VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

WEBOVÝ SYSTÉM PRO VYTVÁŘENÍ MATEMATICKÝCH PŘÍKLADŮ

WEB SYSTEM FOR MATH TESTS CREATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **DÁVID NIKOLAS CZIROK**
AUTHOR

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

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Czirok Dávid N.**
Program: Informační technologie
Název: **Webový systém pro vytváření matematických příkladů**
Web System for Math Tests Creation
Kategorie: Uživatelská rozhraní

Zadání:

1. Nastudujte základy moderních webových frameworků (pro frontend i backend) a knihovny pro interpretaci matematických výrazů.
2. Navrhěte jazyk, který umožní zápis různých typů matematických příkladů. Cílem je tvorba šablon příkladů, které budou později automaticky doplněné hodnotami.
3. Implementujte interpretr navrhnutého jazyka a otestujte ho.
4. Implementujte jednoduché webové rozhraní, které umožní vytváření a zobrazování matematických příkladů. Ověřte přívětivost a jednoduchost vytváření příkladů.
5. Implementujte jednoduchý webový systém pro tvorbu testů z matematiky.
6. Diskutujte dosažené cíle a navrhěte směry dalšího vývoje.
7. Vyrobte A2 plakátek a cca 30 vteřinové video prezentující výsledky vaší práce.

Literatura:

- Tidwell et al.: Designing Interfaces: Patterns for Effective Interaction Design, O'Reilly, 2020
- Steve Krug: Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability, ISBN: 978-0321965516
- Steve Krug: Rocket Surgery Made Easy: The Do-It-Yourself Guide to Finding and Fixing Usability, ISBN: 978-0321657299
- Joel Marsh: UX for Beginners: A Crash Course in 100 Short Lessons, O'Reilly 2016
- Dále podle pokynů vedoucího

Pro udělení zápočtu za první semestr je požadováno:

- Body 1, 2, 3 a část bodu 4 ze zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Szőke Igor, Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 1. listopadu 2021

Abstrakt

Cieľom tejto práce bolo uľahčiť tvorbu a vyhodnocovanie matematických online testov automatizovaním procesu vytvárania ich príkladov a ich následného využitia pri zostavovaní testov. Zahrňovalo to návrh jazyka, ktorý umožní zápis šablón príkladov a interpretera, ktorý tento jazyk dokáže spracovať a generovať konkrétne príklady. Výsledok práce predstavuje jednoduchý webový systém, využívajúci tento jazyk a interpreter, umožňujúci učiteľom vytvárať príklady a testy, ktoré žiaci môžu vypĺňať. Klientská časť systému je implementovaná pomocou technológií z ekosystému Vue.js a knižnice Vuetify. Serverová časť je postavená na technológii Express.js s využítím knižnice Sequelize pre komunikáciu s MySQL databázou.

Abstract

This thesis aims to simplify the creation and evaluation of online math tests by automating the creation of tasks and their subsequent use in test building. It includes the proposal of a language intended for creating task templates and the implementation of an interpreter which processes it and generates concrete tasks. The final product is represented by a simple web system using this language and its interpreter, enabling teachers to create tasks and tests which students can take. The front-end of the application is built with technologies from the Vue.js ecosystem and the Vuetify library. The back-end uses the Express.js framework and the library Sequelize for communication with a MySQL database.

Kľúčové slová

webová aplikácia, matematické príklady, matematické testy, IS, JavaScript, Vue, Vuetify, SQL, Sequelize, Axios, API, Node, interpreter, jazyk

Keywords

web application, math tasks, math tests, IS, JavaScript, Vue, Vuetify, SQL, Sequelize, Axios, API, Node, interpreter, language

Citácia

CZIROK, Dávid Nikolas. *Webový systém pro vytváření matematických příkladů*. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Igor Szőke, Ph.D.

Webový systém pro vytváření matematických příkladů

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Igora Szőkeho, Ph.D. Uviedol som všetky zdroje a publikácie, z ktorých som čerpal.

.....
Dávid Nikolas Czirok
9. mája 2022

Poďakovanie

Ďakujem svojmu vedúcemu práce pánovi Ing. Igorovi Szőkemu, Ph.D. za pomoc pri vypracovávaní tejto práce, veľkú trpezlivosť a všetky poskytnuté znalosti.

Obsah

Kapitola 1

Úvod

Online testy sú v dnešnej dobe už neodmysliteľnou súčasťou výuky na školách. Učitelia si pomocou nich dokážu jednoduchšie a rýchlejšie overiť znalosti svojich študentov. Vyučujúci môžu test „rozdať“ naraz všetkým žiakom, nemusia lúštiť ich rukopis pri opravovaní a pomocou automatického vyhodnocovania si dokážu ušetriť veľa času a práce.

Digitalizácia testov prináša mnoho výhod pre vzdelávanie, no taktiež so sebou nesie aj hromadu problémov, či už zo strany vyučujúcich alebo žiakov. Jednou z hlavných výziev je tvorba samotných príkladov a testov.

Pri matematických príkladoch sa často musia manuálne zadávať rôzne hodnoty pre rovnaký typ príkladu, a tým pádom zmena formy testu učiteľovi nemusí priniesť skoro žiadny ušetrený čas. Učitelia musia byť schopní zvoliť si požadovanú formu odpovede na daný príklad a dať žiakom dostatočné množstvo rôznych príkladov, aby si mohli čo najlepšie precvičiť a pochopiť dané učivo. Výsledkom sú častokrát testy s identickými príkladmi pre každého žiaka, čo zvyšuje šancu, že pri vypracovávaní príkladu si žiaci odpovede navzájom iba odpíšu.

Cieľom tejto práce je navrhnúť jazyk umožňujúci zápis šablón príkladov a vytvoriť k nemu nástroj pre tvorbu testov z matematiky pre základné školy. Nástroj bude schopný generovať príklady s rôznymi hodnotami pre zadaný typ úlohy a z nich následne zostaviť test. Vytváranie príkladov nemusí byť nutne limitovaný iba pre matematiku, ale mohol by byť využitý aj v iných predmetoch.

V kapitole 2 prevediem analýzu problému a porovnanie podobných súčasných riešení a nástrojov. Návrh jazyka, ktorý umožní zápis šablón príkladov je v kapitole 3, kde popisujem jeho význam, zvolený prístup na základe problematiky a tvorbu jeho gramatiky. Kapitola 4 sa zaoberá následnou implementáciou a testovaním interpretera pre navrhnutý jazyk. Prehľad a porovnanie technológií pre tvorbu webových rozhraní a vybrané nástroje pre implementáciu rozhrania systému sú opísané v kapitole 5. Opis návrhu jednotlivých častí webového systému je možné nájsť v kapitole 6. V kapitole 7 opisujem následnú implementáciu systému na základe návrhu. Kapitola 8 popisuje priebežné testovanie systému, užívateľské testovanie výsledného systému, jeho výsledky a ďalší smer vývoja.

Kapitola 2

Analýza problému a porovnanie podobných riešení

Táto kapitola sa zaoberá analýzou problému a porovnávaním existujúcich riešení a nástrojov. Porovnávané boli rôzne platformy pre výučbu matematiky, webové stránky a aplikácie ponúkajúce možnosť vytvárania príkladov a testov z matematiky.

2.1 Porovnanie matematických platforiem

Prvým krokom v práci bolo porovnanie vybraných matematických platforiem pre výuku matematiky a webových stránok či aplikácií ponúkajúce vytváranie príkladov alebo testov. Pri porovnaní som sledoval nasledujúce vlastnosti:

- Dostupnosť – či je nástroj dostupný bezplatne alebo je nutné si zaň zaplatiť.
- Tvorba príkladov – či nástroj umožňuje vytváranie príkladov.
- Tvorba testov – či nástroj umožňuje vytváranie testov.
- Prispôsobenie – či je možné dané príklady a testy podrobnejšie modifikovať.
- Vypĺňovanie – či nástroj umožňuje vyriešiť vytvorené príklady v rámci svojho systému.
- Vyhodnocovanie – či nástroj automaticky opraví vyplnené testy alebo ich musí učiteľ manuálne skontrolovať.

2.1.1 Prodigy

Prodigy[1] je webová stránka, ktorá vyučuje matematiku vo forme turn-based RPG hry. Hráči, teda žiaci, tu majú možnosť bojovať proti rôznym príšerám, pri čom pred každým útokom musia vypočítať príklad. Obtiažnosť jednotlivých príkladov sa nastaví podľa zvoleného ročníka hneď pri registrácii a ďalej sa zvyšuje podľa levelu hráča a typu príšery.

Užitočnou funkcionalitou je možnosť nápovedy, respektívne vysvetlenia daného slova v príklade, ak by žiak nepochopil alebo zabudol jeho význam. Pri riešení príkladu je dostupná klávesnica, pre zadávanie čísiel a znakov, a dokonca aj kalkulačka pre pomocné výpočty.

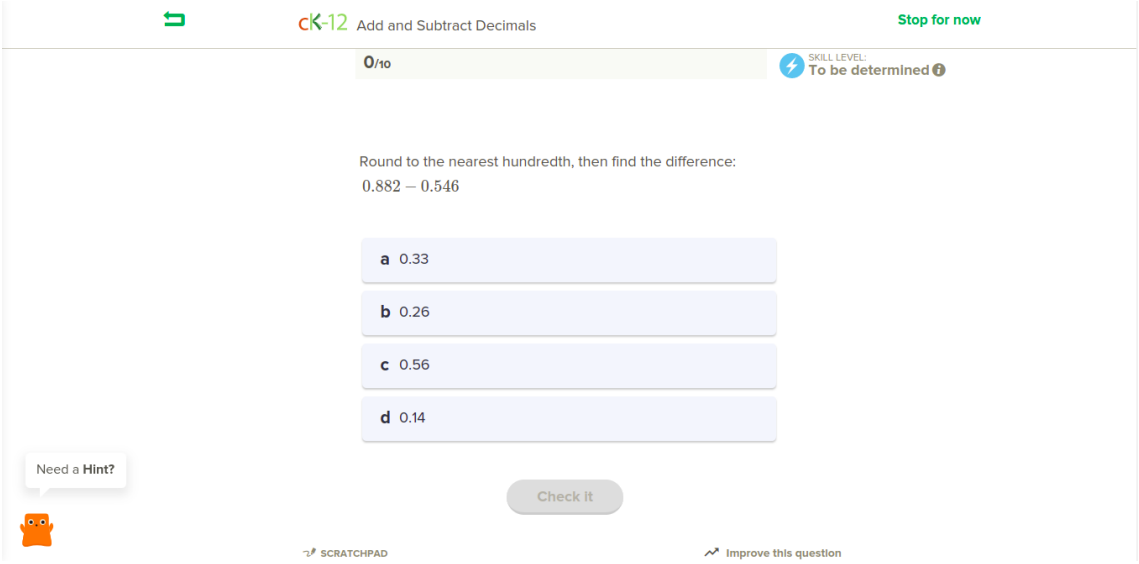
[1] Prodigy – Platforma pre výučbu matematiky, https://www.prodigygame.com/main-en/

Pre učiteľov tento nástroj ponúka rôzne štatistiky, ktoré prinášajú prehľad ako sa žiakom darí v jednotlivých príkladoch a študijných okruhoch.

2.1.2 ck12

Stránka ck12[2] ponúka výukové videá, články a príklady od matematiky, fyziky až po sociálne vedy. V rámci matematiky ponúka obsah členený podľa jednotlivých ročníkov a ďalej podľa tém, kde sa zameriava na pochopenie jednotlivých konceptov. Ku každej téme má video a následne ponúka možnosť testu. Ukážka 2.1 znázorňuje zadanie príkladu v aplikácii.

Príklady sú zaujímavé a kreatívne. Nástroj ponúka možnosť nápovedy vo forme videa, ktorý predchádza testu. Pri riešení príkladov je k dispozícii aj plocha na kreslenie, pre ktorú však nevidím veľké využitie. Obsah testov je dostupný iba pre prihlásených užívateľov.



