyplnenie testu z pohľadu žiaka	10	10	10	10
Zobrazenie výsledkov testu	9	10	9	10

Tabuľka 7.1: Výsledky užívateľských testov

Z výsledkov testovania sa dá vyčítať, že časť, na ktorú sa treba zamerať je formulár na vytvorenie cvičenia s doplňovaním, kde užívateľom nie je na prvý pohľad jasné ako vybrať slovo, ktoré bude dopĺňané v texte. Pre túto časť by bolo vhodné spraviť v aplikácií jednoduchého sprievodcu, nakoľko ide o nie veľmi intuitívny spôsob tvorby tohto typu cvičenia. Zvyšné dva typy cvičení boli pre užívateľov jednoduché a intuitívne na vytvorenie a takisto aj ostatné časti užívateľského testu. Medzi najjednoduchšie časti patrili úlohy ako registrácia do aplikácie, vyplnenie testu z pohľadu žiaka alebo zobrazenie výsledku testu.

Kapitola 8

Možnosti pre vylepšenie

Je množstvo vecí, ktoré by bolo možné vylepšiť v rámci aplikácie Vobo nakoľko ide stále o aplikáciu, ktorá je priebežne vylepšovaná na základe užívateľskej spätnej väzby. Zásadný priestor pre vylepšenie je v užívateľskej prívetivosti v časti určenej pre učiteľov a rozšírení o ďalšie funkcie.

8.1 Implementácia ďalších testov

Aplikácia je stále doplňovaná o novú funkcionalitu a vylepšenia. Pre túto funkcionalitu bude potrebné implementovať ďalšie testy. Do budúcnosti budú pridané možnosti pre správu tried a domácich úloh, ktoré sú momentálne stále vo vývoji. Ďalej je aplikáciu potrebné doplniť o testy užívateľských prípadov z pohľadu žiakov. V tejto časti sú potrebné end-to-end testy nakoľko testy pre backendovú časť aplikácie pre túto funkcionalitu sú už implementované.

8.2 End-to-end testy pri nasadení

Pri nasadení aplikácie sa nespúšťajú end-to-end testy. Do budúcnosti by bolo dobré nakonfigurovať proces nasadenia tak aby zahŕňal aj testy v prehliadačovom okne pre lepšie zabezpečenie kvality a funkčnosti aplikácie.

8.3 Pridanie testov pre React komponenty

Testovanie klientskej časti nemusí byť pokryté len testami v prehliadači ale je možné testovať jednotlivé komponenty pomocou unit testov oddelene. Je to spôsob testovania vhodný skôr počas vývoja pre otestovanie jednodlivých častí.

8.4 Migrácia na plnohodnotný hosting

Použité riešenie na nasadenie má svoje obmedzenia ako napríklad automatické uspávanie po 30 minútach nečinnosti. Pri ostrej prevádzke to nie je vhodné riešenie a do budúcnosti keď bude aplikácia pravidelne používaná na testovanie alebo na štúdium bude potrebné použiť pokročilejšie riešenie. Jedným z riešení môže byť platená verzia Heroku alebo služba od iného poskytovateľa ako napríklad služba Amazon Web Services[7] od firmy Amazon, DigitalOcean[8] alebo Google Cloud [9]. Všetky tri služby ponúkajú cenovo dostupné riešenia.

[7] https://aws.amazon.com/
[8] https://www.digitalocean.com/
[9] https://cloud.google.com/

Záver

Práca bola zameraná na testovanie javascript webových aplikácií. Dôležitou súčasťou práce bolo porovnanie technologií na testovanie. Porovnávané boli frameworky, ktoré slúžia na spúšťanie a organizáciu testov ako Jest a Mocha. V tomto porovnaní bol vybraný na implementáciu framework Mocha. Ďalej boli porovnávané frameworky pre end-to-end testovanie ako Puppeteer, Selenium a Cypress. Víťazom tohto porovnania sa stal framework Cypress pre svoju pohodlnosť pri písaní testov a užívateľského prostredia vytvoreného pre testovanie.