Obr. 2.1: Ukážka zadania príkladu na počítanie s desatinnými číslami v aplikácii ck12. Prevzaté z: https://www.ck12.org/c/elementary-math-grade-4/add-and-subtract-decimals/?referrer=concept_details

2.1.3 Eductify

Slovenská stránka Eductify[3] ponúka teóriu a príklady, rozdelené podľa ročníkov alebo tém z osnovy pre výuku matematiky. Obsah je dostupný v niekoľkých jazykoch. Na stránke sa dajú precvičiť získané znalosti vo forme testov. Ku každej téme je dostpuný jeden test alebo si môžeme spustiť vlastný, do ktorého si vieme vybrať typy príkladov. Niektoré typy príkladov sú dostupné iba pre platenú verziu. Eductify je taktiež dostupná ako mobilná aplikácia, pre Android aj iOS. Ako výhodu by som uviedol jednoduché grafické užívateľské rozhranie, rozsiahly počet príkladov a možnosť mobilnej aplikácie.

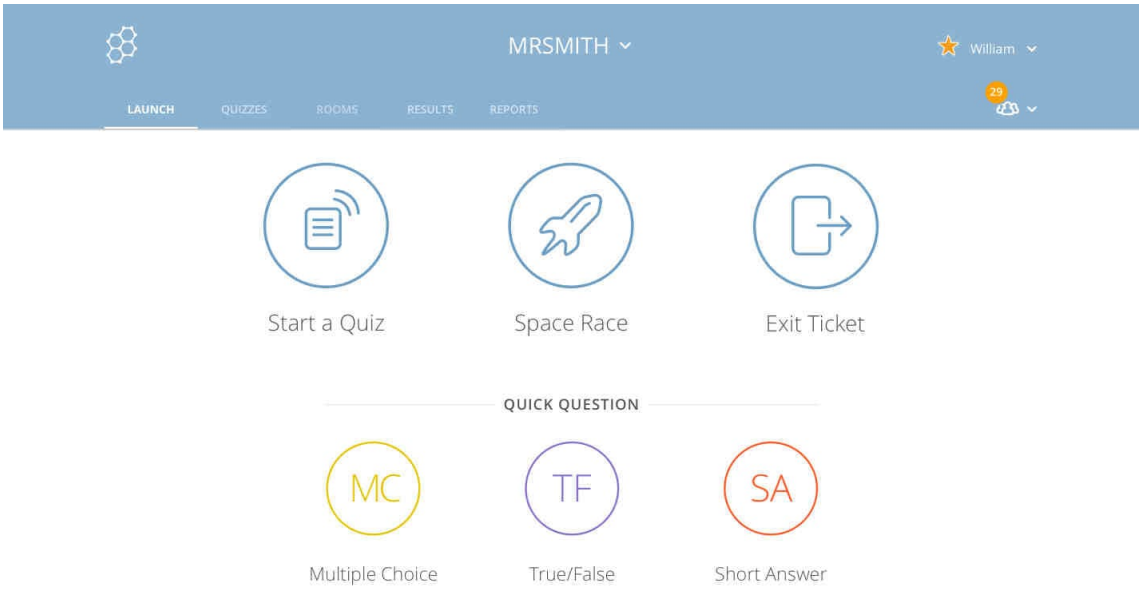
[2] ck12 – Vzdelávacia platforma, https://www.ck12.org/student/
[3] Eductify – Vzdelávacia platforma, https://www.eductify.com/sk/matematika-testy

2.1.4 The Math Learning Center

Na stránke[4] je môžené nájsť rozličné aplikácie pre grafické znázornenie zlomkov, geometrických útvarov, času alebo číselnej osi. Pomocou nich môžu žiaci lepšie spoznať a pochopiť základné koncepty jednotlivých operácií a tém. Táto stránka však konkrétne neponúka tvorbu príkladov ani žiadnych testov. Jedná sa iba o grafickú pomôcku pre výučbu niektorých konceptov matematiky.

2.1.5 Socrative

Socrative[5] pozostáva z dvoch častí – študentskej aplikácie Socrative Student a učiteľskej aplikácie Socrative Teacher. Pomocou Socrative Teacher môžu učitelia zadávať študentom otázky alebo kvízy, v ktorých si definujú vlastné úlohy pre študentov. Odpovede na otázky možu byť zadané v tvare áno/nie, výberom z niekoľkých možností alebo ako odpoveď v textovej podobe. Pomocou študentskej aplikácie žiaci vedia odpovedať na zadané otázky a kvízy. Učitelia dokážu sledovať výsledky v reálnom čase a majú prehľad o výkone žiakov. K otázkam sa môže pridávať aj vysvetľujúci popis či obrázok. Ukážka 2.2 zobrazuje užívateľské rozhranie aplikácie pre učiteľov.



Obr. 2.2: Ukážka učiteľskej aplikácie od Socrative. Prevzaté z: https://www.socrative.com/wp-content/uploads/2019/04/carousel-slide.jpg

Možnosť vytvárania vlastných otázok v testoch predstavuje asi najväčšiu výhodu tohto nástroja. Jednotlivé otázky sú ale viazané na konkrétny test a nedajú sa jednoducho znovu

[4]The Math Learning Center – Vzdelávacia platforma, https://www.mathlearningcenter.org/apps
[5]Socrative – Aplikácie na vzdelávanie, https://www.socrative.com/

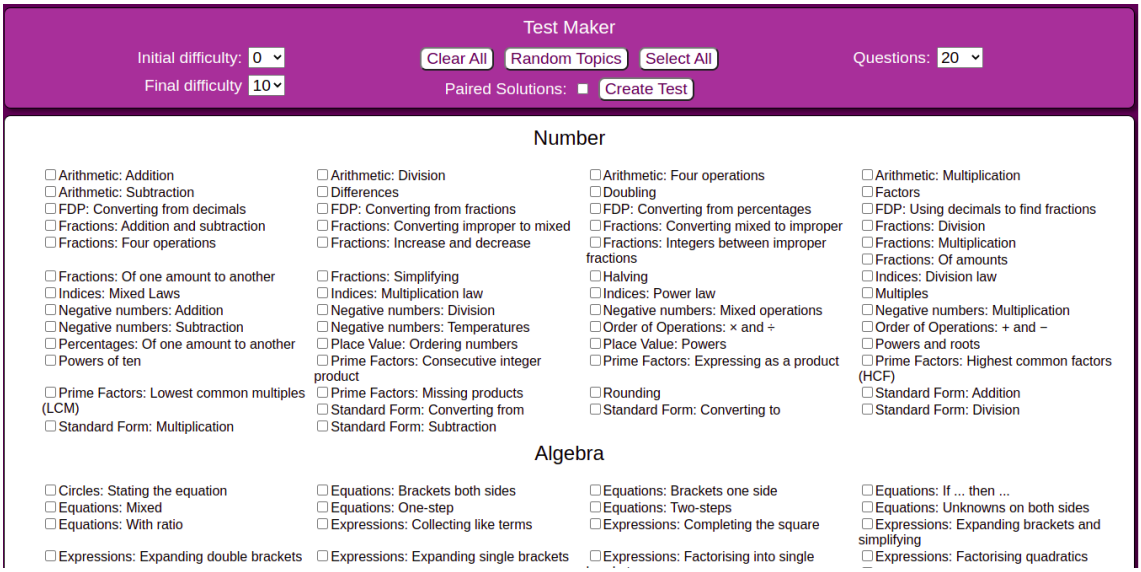
použiť v ďalších testoch. Okamžitá spätná väzba pre učiteľa a prehľadné grafické užívateľské rozhranie robí tento nástroj veľmi atraktívnym a praktickým. Aplikácia sa podobá mojej prvotnej predstave pre cieľ tejto práce.

2.1.6 Mathletics (3P Learning)

Platforma Mahletics[6] ponúka výuku matematiky v hravej forme. Obsah je rozdelený podľa tried, kde študent dostáva zoznam potrebných tém v danom ročníku. Ku každej téme je niekoľko videí, kde sú vysvetlené potrebné koncepty. Študent následne vypĺňa testy s príkladmi na precvičenie danej témy, pričom získava body za správne odpovede. Platforma ďalej ponúka možnosť súťažiť s inými žiakmi v krátkych hrách, kde študent dostáva príklady a za správne odpovede posúva svoje auto vpred. Cieľom je dopraviť svoje auto do cieľa čo najskôr, tým, že správne a v čo najkratšom čase odpovie na zadané otázky.

2.1.7 Mathsbot

Webová stránka[7], na ktorej je pomocou nástroja *Question Generator* môžné generovať príklady a pomocou nástroja *Test Maker* je možné vytvárať testy na základe vybraných typov príkladov z niekoľkých tém. Okrem výberu typov je možné vybrať počet otázok a obtiažnosť príkladov na škále od 0 do 10.



Obr. 2.3: Ukážka webovej stránky MathsBot a jej jednotlivých možností pri generovaní matematických testov. Prevzaté z: https://mathsbot.com/testMaker

Spomínané nástroje na stránke slúžia výhradne na generovanie príkladov a testov. Užívateľ nemôže zadávať odpovede na príklady. Stránka ponúka možnosť si vygenerované príklady a testy vytlačiť.

[6] Mathletics – Platforma pre výuku matematiky, https://www.mathletics.com/uk/
[7] Mathsbot – Nástroj na výučbu matematiky, https://mathsbot.com/

Zhrnutie porovnania platforiem

Každá skúmaná platforma má svoje prednosti aj obmedzenia. Kým nástroj platformy Eductify vyniká jednoduchým, prehľadným grafickým rozhraním, nástroje na stránke Mathsbot ponúkajú rozsiahle možnosti generovať príklady či testy. Aplikácia Socrative, v ktorej je učiteľ schopný vytvárať svoje vlastné príklady predstavuje výbornú pomôcku pre výuku. Tabuľka 2.1 znázorňuje prehľadné zhrnutie zistených informácií o jednotlivých platformách. Jednotlivé skúmané vlastnosti sú popísané v úvode kapitoly. Výsledkom je, že ani jedna z vyššie analyzovaných nástrojov neponúka možnosť vytvárať príklady spôsobom, akým by bol schopný výsledný nástroj tejto práce.

2.2 Podrobnejší popis problému a návrh riešenia

Z porovnania jednotlivých nástrojov v predošlej podkapitole vyplýva, že súčasné platformy pre výuku matematiky využívajú skôr predpripravené príklady. Nástroje s možnosťou vytvárať vlastné príklady ponúkajú možnosť vytvárania príkladov iba s pevne definovanými hodnotami a odpoveďami. Matematické testy, ktoré obsahujú takéto príklady s rovnakými hodnotami môžu viesť u študentov k jednoduchému odpisovaniu výsledkov. Počas vytvárania advekvátneho množstva príkladov s rôznymi hodnotami, v snahe tomuto odpisovaniu predísť a zároveň zaistiť dôkladné preverenie znalostí študentov, musí učiteľ stráviť priveľa času vymýšlaním hodnôt, kopírovaním textov a manuálnou zmenou jednotlivých hodnôt v príkladoch.

Aby mohol učiteľ efektívnejšie vytvárať príklady a nemusel pre každého žiaka zvlášť vymýšľať príklad s inými hodnotami alebo deliť test s príkladmi na skupiny, postačilo by mu vymyslieť šablónu na základe jedného konkrétneho príkladu, do ktorej by bolo možné automaticky generovať rôzne hodnoty. Jednotlivé generované hodnoty by bolo možné určiť definíciou alebo rozsahom. Definíciu môže predstavovať konkrétna hodnota alebo vzorec pre jej výpočet. Rozsah týchto generovaných hodnôt by sa musel dať jednoducho meniť, aby sa mohli šablóny znovu použiť pre viaceré ročníky. Pomocou zápisu bude možné určiť očakávaný správny výsledok ale a aj jednotlivé možné nesprávne výsledky. To by umožňovalo automatické vytváranie príkladov s otázkami s voliteľnými odpoveďami ale aj otázok vyžadujúce iba jednu konkrétnu odpoveď.

Zápis takejto šablóny bude vyžadovať návrh nového jazyka, ktorý bude schopný popísať jednotlivé časti príkladu a zachytiť všetky potrebné informácie o príklade. Ďalej bude nutné navrhnúť a implementovať nástroj, ktorý daný jazyk spracuje a bude schopný generovať konkrétne príklady na základe vytvorených šablón príkladov.

Ďalším krokom bude vytvoriť rozhranie, ktoré umožní jednoduchšie vytváranie týchto šablón a využije nástroj pre ich spracovanie. Toto rozhranie bude súčasťou systému, ktorý učiteľom dovolí vytvárať matematické príklady a vyskladať si z nich testy, ktoré môžu následne rozdať žiakom. Učiteľ bude mať v systéme prehľad o výsledkoch jednotlivých študentov z daného testu. Systém budú môcť využiť aj žiaci, ktorý pomocou neho budú môcť vytvorené testy vyplniť. Na vyplnenie budú mať vymedzený čas a výsledky budú mať ihneď po odovzdaní testu.

Názov	Vlastnosti	Hodnota	Výhody
Prodigy	Dostupnosť Tvorba príkladov Tvorba testov Prispôsobenie Vyplnenie Vyhodnocovanie	Platené Nie Nie Nie Áno Automatické	farebné a hravé, možnosť nápovedy, vysvetlivky výrazov,
ck12	Dostupnosť Tvorba príkladov Tvorba testov Prispôsobenie Vyplnenie Vyhodnocovanie	Bezplatné Nie Nie Nie Áno Automatické	pomocné videá, zamerané na koncepty, množstvo príkladov
Eductify	Dostupnosť Tvorba príkladov Tvorba testov Prispôsobenie Vyplnenie Vyhodnocovanie	Bezplatné Nie Nie Nie Áno Automatické	mobilná aplikácia, slovenský jazyk
The Math Learning Center	Dostupnosť Tvorba príkladov Tvorba testov Prispôsobenie Vyplnenie Vyhodnocovanie	Bezplatné Nie Nie Nie Nie Automatické	vytváranie interaktívnych ukážok, vhodné na pochopenie konceptov
Socrative	Dostupnosť Tvorba príkladov Tvorba testov Prispôsobenie Vyplnenie Vyhodnocovanie	Platené Áno Áno Nie Áno Automatické	mobilná aplikácia, aplikácia pre študenta, aplikácia pre učiteľa,
Mathletics (3P Learning)	Dostupnosť Tvorba príkladov Tvorba testov Prispôsobenie Vyplnenie Vyhodnocovanie	Platené Nie Nie Nie Áno Automatické	hravá forma, odmena za správny výsledok vo forme postupne vyskladaného obrázka
Mathsbot	Dostupnosť Tvorba príkladov Tvorba testov Prispôsobenie Vyplnenie Vyhodnocovanie	Bezplatné Áno Áno Nie Nie Manuálne	široký výber oblastí matematiky, možnosť vytlačiť vytvorené testy

Tabuľka 2.1: Prehľad skúmaných platforiem, ich vlastností a výhod

Kapitola 3

Návrh jazyka pre vytváranie príkladov

V tejto kapitole predstavím koncepty, potrebné pre pochopenie samotného návrhu jazyka. Popíšem zápis šablóny príkladu a postup jej tvorby. Ďalej bude nasledovať samotný návrh jazyka, ktorý umožní zápis matematických príkladov, podrobné vysvetlenie a odôvodnenie hlavných častí jeho zvoleného zápisu a formálny popis jeho gramatiky.

3.1 Princíp vytvárania príkladov

Hlavnou myšlienkou vytvárania matematických príkladov je nasledujúci postup:

1. Napísať zadanie matematického príkladu.
2. Definovať postup (vzorce) pre riešenie príkladu.
3. Previesť všetky požadované numerické hodnoty na premenné.
4. Určiť si rozsah pre jednotlivé premenné.
5. Každú premennú nahradiť náhodne vygenerovanou hodnotou z jej zadaného rozsahu.

Pre vytváranie príkladov teda vychádzam z predpokladu, že učiteľ vie, aký príklad chce žiakom zadať, pozná postup ako sa dopracovať k výsledku a jeho problém predstavuje vymyslieť dostatočné množstvo takýchto príkladov s rôznymi hodnotami.

Cieľom tejto časti práce je navrhnúť jazyk, ktorý umožní vytváranie šablón matematických príkladov. Šablónu príkladu predstavuje výsledok po 4. kroku (textový popis príkladu, definícia vzorcov potrebných na vyriešenie a otázky, na ktoré majú žiaci odpovedať). Na základe týchto šablón sa budú generovať konkrétne príklady.

3.2 Typy príkladov

Prvým krokom pred akýmkoľvek návrhom bola najprv štúdia učebnej osnovy pre výuku matematiky na základných školách v Českej republike[1]. Cieľom preštudovania osnovy bolo získať prehľad o jednotlivých typoch príkladov a vytýčiť vlastnosti, ktoré majú príklady spoločné, za účelom zadefinovania čo najobecnejšieho zápisu. Na základe tejto osnovy som

[1] Osnovy dostupné na: http://zs-ucebni-osnovy.blogspot.com/2011/03/21-matematika.html

si definoval typy príkladov, ktoré sa opakujú nezávisle od ročníka, alebo okruhu učiva a v nich hľadal spôsob, ako napísať vzorce potrebné na dopracovanie sa k ich správnemu výsledku.

3.2.1 Príklady s aritmetickými operáciami

Pre tento typ príkladu môžeme postup zo sekcie 3.1 aplikovať najzreteľnejšie. Pre príklady so základnými aritmetickými operáciami (+, -, *, /) je postup riešenia definovaný samotným zadaním príkladu. Obtiažnosť príkladu by zvyšoval čas potrebný na jeho zápis, identifikáciu a prevedenie konkrétnych hodnôt na premenné.

Pod tento typ príkladu môžeme zahrnúť matematické úlohy na sčitovanie, odčitovanie, násobenie, delenie, umocňovanie, odmocňovanie aj ich ľubovoľnú kombináciu. Taktiež zahrňujem aj špeciálne varianty ako je výpočet obvodu, obsahu, objemu a iných vlastností geometrických útvarov alebo prevod medzi jednotkami veličín SI[2]. Ďalej je možné do tejto sekcie zahrnúť aj príklady s desatinnými číslami, ktoré majú rovnaký postup pri vyhodnocovaní.

3.2.2 Slovné úlohy

Slovné úlohy predstavujú typ príkladu, kde nemusí byť hneď jednoznačné, ako sa dopracovať k výsledku, respektívne je možné dopracovať sa k výsledku viacero spôsobmi. Pre splnenie 2. bodu postupu zo sekcie 3.1 sa nemusí rátať s viacerými spôsobmi, stačí si vybrať jeden.

V ich podstate, slovné úlohy predstavujú príklady, ktoré vieme redukovať na postupnosť vzorcov. Pri prevedení číselných hodnôt na premenné je potreba najprv identifikovať v texte slovnej úlohy, ktoré informácie a hodnoty sú dôležité pre riešenie.

3.2.3 Dopĺňanie hodnôt, operátorov, zátvoriek

Patria sem príklady na dopĺňanie aritmetických operátorov a operandov s predom daným výsledkom, porovnávanie so vzťahmi väčší, menší alebo rovná sa. Postup riešenia príkladov na doplnenie operanda by sa dali odvodiť ako vzorec pre výpočet vyjadrením neznámeho operanda z príkladu.

Zložitejší problém predstavujú príklady na dopĺňanie operátorov a zátvoriek, pri ktorých nejde chýbajúce členy vyjadriť pomocou vzorca zo zadaného príkladu. Ak by sa pri dopĺňaní operátorov presne stanovil, ktorý operátor je ten chýbajúci a premennými by sa nahradili operandy, obmedzilo by sa vytváranie príkladu len na ten daný typ, s rovnakou odpoveďou. Pri dopĺňaní zátvoriek by sa dalo postupovať analogicky. Každá jedna zátvorka by bola definovaná ako samostatná odpoveď.

3.2.4 Zaokrúhľovanie

Zaokrúhľovanie je dôležitá súčasť matematiky, ktorú sa dá poňať ako samostatný príklad alebo ako požiadvka pre riešenie iných príkladov. Na jej definíciu budeme potrebovať funkciu, ktorá prevedie zaokrúhľovanie zadanej premennej. Môže slúžiť aj ako súčasť výrazu pre výpočet u iných pírkladoch.

[2]SI - Medzinárodná sústava jednotiek

3.2.5 Základné goniometrické funkcie

Pre vyššie ročníky sa už môžu zadávať aj príklady spojené s funkciami ako je sínus, kosínus alebo tangens. Podobne ako u zaokrúhľovaní by sa využili funkcie, ktoré dokážu zápis goniometrických funkcií spracovať a správne vyhodnotiť po dosadení premenných.

3.3 Model šablóny pre zápis príkladu

Účelom výpisu jednotlivých typov príkladov bolo zistiť ich spoločné vlastnoti a identifkovať problémy, ktoré by mohli nastať pri ich spracovaní. Na základe zistených znalostí vytvoriť čo najobecnejšiu štruktúru pre zápis príkladov. Z uvedených typov príkladov v sekcii 3.2 som zostavil model šablóny pre zápis matematického príkladu.

V zápise používam dva kľúčové pojmy. *Premenná* označuje hodnotu, ktorá bude vygenerovaná z daného rozsahu alebo vypočítaná na základe definície, kde je možné použiť aj iné premenné. *Odpoveď* predstavuje hodnotu, ktorá je riešením príkladu. Pred popisom samotného modelu je potrebné si tieto pojmy detailnejšie popísať.

3.3.1 Premenná

Premenná predstavuje vygenerovanú alebo vypočítanú hodnotu, ktorá sa ďalej využije v príklade. Vzorec, ktorý definuje danú premennú je možné priradiť v časti definície. Premennú tvorí reťazec začínajúci znakom `$` nasledovaný ľubovolným počtom alfanumerických znakov, pomlčiek alebo podtržníkov. Regulárny výraz pre premennú vyzerá nasledovne:

`^\$([a-zA-Z0-9_])+` (Príklady: `$x, $y1, $prepona, $1_a, $cb`)

3.3.2 Odpoveď

Odpoveď v zápise predstavuje miesto, kde sa bude po spracovaní zápisu očakávať riešenie zadané žiakom. V časti definície je možné priradiť k odpovedi vzorce pre výpočet správneho, ale aj vzorce pre výpočet nesprávneho riešenia. Odpoveď predstavuje reťazec, ktorý začína so znakom `@`. Za ním môže taktiež ako u premennej nasledovať ľubovoľný počet alfanumerických znakov. Regulárny výraz pre odpoveď:

`^@([a-zA-Z0-9_])+` (Príklady: `@res, @result_1, @1, @1a, $res_1A`)

3.3.3 Typy odpovedí

Na matematické príklady, pre ktoré vytváram daný jazyk môže žiak odpovedať dvoma spôsobmi. Prvou možnosťou je otvorená odpoveď. Žiak musí samostatne odpovedať na otázku v zadaní vyplnením príslušného poľa. Druhou možnosťou je výber správnej odpovede z niekoľkých možností.

3.4 Časti šablóny

Navrhnutý model šablóny pre zápis príkladov pozostáva zo štyroch častí, ktoré sú oddeľované kľúčovými slovami vystihujúce danú časť šablóny. Kľúčové slová v šablóne sa vyznačujú reťazcom začínajúcim so znakom #. Ďalej nasleduje popis jednotlivých častí.

3.4.1 Text

Úvodný text k príkladu, ktorý predstavuje jeho zadanie alebo popisuje slovnú úlohu. Začína sa za kľúčovým slovom *#text*, po ktorom môže nasledovať ľubovolný počet alfanumerických znakov. Môžeme sem uviesť premenné, ktoré by sme chceli nahradiť konkrétnym číslom po vytvorení príkladu. Šablóna musí povinne obsahovať iba kľúčové slovo, za ním nie je nutné písať nič, ak si to zadanie nevyžaduje. Konkrétny príklad pre túto časť:

```
#text
V pravouhlom trojuholníku ABC sú dané odvesny a = $a cm, b = $b cm.
```

3.4.2 Otázka

Predstavuje otázku k danému príkladu. Vyznačená kľúčovým slovom *#question*, za ktorým môže nasledovať ľubovolný počet alfanumerických znakov, premenných a odpovedí. Pomocou premenných, ktoré sa neskôr nahradia konkrétnymi hodnotami, vieme vytvoriť definíciu otázky. Tam, kde sú použité odpovede sa zobrazí miesto na vyplnenie alebo otáznik, v závislosti na typu odpovede akú požadujeme. K jednému príkladu môže byť viacero otázok, v tom prípade by sa znova zopakovalo kľúčové slovo a za ním potrebné znaky, premenné a odpovede. Otázok môže byť neobmedzené množstvo. Časť pre otázky môže vyzerať nasledovne:

```
#question
Vypočítajte preponu c: @res_c
#question
Vypočítajte výšku na prepone vc: @vc
```

3.4.3 Definície

Za kľúčovým slovom *#definitions* musia nasledovať deklarácie a definície premenných a odpovedí pre daný príklad. Deklarácia premennej predstavuje zápis jej názvu. Definícia priraďuje daný vzorec, ktorý sa použije pre výpočet premennej. Každá premenná alebo odpoveď musí byť deklarovaná či definovaná na samostatnom riadku.