Cieľom práce bolo implementovať testy pre backendové API a zároveň aj klientskú časť aplikácie čo sa podiarilo pomocou end-to-end testov, ktoré tvoria významnú časť testovania. Implementácia testov backendového API bola implementovaná pomocou frameworku Mocha a knižnice Chai. Ďalej bola aplikácia nasadená na server Heroku a bol nakonfigurovaný automatizovaný proces nasadenia aj za použitia služby TravisCI. Okrem toho boli prevedené aj užívateľské testy, ktoré napomoholi odhaliť časti, ktoré nie sú pre užívateľov zrozumiteľné a dávaju podklad pre ďalší vývoj aplikácie.

Čo sa týka samotnej aplikácie Vobo, na ktorej boli testy implementované tak sa jedná o použiteľnú verziu pre testovanie žiakov na predmetoch pre cudzie jazyky. V súčasnosti sa zatiaľ nepodarilo presadiť používanie nakoľko sú ešte potrebné vylepšenia z hľadiska užívateľskej prívetivosti. Ďalšou prekážkou pri používaní aplikácie je samotná tvorba testov, ktorá zaberá pomerne veľa času. Tento problém by do budúcnosti mohlo pomôcť odstrániť vytvorenie obsahu pre jednotlivé jazyky ako angličtina, franzúština alebo nemčina. Samotné použitie aplikácie nemusí byť len pre testovanie ale môže slúžiť aj ako miesto kde si učitelia navzájom zdieľajú testy prípadne cvičenia, čo by tiež šetrilo čas potrebný na tvorbu testov. Okrem základných a stredných škôl je možné aplikáciu použiť aj pre výuku na jazykových školách. V búducnosti ale

nič nebráni tomu aby aplikáciu pre učenie mohol použiť ktokoľvek kto sa učí cudzí jazyk a nie len študenti na školách. Aplikácia je voľne dostupná pre použitie v prehliadači na adrese vobo.io.

Bibliografia

1. ROBBINS, Jennifer Niederst. *Learning web design: A beginner's guide to HTML, CSS, JavaScript, and web graphics*. Ö'Reilly Media, Inc.", 2012.
2. FLANAGAN, David. *JavaScript: the definitive guide*. Ö'Reilly Media, Inc.", 2006.
3. *Front-end Frameworks - Overview* [online] [cit. 2019-04-26]. Dostupné z: `https://2018.stateofjs.com/front-end-frameworks/overview/`.
4. *React* [online]. Facebook Inc. [cit. 2019-04-18]. Dostupné z: `https://reactjs.org/`.
5. ABRAMOV, Dan. *Redux* [online] [cit. 2019-04-18]. Dostupné z: `https://redux.js.org/`.
6. *Node.js* [online]. Node.js Foundation [cit. 2019-04-18]. Dostupné z: `https://nodejs.org/en/`.
7. *Back-end Frameworks - Overview* [online] [cit. 2019-04-16]. Dostupné z: `https://2018.stateofjs.com/back-end-frameworks/overview/`.
8. *Most Popular NoSQL Database* [online]. MongoDB, Inc [cit. 2019-04-18]. Dostupné z: `https://www.mongodb.com/scale/most-popular-nosql-database`.
9. *MongoDB* [online]. MongoDB, Inc [cit. 2019-04-18]. Dostupné z: `https://www.mongodb.com/`.
10. SINGH, Ashish Nandan. *An intro to Webpack: what it is and how to use it* [online] [cit. 2019-04-22]. Dostupné z: `https://medium.freecodecamp.org/an-intro-to-webpack-what-it-is-and-how-to-use-it-8304ecdc3c60`.
11. *Node.js NPM* [online] [cit. 2019-04-22]. Dostupné z: `https://www.w3schools.com/nodejs/nodejs_npm.asp`.