Pre každú premennú, ktorá sa vyskytuje v sekcii *#text* alebo *#question* musí existovať príslušná deklarácia alebo definícia. Ak je premenná iba deklarovaná, musí jej prislúchať v sekcii *#ranges* definovaný rozsah. V prípade definície premennej, nie je nutné určovať jej rozsah.

Ak sa jedná o odpoveď, tá musí mať vyplnený atribút *correct*. Bez vyplnenia atribútu *incorrect*, je možné vybrať iba možnosť textovej odpovede.

Formát jednotlivých možností v sekcii definície:

```
#definitions
$a              // deklarácia premennej
$b = $a + 3     // definícia premennej

// priradenie správneho vzorca na výpočet
@c.correct = [ sqrt(pow($b,2) + pow($b,2)) ]
```

```
// priradenie nesprávnych vzorcov na výpočet
@c.incorrect = [
    sqrt(pow($a,2) + pow($b,2)) - 1,
    sqrt(pow($a,2) + pow($b+1,2)),
    pow($a,2) + pow($b,2)
]
```

3.4.4 Rozsah

Za kľúčovým slovom *#ranges* nasledujú definície rozsahov pre jednotlivé zvolené premenné. Rozsah premennej je definovaný funkciou *range*, ktorá berie tri argumenty. Pracuje so začiatkom rozsahu, koncom rozsahu a krokom. Táto funkcia udáva v akom rozsahu sa majú generovať čísla pre dané premenné. Pri jednoduchších príkladoch, stačí zmeniť rozsah na väčšie čísla a to umožní jednoducho škálovať obtiažnosť príkladov bez nutnosti opakovanej definície celého príkladu. Príklad zápisu:

```
#ranges
$x = range(1, 25, 1)    // 1, 6, 3, 17, 9, 21, ...
$y = range(0, 10, 0.01)  // 0.21, 5.34, 6.63, ...
$z = range(-20, 20, 2)     // 4, 18, -2, ...
```

3.4.5 Správne odpovede

Pre každý príklad existuje postupnosť vzorcov, ktorý nás skôr či neskôr dovedie k správnemu výsledku. U niektorých príkladoch stačí jeden vzorec pre výpočet správneho výsledku. Pri tažších príkladoch, ktoré vyžadujú medzivýsledky, môžeme ich hodnoty ukladať do premenných. Tie sa dajú použiť pri finálnej definícii vzorca pre správne riešenie. Premenné, ktoré sú už definované nevyžadujú mať stanovený rozsah.

Vzorec pre správne riešenie sa udáva ako pole. Je to z dôvodu, ak by nastalo, že daný príklad bude mať viacero správnych riešení. To môže nastať napríklad v prípade kvadratickej rovnice, ktorá má dva korene.

```
#definitions
...
// priradenie viacerých správnych riešení
@res2.correct = [
    -$b*sqrt($D) / 2*$a,
    -$b*(-sqrt($D)) / 2*$a
]
```

Správnou odpoveďou nemusí byť nutne vzorec, môže to byť aj konštanta, operátor alebo zátvorka. V prípade príkladu pre doplnenie zátvoriek bude zápis správnej odpovede vyzerať nasledovne:

```
@r1.correct = [ '(' ]
@r2.correct = [ ')' ]
```

3.4.6 Nesprávne odpovede

Ak si učiteľ vyberie možnosť odpovede formou výberu z viacero možností, máme pre generovanie nesprávnych odpovedí taktiež možnosť definovať vzorce. Tie však budú mať nejakú chybu, ku ktorej sa žiak pri rátaní mohol dopustiť. Pre väčšiu flexibilitu je možné zadávať ľubovoľné množstvo nesprávnych odpovedí. Jednotlivé nesprávne odpovede môžeme zadať aj ako konštanty, funkcie či iné hodnoty, rovnako ako v prípade správnych riešení. Pre ukážku si môžeme zobrať jednoduchý príklad na sčítanie dvoch čísiel:

```
#definitions
...
@result.incorrect = [
    $x1 + $x2 - 1,
    $x1 + $x2 + 1,
    $x1 - $x2
]
```

3.4.7 Ukážka šablóny zápisu príkladu

Nasledujúca šablóna ukazuje kompletný zápis príkladu[3]:

Vypočítajte strany pravouhlého trojuholníka, ak viete, že má obsah S = 180 m2 a jedna jeho odvesna je o 31 m dlhšia ako druhá.

```
#text Vypočítajte strany pravouhlého trojuholníika, ak viete, že jeho
obsah je $S m^2 a~jedna jeho odvesna je o 31m dlhšia ako tá druhá.

#question odvesna = @odvesna1
#question dlhšia odvesna = @odvesna2
#question prepona = @prepona

#definitions
$x
$y = $x + 31
$z = sqrt(pow($x,2) + pow($y,2))
$S = ($x*$y) / 2

@odvesna1.correct = [ $x ]
@odvesna2.correct = [ $y ]
@prepona.correct  = [ $z ]

#ranges
$x = range(1, 10, 1)
```

[3]Príklad dostupný na príklady.eu

3.5 Popis jazyka

Na základe šablóny viem, že pre riešenie matematických príkladov je potreba v navrhovanom jazyku umožniť tvorbu matematických výrazov. Taktiež je potrebné umožniť použitie zátvoriek, definíciu vzorcov pomocou aritmetických operácií, ale aj pokročilejších operácií ako je odmocňovanie a iné pomocné funkcie. Ďalej je potrebné zaistiť funkcie pre generovanie náhodných čísiel v danom rozsahu s daným krokom.

Cieľom nie je vytvoriť celý programovací jazyk, ale jazyk, ktorý je schopný popísať jednotlivé časti príkladu a zároveň umožní zápis vzorcov, matematických operácií a funkcií. Šablóna bola vytváraná s myšlienkou, že bude predstavovať zápis príkladu v navrhovanom jazyku. Jej hlavné časti sú popísané v sekcii 3.4.

Pre zápis príkladu v mojom jazyku som bral inšpiráciu z existujúcich známych programovacích jazykov. Konkrétne, spôsob zápisu premennej je prebratý z jazyka PHP[4].

3.6 Gramatika jazyka

Pre kontrolu a zjednodušenie následnej implementácie je vhodné si formálne definovať vytvorený jazyk. Pre popis môžeme použiť regulárne výrazy, formálne gramatiky či automaty. Daný jazyk je možné popísať pomocou bezkontextovej *LL(1)* gramatiky.

`txt var ans num eol ( ) op eq range [ ] corr incorr , t q d r EXP`.

- **txt** - Ľubovoľný reťazec tvorený ascii znakmi.
- **var** - Označuje *premennú*.
- **ans** - Označuje *odpoveď*.
- **num** - Predstavuje numerickú hodnotu.
- **eol** - Označuje znak `'\n'`.
- **op** - Označuje znaky `'+', '-', '*', '/', '^', '<', '>', '%'`.
- **eq** - Označuje znak `'='`.
- **range**, **corr**, **incorr** - Označujú reťazce `range, .correct a~.incorrect` respektívne.
- **t**, **q**, **d**, **r** - Označujú kľúčové slová `#text, #questions, #definitions a #ranges`.
- **EXP** - Predstavuje ľubovoľný matematický výraz (napr. `2 + 2, 4^2, sqrt(4)`).

3.6.1 Neterminály v gramatike

- **P** – Počiatočný neterminál, ktorý reprezentuje začiatok programu. Po ňom nasledujú štyri hlavné časti zápisu, ktoré sú povinné pre každý validný zápis šablóny príkladu:
  - **T** – Časť textového zadania príkladu, predstavuje kľúčové slovo #text.
  - **Q** – Časť pre otázky daného zadania. Pravidlo Qn umožňuje zadávať viacero otázok. Označuje kľúčové slovo #question.

[4]PHP: Hypertext Preprocessor - https://www.php.net/

  - **D** – Časť pre definície premenných a odpovedí, kľúčové slovo #definitions.
  - **R** – Časť pre definíciu rozsahu jednotlivých premenných (a odpovedí), začína kľúčovým slovom #ranges.
- **W** – Predstavuje text zadania alebo otázky.
- **A** – Predstavuje operáciu priradenia.
- **AV** – Označuje definíciu premennej.
- **AA** – Označuje definíciu odpovede a s ňou sú spojené:
  - **ARR0** – Označuje pole výrazov, ktoré sú priradené k odpovedi.
  - **ARR1** – Prvý výraz v zozname.
  - **ARR2** – Ľubovolný nasledujúci výraz v zozname.

3.6.2 Pravdilá gramatiky

$P \rightarrow T\ Q\ D\ R$

$T \rightarrow t\ W$

$Q \rightarrow q\ W\ Qn$

$Qn \rightarrow q\ W \mid \varepsilon$

$D \rightarrow d\ A$

$R \rightarrow r\ S$

$W \rightarrow txt\ W \mid num\ W \mid var\ W \mid ans\ W \mid eq\ W \mid op\ W \mid (\ W \mid)\ W \mid eol\ W \mid \varepsilon$

$A \rightarrow var\ AV\ eol\ A \mid ans\ AA\ A \mid \varepsilon$

$AV \rightarrow eq\ EXP \mid EXP$

$AA \rightarrow corr\ eq\ ARR0 \mid incorr\ eq\ ARR0$

$ARR0 \rightarrow [\ ARR1\ ]\ eol \mid \varepsilon$

$ARR1 \rightarrow EXP\ ARR2$

$ARR2 \rightarrow ,\ ARR1 \mid \varepsilon$

$S \rightarrow var\ eq\ range\ \ (\ num\ ,\ num\ ,\ num\ )\ eol\ S \mid \varepsilon$

Kapitola 4

Spracovanie zápisu

Táto kapitola sa zaoberá interpretáciou jazyka pre zápis príkladov. Popisuje výber programovacieho jazyka, nástrojov, spôsob implementácie jednoduchého interpreteru, jednotlivé jeho časti, zvolené metódy spracovania a ukladania získaných dát po interpretácii zápisu.

4.1 Technológie a knižnice pre implementáciu

Pre implementáciu tejto časti práce som si zvolil jazyk JavaScript. Pôvodne som rozmýšľal nad jazykom PHP a nástrojom Nette, či nástrojom Flutter, avšak ich krivka učenia sa mi zdala byť príliš časovo náročná.

4.1.1 Javascript

Javascript predstavuje najrozšírenejší programovací jazyk pre vývoj klientských webových aplikácií. Syntakticky je podobný jazyku C, čo môže prispieť k rýchlejšiemu pochopeniu jeho kódu. Tam však jeho podobnosť s jazykom C končí. Javascript je univerzálny, interpretovaný programovací jazyk s objektovo orientovanými prvkami a predstavuje jazyk so slabou typovou kontrolou [6].

Z dôvodu, že je najviac využívaný v prehliadači, jeho základ je rozšírený o objekty a funkcie, ktoré umožňujú interakciu s užívateľom a objektmi DOM[1], reprezentujúce obsah webovej stránky.

Formálne nazývaný aj ECMAScript, Javascript je aktívne vyvíjaný jazyk, ktorý je vylepšovaný a dopĺňaný funkcionalitou s každou jeho iteráciou. Jeho najväčšími výhodami sú obrovský výber knižníc a frameworkov doplňujúcich jazyk o ďalšiu funkcionalitu. Výhodou je aj široká komunita ľudí, ktorí s ním pracuje.

4.1.2 Knižnice pre prácu s matematickými výrazmi

V základnej implementácií JavaScriptu máme dostupný objekt Math[2], ktorý ponúka užitočné matematické konštanty a niekoľko základných matematických funkcií. JavaScript je jazyk zameraný primárne na vytváranie logiky pre webové stránky. Neponúka žiadnu vstavanú funkcionalitu pre spracovanie matematických výrazov.

V snahe rozšíriť možnosti jazyka a doplniť tak tieto nedostatky, vznikla rada knižníc, ktorých cieľom je pridať rozšírené výpočetné možnosti v oblasti matematiky. Pre prácu s ma-

[1] DOM - Document Object Model

[2] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math

ticami a výpočty s vektormi je dostupná knižnica ndarray[3] alebo Sylvester[4]. Complex.js[5] a Numbers.js[6] podporujú výpočty s komplexnými číslami. Pre výpočet diferenciálov, integrálov, zjednodušenie výrazov a prácu s veličinami je dostupná knižnica Algebrite[7] alebo algebra.js[8]. Algebra.js umožňuje výpočet algebraických rovníc. Spracovanie a vyhodnocovanie výrazov je možné pomocou knižníc expr–eval[9] alebo jsep[10] [4].

V nasledujúcich podsekciách opíšem knižnice, ktoré som sa rozhodol využiť pri implementácii interpretera.

4.1.3 math.js

Math.js[11] predstavuje viacúčelovú knižnicu pre matematické výpočty. Jej snahou bolo zjednotiť vyššie uvedené knižnice. Jednotlivé knižnice ponúkajú exceletnú podporu pre danú oblasť matematiky, pre ktorú sú vytvorené. Slúžia však ako samostatné riešenia, bez možnosti spolupráce s ostatnými knižnicami.

Knižnica math.js ponúka integrované prostredie pre prácu s reálnymi aj komplexnými číslami, zlomkami, maticami, veličinami a algebraickými výrazmi pod jednou syntaxou. Podporuje zjednodušovanie, ale aj vyhodnocovanie výrazov.

Pre zápis matematických operácií, ako napríklad výber rozsahu, indexovanie a transponovanie matice, prevod či aritmetické operácie s veličinami, ale aj výpočty s komplexnými číslami, ponúka rozšírenú a úspornejšiu syntax oproti základnému Math modulu JavaScriptu.

Túto syntax spracováva vstavaný syntaktický analyzátor výrazov. Pri analýze výrazu sa vytvára abstraktný syntaktický strom, ktorý reprezentuje daný výraz. Ten je potrebný skompilovať do natívneho JavaScript kódu a následne je možné vyhodnocovať výraz s konkrétnymi hodnotami. Demonštrácia vyhodnotenia jednoduchého výrazu pomocou knižnice math.js [4]:

```
var tree = math.parse('3x ^ 2 + 5');  // abstraktný syntaktický strom
var compiled = tree.compile();        // natívny JavaScript kód
var result = compiled.eval( {x: 4} ); // 53
```

Ukážka algebraických operácií pomocou transformácie abstraktného syntaktického stromu (zjednodušenie a derivácia) [4]:

```
var tree = math.parse('x * x^2 ');
math.simplify(tree);    // x ^ 3
math.derivative(tree);  // 3 * x ^ 2
```

[3] ndarray - https://www.npmjs.com/package/ndarray
[4] Sylvester - http://sylvester.jcoglan.com/
[5] Complex.js - https://github.com/infusion/Complex.js/
[6] Numbers.js - http://numbers.github.io/
[7] Algebrite - http://algebrite.org/
[8] algebra.js - https://algebra.js.org/
[9] expr-eval - https://www.npmjs.com/package/expr-eval
[10] jsep - https://ericsmekens.github.io/jsep/
[11] mathjs - https://mathjs.org/

4.2 Interpreter

Úlohou kompilátorov a interpreterov je preklad programu napísaného v danom vysokoúrovňovom programovacom jazyku.

Kompilátor zvyčajne prekladá zdrojový kód programu do jazyka Assembler alebo na nízkoúrovňový strojový kód daného systému. Vstupom je zdrojový kód programu zapísaný v textovom súbore, ktorý je preložený na objektový (cieľový) kód. Po preklade sa tento objektový kód ukladá ako súbor. V prípade, že program tvorí viacero súborov so zdrojovým kódom, každý je najprv uložený ako zvlášť súbor s objektovým kódom. Tieto súbory sú následne pomocou spojovacieho programu (*linker*) prepojené do jedného objektového programu, ktorý je možné načítať do pamäte a spustiť [11].

Interpreter, na rozdiel od kompilátora, súbor so zdrojovým kódom neprekladá ale číta ho riadok po riadku a priamo ho vykonáva. Jeho výhodou je, že zdrojový kód programu je možné spustiť nezávisle od platformy, pretože nevytvára objektový kód špecifický pre danú platformu. Ďalším rozdieľom medzi interpretom a kompilátorom je detekcia chýb pri behu programu. Po vytvorení objektového kódu pomocou kompilátoru, je program samostatne spustený. Pri behovej chybe nám môže posktynúť iba informáciu o tom, že program zlyhal. Zistenie chyby v kóde teda závisí na schopnostiach programátora. Na rozdiel nám interpreter dokáže špecifikovať konkrétny riadok, na ktorom sa chyba vyskytla, či dokonca presnejšie určiť dôvod zlyhania [11].

V závislosti na interpretovanom jazyku môže interpretácia prebiehať na troch úrovniach. Priama interpretácia zdrojového kódu predstavuje výpočtovo najnáročnejšiu variantu. Pri opakovanom volaní funkcie sa musí každý riadok kódu pred vykonaním znova spracovať. Interpretáciou medzikódu sa zbavíme tohto problému avšak treba rátať s neskorším spustením aplikácie z dôvodu prvotného prekladu na medzikód [20].

4.2.1 Stavba interpreteru

Preklad programovacieho jazyka, teda vývoj kompilátoru alebo interpreteru, predstavuje nemalú technickú výzvu. Obecne sa vývoj prekladača rozdeľuje do niekoľkých logických modulov, ktoré navzájom spolupracujú. Kompilátor aj interpreter sú tvorené lexikálnym, syntaktickým a semantickým analyzátorom. Tieto komponenty predstavujú spoločný základ pre preklad akéhokoľvek jazyka [11].

4.2.2 Lexikálny analyzátor

Prvý modul potrebný pre preklad je lexikálny analyzátor. Lexikálny analyzátor číta zdrojový kód daného jazyka a rozdeľuje ho na lexikálne symboly. Tie môžu reprezentovať literály, indentifikátory, operátory, komentáre, klúčové slová a pod. Kontroluje či zdrojový kód obsahuje iba povolené znaky a správne postupnosti znakov. Lexikálny analyzátor sa často nazýva aj *skener*. Jeho úlohou je vlastne preskenovať celý dokument so zdrojovým kódom a rozdeliť ho na jednotlivé lexikálne symboly. Tieto symboly sa nazývajú *tokeny*.

Vstupom lexikálneho analyzátora je súbor so zdrojovým kódom. Výstupom je postupnosť tokenov, ktorá je predaná ďalej do ďalšieho modulu. Táto postupnosť tokenov môže byť posielaná po jednom na vyžiadanie, alebo po spracovaní celého zdrojového dokumentu, ako pole tokenov [20].

4.2.3 Syntaktický a sémantický analyzátor

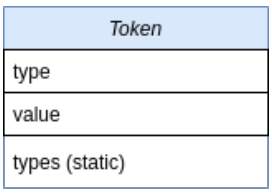
Úlohou syntaktického analyzátora je kontrola či súbor so zdrojovým kódom obsahuje povolené syntaktické konštrukcie – teda správne postupnosti tokenov. Správnosť postupnosti tokenov kontroluje na základe definovanej gramatiky pre daný jazyk. Tokeny prevádza do štruktúr, ktoré reprezentujú syntaktickú stavbu programu. Najvyužívanejším sú stromové štruktúry, ktoré sa nazývajú abstraktné syntaktické stromy. Sémantický analyzátor je zodpovedný napríklad za typovú kontrolu alebo kontrolu deklarácií [20].

Tieto dva analyzátory sú často spojené pod jedným modulom, ktorý sa nazýva *parser*. Jeho vstupom je postupnosť tokenov a výstupom je štruktúra, ktorá reprezentuje daný program. Táto štruktúra sa v ďalších moduloch prevádza na medzikód. Pri kompilátoroch za syntaktickou a sémantickou analýzou nasleduje ďalší modul pre optimalizáciu medzikódu. Finálny modul predstavuje generátor cieľového kódu [20]. Pri interprete sa nasledujúci modul nazýva *exekútor*, ktorý vyhodnocuje a prevádza medzikód [11].

Vytvorený jazyk pre zápis príkladov je taktiež potrebné interpretovať, aby sme mohli so zápisom príkladu ďalej pracovať. Interpreter bude v tomto prípade predstavovať program, ktorý prevedie matematický zápis príkladu do štruktúr jazyka JavaScript, ktorý môžeme ďalej spracovať.

4.3 Implementácia interpretera

Interpreter jazyka sa skladá z dvoch častí, ktoré navzájom spolupracujú. Prvá časť je lexikálny analyzátor, ktorý najprv spracuje celý zadaný zápis a vráti pole tokenov. Tokeny sú reprezentované objektom triedy `Token`. Trieda `Token` definuje nasledujúcu jednoduchú štruktúru:



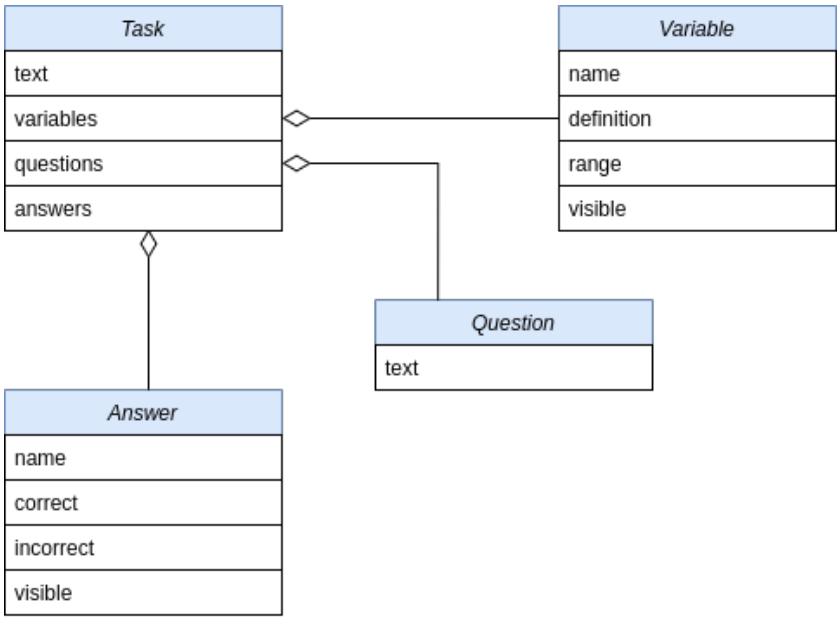
Obr. 4.1: Štruktúra objektu *Token*

Typ jednotlivých tokenov som si definoval podľa terminálov gramatiky, ktoré sú popísané v sekcii 3.6. Trieda `Token`, znázornená na ukážka 4.1, v sebe obsahuje objekt `types`, v ktorom sú definované jednotlivé typy tokenov a prislúchajúce regulárne výrazy alebo hodnoty, ktoré reprezentujú. Samotné spracovanie zdrojového textu je založené na stavovom automate. Základom je cyklus, ktorý prechádza text zápisu znak po znaku a pomocou konštrukcie switch, triedi jednotlivé skupiny znakov do tokenov daného typu. Lexikálny analyzátor spracováva aj jednotlivé operátory v neterminál `EXP`, ktoré sú následne reprezentované ako reťazce. Lexikálnu analýzu reprezentuje funkcia *tokenize* implementovaná v súbore `tokenizer.js`.

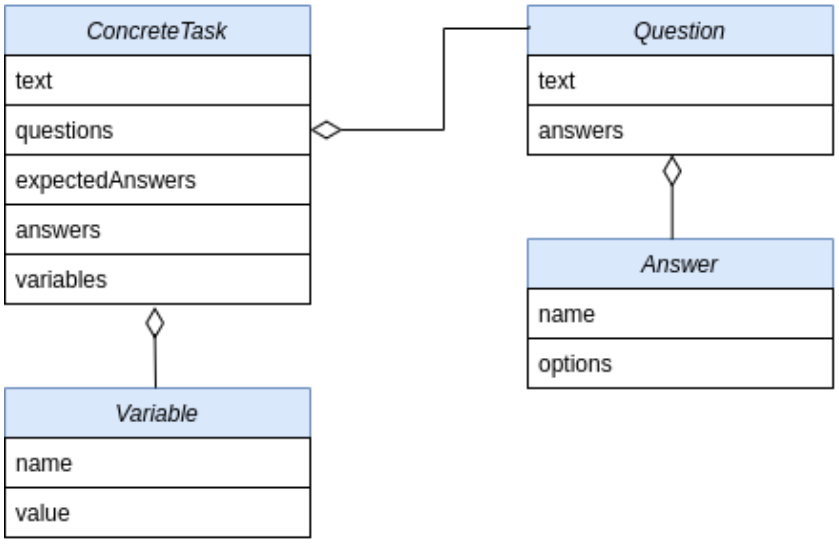
Syntaktický analyzátor v súbore `taskHandler.js` predstavuje funkciu *parseMathLang*, ktorá na základe zoznamu tokenov z lexikálnej analýzy (výsledok funkcie *tokenize*) prevedie zápis príkladu na príslušné štruktúry. Správnu postupnosť tokenov definuje gramatika opisovaná v sekcii 3.6. Princíp spracovania spočíva v postupnom čítaní tokenov a kontroly tejto postupnosti. Každá časť príkladu, ktorá bola definovaná v sekcii 3.4 sa spracováva v samostatnej funkcii, po ktorej sa zakaždým vyhodnocuje či nenastala chyba. Toto modulárne

spracovanie umožňuje previesť dodatočné zmeny v zápise a spracovaní jednotlivých častí zápisu bez toho, aby sa ovplyvnila funkcionalita ostatných častí. Pri chybe v spracovaní sa vyhadzuje výnimka s príslušnou chybovou správou.