12. FIELDING, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000.

13. *API - application program interface* [online] [cit. 2019-04-18]. Dostupné z: https://www.webopedia.com/TERM/A/API.html.

14. *AJAX Introduction* [online] [cit. 2019-04-18]. Dostupné z: https://www.w3schools.com/xml/ajax_intro.asp.

15. *ASP.NET - Single-Page Applications: Build Modern, Responsive Web Apps with ASP.NET* [online] [cit. 2019-04-20]. Dostupné z: https://msdn.microsoft.com/en-us/magazine/dn463786.aspx.

16. *Jest* [online] [cit. 2019-04-26]. Dostupné z: https://jestjs.io/.

17. *Testing - Overview* [online] [cit. 2019-04-26]. Dostupné z: https://2018.stateofjs.com/testing/overview/.

18. *Chai Assertion Library* [online] [cit. 2019-04-16]. Dostupné z: https://www.chaijs.com/.

19. *What is Selenium?* [online] [cit. 2019-04-26]. Dostupné z: https://www.seleniumhq.org/.

20. *Puppeteer* [online] [cit. 2019-04-17]. Dostupné z: https://github.com/GoogleChrome/puppeteer.

21. *Sinon-Chai* [online] [cit. 2019-05-07]. Dostupné z: https://docs.cypress.io/guides/references/bundled-tools.html#Sinon-Chai.

22. *Unit Testing* [online]. Software Testing Fundamentals [cit. 2019-04-12]. Dostupné z: http://softwaretestingfundamentals.com/unit-testing/.

23. *Integration Testing* [online]. Software Testing Fundamentals [cit. 2019-04-12]. Dostupné z: http://softwaretestingfundamentals.com/integration-testing/.

24. *supertest* [online] [cit. 2019-04-28]. Dostupné z: https://www.npmjs.com/package/supertest.

25. *Stubs and Mocks* [online] [cit. 2019-05-12]. Dostupné z: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff798400(v=pandp.10).

26. *Migrating from Mocha to Jest* [online] [cit. 2019-05-05]. Dostupné z: https://medium.com/airbnb-engineering/unlocking-test-performance-migrating-from-mocha-to-jest-2796c508ec50.

27. *Puppeteer* [online]. Puppeteer [cit. 2019-04-29]. Dostupné z: https://pptr.dev/#?product=Puppeteer&version=v1.15.0&show=api-pageselector.

28. *Class By* [online]. Selenium [cit. 2019-04-29]. Dostupné z: `https://seleniumhq.github.io/selenium/docs/api/javascript/module/selenium-webdriver/index_exports_By.html`.

29. *Status Code Definitions* [online] [cit. 2019-05-08]. Dostupné z: `https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html`.

30. FOWLER, Martin; FOEMMEL, Matthew. Continuous integration. *ThoughtWorks) http://www. thoughtworks. com/Continuous Integration. pdf.* 2006, roč. 122, s. 14.

31. *Travis CI* [online] [cit. 2019-05-05]. Dostupné z: `https://travis-ci.com/`.

32. *Heroku* [online]. Salesforce.com [cit. 2019-05-05]. Dostupné z: `https://www.heroku.com/`.

Dodatok A

Zoznam použitých skratiek

AJAX Asynchronous JavaScript And XML.

API Application programming interface

CI Continuous integration

CSS Cascading Style Sheets

HTML Hypertext Markup Language

REST Representational state transfer

SPA Single page application

PaaS Platform as a service

Dodatok **B**

Obsah priloženého DVD

```
readme.txt ..................................stručný popis obsahu DVD
├── src. ....................................zdrojové kódy implementácie
└── text ....................................text práce
    ├── thesis.tex ..................zdrojová forma práce vo formáte LaTeX
    └── thesis.pdf ..............................text práce vo formáte PDF
```