Výsledkom po syntaktickej analýze je dátová štruktúra typu `Task` znázornená na ukážke 4.2. Obsahuje text príkladu a tri ďalšie zoznamy. Štruktúru premennej definuje trieda `Variable`, ktorá v sebe obsahuje atribúty pre identifikátor premennej, definíciu, rozsah generovaných hodnôt. Trieda `Answer` predstavuje štuktúru pre odpoveď. Jej atribúty tvorí indentifikátor, pole pre správne odpovede a pole pre nesprávne odpovede. V triede `Question` mám atribúty definujúce text danej otázky a pole príslušných odpovedí.



(a) Dátová štruktúra *Task*



(b) Dátová štruktúra *ConcreteTask*

Obr. 4.2: Ukážka dátových štruktúr pre spracovanie zápisu príkladu

V štruktúre je uložený celý zápis príkladu a na základe nej sa tvorí konkrétny príklad a to nasledujúcim spôsobom:

- Pre všetky premenné, ktoré majú prázdny atribút definície, vygeneruj náhodné číslo v rozsahu definovanom v zápise.
- Pre všetky premenné vyčísli hodnoty na základe ich definovaného vzorca pre výpočet.
- Nahraď každý výskyt identifikátora premennej v texte zadania.
- Nahraď každý výskyt identifikátora premennej v reťazcoch, ktoré definujú vzorce pre výpočet správneho riešenia konkrétnou vyčíslenou hodnotou.
- Pre všetky správne a nesprávne odpovede vyhodnoť, pomocou funkcie *evaluate* z knižnice math.js, reťazce predstavujúce definíciu alebo vzorec pre ich výpočet.

Samotné generovanie je implementované funkciou *generate* v súbore `taskGenerator.js`. Výsledkom generovania je dátová štruktúra `ConcreteTask`, ktorej schému zobrazuje ukážka 4.2.

Kapitola 5

Nástroje a technológie pre vývoj webového systému

V tejto kapitole sú opísané možnosti pri výbere JavaScript rámcov pre vývoj webových rozhraní a štručne vysvetlené jednotlivé nástroje a technológie použité pri vývoji webovej aplikácie.

5.1 Rámce pre vývoj webových aplikácií

Vývoj webových aplikácií predstavuje rýchlo vyvíjajúce a meniace sa prostredie. Dochádza tu k neustálemu vytváraniu nových rámcov a knižníc. Niektoré z nich predstavujú iba nástroj na lepšie štruktúrovanie CSS štýlov, niektoré pomáhajú štruktúrovať a logicky rozčleniť zdrojový kód pre rýchlejší vývoj [3].

Využitie webových aplikačných rámcov pri vývoji webovej aplikácie nie je povinné, ale aj pre malé webové aplikácie dokážu priniesť veľké množstvo benefitov a to konkrétne v nasledujúcich aspektoch [3]:

- Architektúra založená na komponentoch – Komponenty predstavujú spôsob organizácie častí aplikácie do logických, znovupoužiteľných celkov. Zvyšujú prehľadnosť zdrojového kódu a znižujú časové náklady pre tvorbu prvkov užívateľského rozhrania. Každý framework ponúka iný spôsob ako ich definovať, nastaviť ich logiku, HTML kostru či CSS štýlovanie danej komponenty.

- Správa stavov – Každá aplikácia, ktorá zobrazuje dáta a dynamicky mení obsah na stránke potrebuje uchovávať informáciu o stave a hodnote týchto dát. Tie môžu byť ukladané priamo v komponente alebo globálne tak, aby k nim mali ostatné komponenty prístup.

- Manipulácia DOM – DOM predstavuje stromovú štruktúru webovovej stránky, ktorú tvoria HTML elementy. Frameworky umožňujú jednoduchšie prevedenie zmien v stromovej štruktúre bez nutnosti opätovného načítania celej stránky.

- Smerovanie – Navigácia medzi jednotlivými kartami webovej aplikácie môže prebiehať na základe URL adresy. Klient si žiada nový obsah od servera a ten presmeruje klienta na danú adresu. Prehliadač si uchováva históriu stránok, v ktorej je možné sa posúvať. Moderné aplikácie využívajú možnosť manipulácie DOM. Aktualizujú a vykreslujú v ňom nové elementy bez presmerovania a načítania novej stránky. Pri

komplexnejších webových aplikáciách je vhodné mať spôsob ako cez tieto zobrazenia efektívne prechádzať.

V rámci prieskumu z článku [3] bolo zistené, že medzi štyri najpoužívanejšie JavaScript nástroje pre vývoj webových aplikácií patria Angular, React.js, Vue.js a Svelte.

5.1.1 Angular

Angular[1], vytvorená spoločnosťou Google, vo svojej podobe od roku 2015 predstavuje rozsiahly framework zameraný na škálovetaľnosť a vývoj komplexných webových aplikácií. Je založený na jazyku TypeScript a nasleduje princíp jednej zodpovednosti, ktorý aplikuje pre všetky komponenty či služby a stanovuje predpísaný štýl vývoja aplikácií. Zastáva pravidlo, že každá komponenta, či služba by mala byť definovaná v samostatnom súbore. Ponúka program v príkazovom riadku, pomocou ktorého dokážeme ušetriť veľa času pri generovaní nového projektu, novej komponenty alebo služby. Kvôli svojej rozsiahlosti a odporúčanému štýlu vývoja, je jeho krivka učenia pomerne strmá. Na druhej strane však ponúka robustné a škálovateľné riešenie pre vývoj webových aplikácií [3].

5.1.2 React.js

React[2] predstavuje knižnicu od spoločnosti Facebook, ktorej kľúčové vlastnosti sú kompozícia komponentov, správa stavov a vykreslovanie DOM. Zameriava sa na stabilitu, interoperabilitu a dobrú vývojársku skúsenosť pri práci [3]. Framework predstavuje časť View z MVC (Model-View-Controller) architektúry a je možné ho využiť spolu aj s inými nástrojmi. Vďaka Virtual DOM, ktorý predstavuje kópiu stromovej štruktúry HTML elementov uloženú v pamäti, vyniká React v rýchlosti manipulácie s DOM [2]. V roku 2020 bola najpoužívanejším JavaScriptovým nástrojom pre tvorbu webových aplikácií a v rámci spokojnosti developerov skončila na druhom mieste [3].

5.1.3 Vue.js

Vue.js[3] je knižnica s otvoreným zdrojovým kódom, prispôsobiteľná a škálovateľná. Podľa prípadu použitia, je možné ju využiť ako knižnicu alebo ako plnohodnotný framework. Je porovnávaná s Angularom avšak Vue nekladie tak striktné medze na štruktúru kódu a súborov.[3] Prirovnať ho je možné ku knižnici React.js, s ktorou zdieľa podobnosť v tom, že predstavuje iba časť View z MVC architektúry. Jeho autorom je Evan You, ktorý spolupracoval na vývoji webového aplikačného rámca Angular. Vue.js vznikol na základe, že Angular predstavoval pri menších projektoch príliš rozsiahly nástroj. Zo všetkých rámcov je najvýkonnejšia a disponuje kvalitnou dokumentáciou, ktorá umožní rýchlo sa v nástroji zorientovať a pochopiť jej základné koncepty v podobe praktických ukážok [16].

5.1.4 Svelte

Svelte[4] je najnovšia zo všetkých štyroch webových aplikačných rámcov [3]. V roku 2016 ho vytvoril Richard Harris s cieľom zminimalizovať veľkosť, ktorú aplikácie zaberajú v pamäti. Svelte sa zameriava aj na výkonnosť, v ktorej poráža všetky vyššie zmienené nástroje [13].

[1] Angular – https://angular.io/
[2] React.js – https://reactjs.org/
[3] Vue.js – https://vuejs.org/
[4] Svelte – https://svelte.dev/

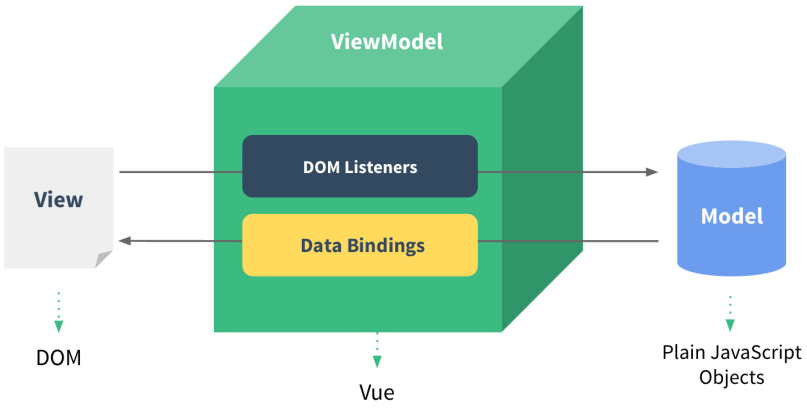
V roku 2019 bola prepísaná do jazyka TypeScript. Narozdiel od ostatných nástrojov Svelte zapísaný kód neinterpretuje ale prekladá priamo do JavaScriptu [3].

5.2 Vybrané nástroje pre vývoj klientskej časti systému

V tejto podkapitole popisujem zvolené technológie, ktoré som využil pri vývoji klientskej časti webového systému.

5.2.1 Vue.js

Po zhodnotení dostupných možností webových rámcov pre vývoj aplikácií som sa rozhodol využiť nástroj Vue.js. Dôvody pre výber tohto aplikačného rámca boli výborné výsledky vo výkonnosti, škálovateľnosť, fexibilita a detailná dokumentácia. Vue.js je možné rozšíriť pomocou ďalších knižníc a modulov z Vue ekosystému ako je napríklad Vue Router[5] alebo Vuex[6] a ďalšie. Tieto rozšírenia sú detailnejšie popísané v nasledujúcich sekciách.



Obr. 5.1: Ukážka architektúry MVVM, využívaná nástrojom Vue.js. Obrázok prevzatý z: https://012.vuejs.org/images/mvvm.png.

Nástroj Vue.js je vďaka svojej flexibilite a škálovateľnosti vhodný na vývoj projektov rôznej veľkosti. Je využitý vo veľkom množstve open-source projektov ale aj viacero populárnejších webových stránok či aplikácií ako sú napríklad Grammarly[7], Habitica[8] alebo 9GAG[9] [5].

Pri vývoji je možné aby bola každá komponenta uložená v súbore s príponou **.vue**. V nej je možné určiť tri základné časti ktoré definuje nasledujúca syntax tagov, podobná značkovaciemu jazyku HTML. Tag **template** v sebe zahŕňa HTML kostru danej komponenty. Medzi tagmi **style** sa majú nachádzať jednotlivé CSS štýly komponenty. Tieto štýly môžu byť limitované iba pre jednu komponentu ale môžu byť dostupné aj pre ostatné komponenty

[5] Vue Router – Oficiálny smerovač pre Vue.js aplikácie, dostupný na https://router.vuejs.org/
[6] Vuex – Nástroj pre centralizované spravovanie stavov vo Vue.js aplikáciách, dostupný na https://v3.vuex.vuejs.org/
[7] Grammarly – https://www.grammarly.com/
[8] Habitica – https://habitica.com/static/home
[9] 9GAG – https://9gag.com/

aplikácie. Hlavnú časť predstavuje tag **script**, ktorý umožňuje definovať aplikačnú logiku danej komponenty. Tento typ zápisu komponent sa nazýva Single-File Component[10] [5].

Architektúra MVVM (Model–View–Viewmodel), znázornená na obrázku 5.1, predstavuje základný pilier, na ktorom je Vue.js postavený. Časť *View*, ktorá je definovaná stromovou štruktúrou DOM, zahŕňa jednotlivé viditeľné prvky aplikácie. *Model* predstavuje dáta, ktoré sú ukladané pomocou JavaScript objektov. Prepojenie týchto dvoch častí sprostredkuje *Viewmodel*, čím sa dosiahne obojstranná väzba dát. Toto správanie v rámci Vue.js je možné dosiahnúť pomocou direktívy `v-model` [5].

Direktívy umožňujú vložiť hodnotu premenných uložených v objektoch do stromovej štruktúry DOM niekoľkými spôsobmi. Textová interpolácia umožňuje nahradiť každý výskyt `{{ nazov_premennej }}` v sekcii **template** za hodnotu danej premennej. Pomocou direktív je možné naviazať vlasnosti HTML elementov na premenné a tým zaistiť dynamický obsah stránok. Direktívy ponúkajú funkcionalitu ako podmienené zobrazenie komponent pomocou `v-if` a `v-show`. Direktíva `v-for` slúži na iteráciu a zobrazenie viacero komponent s rôznymi hodnotami [14].

5.2.2 Vuetify

Vuetify[11] predstavuje rozsiahlu, viacúčelovú knižnicu ponúkajúcu sadu prvkov grafického užívateľského rozhrania. Knižnica definuje vzhľad pre veľké množstvo často používaných, znovupoužiteľných prvkov, ktoré sú prístupné, plne prispôsobiteľné a založené na dizajnovom jazyku Material Design[12] od spoločnosti Google [19].

Knižnica disponuje rozsiahlym aplikačným programovacím rozhraním pre jednotlivé prvky, ktoré umožňuje špecifikovať ich atribúty, pridávať funkcionalitu či dokonca aj meniť vnútornú kompozíciu prvkov. Prvky je možné ďalej prispôsobiť za pomoci predpripravených konfigurovateľných CSS štýlov, ktoré uľahčujú vývoj. Vuetify sprístupňuje a umožňuje použitie niekoľko rôznych kolekcií a verzií ikoniek typu Material Design a Font Awesome[13] [19].

5.2.3 Vue Router

Vue Router[14] je pomocná knižnica, ktorá rozširuje základ Vue.js o možnosť jednoduchého smerovania medzi stránkami aplikácie. Objekt `VueRouter` definuje atribút `routes`, v ktorom je možné namapovať jednotlivé cesty ku komponentám. O zobrazenie aktuálnej komponenty sa stará direktíva `router-view`. Slúži na určenie miesta v aplikácii, v ktorom bude zobrazená komponenta pridelená k ceste na ktorej sa užívateľ nachádza [10].

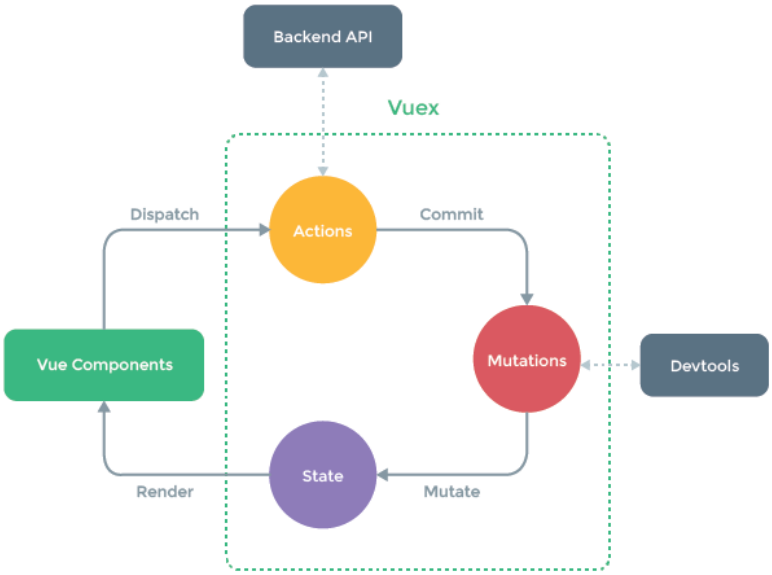
Umožňuje globálne definovať funkcie pri prechode medzi stránkami, ktorými je možné overiť identitu a práva užívateľa. Tieto funkcie je možné definovať aj na úrovni jednotlivých komponentov [10].

5.2.4 Vuex

Knižnica Vuex[15] rozširuje základ Vue.js o možnosť centralizovanej správy dát. Jej využitie je vhodné pri väčších projektoch, nakoľko jeho architektúra a z neho vyplývajúci spôsob

[10] V práci budem ďalej používať skratku SFC.
[11] Vuetify – https://vuetifyjs.com/en/
[12] Material Design – https://material.io/
[13] Font Awesome – https://fontawesome.com/
[14] Vue Router – https://router.vuejs.org/
[15] Vuex – https://vuex.vuejs.org/

zápisu je pomerne komplexný a nadmerne zdĺhavý pri menších aplikáciách. Cena za mierne zdržanie zo začiatku sa vyplatí pri dlhodobej údržbe a prehľadného členenia kódu. [7].



Obr. 5.2: Schéma znázorňujúca spôsob modifikácie dát pomocou knižnice Vuex. Obrázok prevzatý zo stránky: https://vuex.vuejs.org/vuex.png.

Premenné sú uložené v objekte typu `Store` pod atribútom `state` [7]. Modifikácia hodnoty premennej je možná pomocou metód, definovaných v atribúte `actions` a `mutations`. Jednotlivé metódy z atribútu `actions` sú volané pomocou metódy *dispatch* z globálneho objektu `$store`. Tieto metódy môžu byť asynchrónne a najčastejšie sa využívajú na získanie aktuálnych dát z databáze. Samotnú hodnotu je však možné modifikovať výhradne pomocou metód z atribútu `mutations`. Tie predstavujú synchrónne metódy z dôvodu zachovania konzistencie dát [7]. Princíp tohto procesu znázorňuje ukážka 5.2.

Dáta, ktoré `$store` objekt uchováva sú po opätovnom načítaní stránky zmazané. Tento problém rieši malá knižnica `vuex-persistedstate`[16], ktorá predstavuje zásuvný modul pre knižnicu Vuex. Na ukladanie dát využíva API `localStorage`.

5.2.5 Axios

Axios[17] je knižnica slúžiacia na zasielanie HTTP požiadaviek a prijímanie odpovedí. Môže byť využítá na serverovej aj klientskej strane. Na strane klienta využíva API `XMLHttpRequests`, kým na serverovej strane využíva natívny modul `http`. Medzi jej prednosti patrí využitie Promise API, transformácia dát požiadaviek a odpovedí, automatická konverzia JSON[18] súborov a jednoduché použitie [17].

[16] https://www.npmjs.com/package/vuex-persistedstate
[17] Axios – https://axios-http.com/
[18] JSON – JavaScript Object Notation

5.3 Vybrané nástroje pre vývoj serverovej časti systému

V tejto podkapitole sú popísané jednotlivé technológie použité pri vývoji serverovej časti systému.

5.3.1 Node.js

Node.js[19] predstavuje asynchrónne, udalosťami riadené behové prostredie pre JavaScript. Je zameraný na vývoj výkonných a škálovateľných webových aplikácií. Jeho jadro predstavuje slučka udalostí, pomocou ktorej dokáže konkurentne obsluhovať prichádzajúce požiadavky. Výhodou je, že nevyužíva zámky, čím eliminuje možnosť aby pri zápise alebo čítaní došlo k deadlock-u. Od ostatných podobných nástrojov sa odlišuje tým, že nie je nutné túto slučku udalostí spúšťať manuálne na konci skriptu. Je spustená automaticky akonáhle je skript volaný [15].

5.3.2 Express.js

Express.js[20] je minimalistický, flexibilný framework pre Node.js využívaný na tvorbu výkonných webových aplikácií a robustných rozhraní na programovanie aplikácií. Je základom pre viaceré iné webové rámce a knižnice [15].

5.3.3 Sequelize

Sequelize[21] je moderná ORM[22] knižnica stavaná pre TypeScript a Node.js. Podporuje databázy ako sú MySQL[23], PostgreSQL[24], MariaDB[25], SQLite[26] a dalšie. Prevádza automatické mapovanie dát uložených v relačných databázach na objekty reprezentované v objektovo-orientovaných jazykoch. Sequelize umožňuje definovať schému databáze priamo v programovacom jazyku a poskytuje pokročilé možnosti dopytovania a manipulácie dát v databázach [15].

5.3.4 Lodash

Lodash[27] je pomocná JavaScript knižnica, ktorá ponúka funkcie pre jednoduchšiu manipuláciu s číslami, reťazcami, zoznamami a objektmi [1].

5.3.5 Bcrypt

Bcrypt[28] je Node.js knižnica, používaná na hashovanie hesiel.

[19] Node.js – https://nodejs.org/en/
[20] Express.js – https://expressjs.com/
[21] Sequelize – https://sequelize.org/
[22] ORM – Object–relational Mapping
[23] MySQL – https://www.mysql.com/
[24] PostgreSQL – https://www.postgresql.org/
[25] MariaDB – https://mariadb.org/
[26] SQLite – https://www.sqlite.org/index.html
[27] Lodash – https://lodash.com/
[28] Bcrypt – https://www.npmjs.com/package/bcryptjs

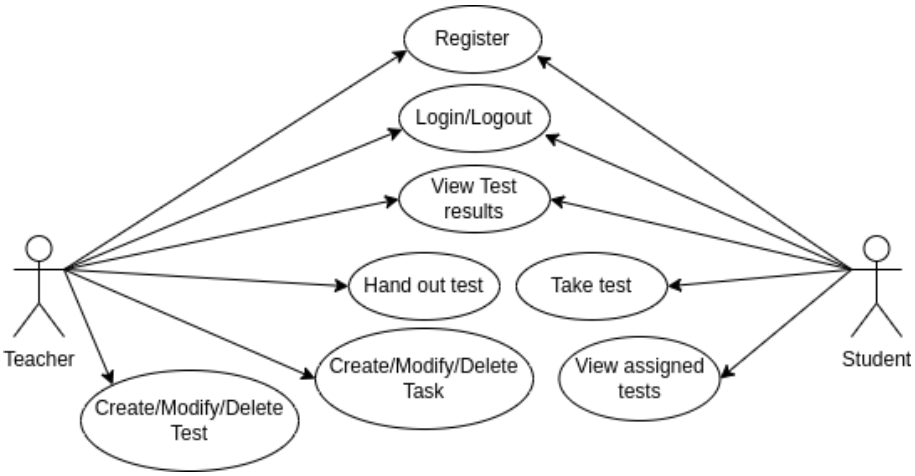
Kapitola 6

Návrh webového systému

Táto kapitola sa venuje návrhu architektúry a grafického užívateľského rozhrania webového systému. Tento systém využije implementovaný interpreter umožňujúci tvorbu matematických príkladov a testov. V návrhu sa zameriam, aby som na základe zvolených technológií, popísaných v kapitole 5, zjednodušil prácu s jazykom a jeho interpretom.

6.1 Požadovaná funkcionalita a prípady použitia

Pred samotným návrhom bolo nutné stanoviť všetky užívateľské požiadavky na funkcionalitu systému. Najdôležitejšiu časť systému predstavuje proces tvorby matematických príkladov a následné využitie týchto príkladov na zostavenie testov. V rámci príkladu by mala byť možnosť nastaviť typ odpovede na otázky, ktoré príklad zahrňuje a možnosť určiť kategóriu príkladu (slovná úloha, geometria, výraz a pod.).



Obr. 6.1: Diagram prípadov použitia

Ďalšie požiadavky vyplývajú z jednotlivých častí potrebných pre správny zápis príkladu. Tieto časti sú podrobnejšie opísané v podkapitole 3.4. Pri vytváraní príkladu by mal systém umožniť učiteľovi špecifikovať a modifikovať text príkladu. Pridávať, odstrániť a meniť jednotlivé otázky, *premenné* a *odpovede* daného príkladu. Každej *premennej* musí byť umožnené nastaviť definíciu a rozsah. Pre premenné, ktoré sa nenachádzajú v texte prí-

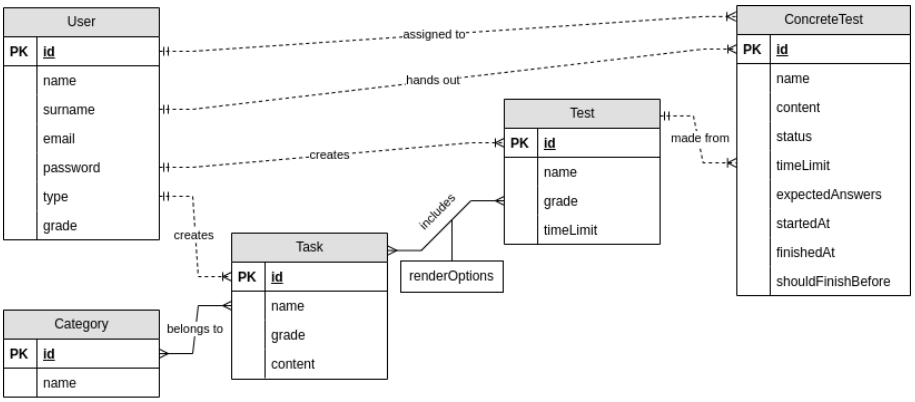
kladu alebo otázky by malo byť umožnené meniť aj ich meno. Pre *odpovede* musí existovať možnosť pridať, zmazať a modifikovať správne a nesprávne možnosti.

Jednotlivé príklady a testy by malo byť ďalej možné modifikovať či zmazať. Vytvorené testy by mohol učiteľ rozdať študentom danej triedy. Študenti by si mohli testy, ktoré im boli pridelené, následne spustiť, v danom časovom intervale vyplniť a odovzdať. Študent by mal mať možnosť pozrieť si výsledky testu ihneď po jeho odovzdaní. Aby si žiaci medzi sebou nemohli odpisovať, nebudú vidieť svoje odpovede pri hodnotení. Učiteľovi by mal byť poskytnutý prehľad o percentuálnych výsledkoch študentov pre daný test. V hodnotení by učiteľ videl odpovede študenta na jednotlivé otázky príkladov z testu.

Aby mohli učitelia či študenti systém využívať, bude potrebné aby sa najprv do systému zaregistrovali. Po úspešnej registrácii by mali možnosť sa prihlásiť či odhlásiť. Prehľad možností, ktoré užívateľ môže v rámci systému vykonať je znázornený na obrázku 6.1. Na základe tohto diagramu som postupoval na ďalšiu časť návrhu, ktorá zahŕňala návrh dátového modelu.

6.2 Dátový model

Štruktúra dátového modelu vyplýva z požiadaviek definovaných v predchádzajúcej podkapitole 6.1. V systéme bude nutné uchovávať údaje o jednotlivých používateľoch, príkladoch, testoch a kategóriách.



Obr. 6.2: ER Diagram znázorňujúci jednotlivé entity v systéme a ich vzťahy

Používateľov systému je môžné rozdeliť na dva typy – učiteľ a študent. O každom užívateľovi je potrebné ukladať jeho meno, priezvisko, e-mailovú adresu, heslo a jeho typ. Pre študenta je potrebné navyše ukladať aj informáciu, do ktorého ročníka patrí.

Každý príklad by mal obsahovať nasledujúce informácie – meno príkladu, ročník pre ktorý je príklad určený, obsah príkladu (reprezentovaný šablónovým zápisom ako na ukážke v kapitole 3.4.7), a informáciu o učiteľovi, ktorý príklad vytvoril. Pre kategórie stačí ukladať ich názov.

Pri ukladaní údajov o testoch treba zohľadniť či sa jedná o test, ktorý už bol alebo ešte nebol rozdaný študentom. O testoch, ktoré ešte neboli rozdané je potrebné uchovávať údaje o názve testu, ročníku, pre ktorý je test určený, časový limit, za ktorý musí študent po spustení odovzdať test a príklady, ktoré sa v teste nachádzajú.

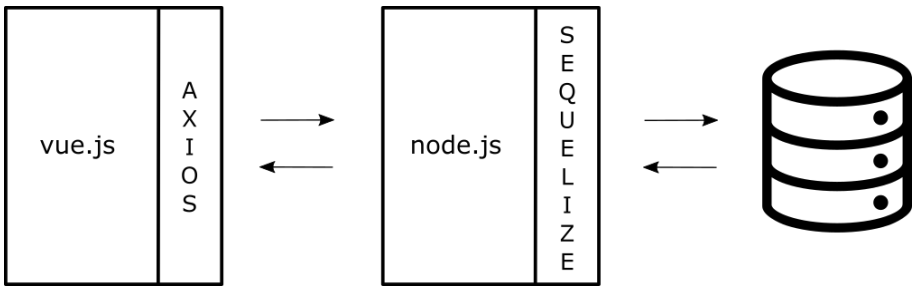
Testy, ktoré už boli rozdané študentom nemôžu byť reprezentované rovnakým modelom ako nerozdané testy. O rozdaných testoch je potrebné mať okrem ich názvu, časového limitu uložený ich obsah, ktorý reprezentuje zoznam príkladov s konkrétnymi hodnotami vo formáte JSON. Ďalej je potrebné ukladať si stav testu, ktorý môže byť jeden z nasledujúcich možností – rozdaný, rozpracovaný alebo odovzdaný. Kým študent test nespustí, tak je v stave *rozdaný*, po spustení pred odovzdaním je test *rozpracovaný* a po ukončení testu alebo vypršaní časového limitu je test v stave *odovzdaný*. Dôležitým údajom pri rozdaných testoch sú očakávané správne odpovede v jednotlivých otázkach, čas spustenia testu, čas ukončenia testu a čas, kedy sa test automaticky ukončí. V neposledom rade, je potrebné mať uložené údaje o tom, ktorému študentovi bol konkrétny test rozdaný a učiteľ, ktorý daný test rozdal. Výsledný dátový model je na ukážke 6.2.

6.3 Architektúra systému

Architektúra systému bude založená na dvojvrstvovej architektúre *klient-server*, kde serverovú časť, bude predstavovať RESTful aplikačné rozhranie. Ukážka 6.3 predstavuje schému návrhu architektúry.

Server bude zodpovedný za spracovanie požiadaviek pre generovanie príkladov a komunikáciu s databázou. Táto časť systému bude postavená na behovom prostredí Node.js s využítím webového rámca Express pre komunikáciu s klientskou časťou. Knižnica Sequelize bude zodpovedná za manipuláciu s dátami v databáze.

Klientská časť systému, zodpovedná za zobrazenie bude využívať knižnicu Vue.js pre vytvorenie webového užívateľského rozhrania a definovanie aplikačnej logiky jednotlivých stránok. Smerovanie v klientskej časti zaistí knižnica Vue Router a pre ukladanie stavov bude využitá knižnica Vuex. Pre jednotlivé prvky grafického užívateľského rozhrania využijem knižnicu Vuetify. Na komunikáciu so serverom bude použitá knižnica Axios, ktorá bude mapovať požadované funkcie na koncové body API servera.



Obr. 6.3: Ukážka znázorňujúca dvojvrstvovú architektúru spolu so zvolenými technológiami, ktoré zabezpečujú požadovanú funkcionalitu na jednotlivých vrstvách.

6.4 Užívateľské rozhranie

Pri návrhu som najskôr preniesol svoje nápady do nástroja Figma[1], pomocou ktorého som si vytvoril prvotný prototyp. Zameral som sa hlavne na stránku pre vytváranie príkladov, ktorej prototyp je možné vidieť na ukážke 6.4.

[1]Figma – www.figma.com

6.4.1 Základné stránky a časti rozhrania

Pred vstupom do systému sa neprihlásenému používateľovi bude zobrazovať stránka s prihlasovacím formulárom. Formulár bude obsahovať vstupné polia pre mailovú adresu a heslo. Pod tlačítkom pre prihlásanie sa bude nachádzať odkaz na stránku s registračným formulárom. Používateľ sa môže registrovať ako učiteľ alebo študent vyplnením všetkých povinných polí.

Na úvodnej stránke som sa snažil držať minimalistického a ilustratívneho [18] dizajnu a dodržať princíp vynechania zbytočných slov [8]. Návrh domovskej stránky je bez ďalších úvodných textov, zložený z dvoch kartičiek popisujúcich hlavné funkcie systému. Kartičky obsahujú pomocnú ilustráciu a tlačítko s popisom.

Navigačná lišta bude tvorená z loga umiestneného na ľavej strane a na pravej strane bude obsahovať odkazy v podobe tlačítok na hlavné funkcie systému. V pravom rohu lišty sa bude nachádzať rozbaliteľné menu s položkami obsahujúce tlačítka. Pre prihláseného učiteľa budú zobrazené talčítka odkazujúce na stránku so zoznamom vytvorených príkladov a stránku so zoznamom vytvorených testov. Prihlásený študent bude vidieť tlačítko s odkazom na stránku so zoznamom jemu pridelených testov. Každý typ používateľa tu bude mať ešte tlačítko s možnosťou odhlásiť sa zo systému.

Systém bude ďalej pre učiteľa poskytovať stránky zo zoznamom všetkých vytvorených príkladov a testov a pre študenta zas stránku so zoznamom jemu pridelených testov.

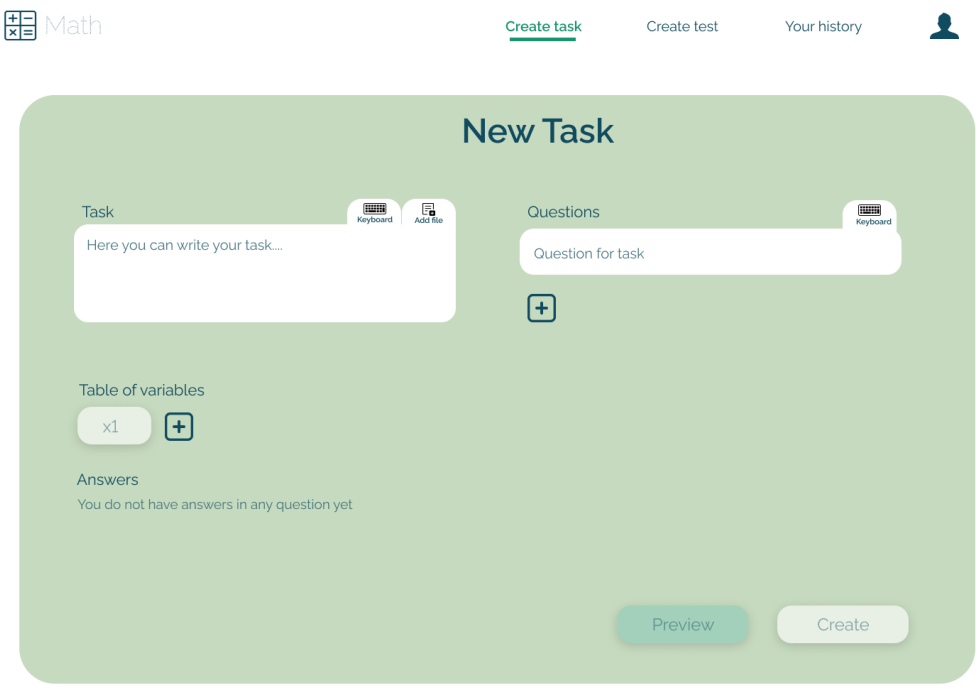
6.4.2 Vytváranie príkladov

Vytváranie šablón príkladov predstavuje jadro webového systému a zároveň aj najzložitejší proces, ktorý je potrebné navrhnúť. Pri návrhu musím zohľadniť spôsob zápisu šablóny popísaného v kapitole 3.4.

Aby sa používateľ nemusel učiť podrobnú syntax navrhnutého jazyka pre vytváranie šablón príkladov, bude potrebné vytvoriť zrozumiteľné užívateľské rozhranie, ktoré zjednoduší prácu so zápisom. Je vhodné aby interné fungovanie nebolo priamo prístupné používateľovi. To znamená, že užívateľ nebude priamo modifikovať zápis šablóny ale bude zápis upravovať pomocou interaktívných prvkov, ktoré mu rozhranie ponúkne. Tým by sa docielila menšia chybovosť pri vytváraní príkladov, jednoduchšia manipulácia s navrhnutým jazykom a časovo effektívnejší spôsob vytvárania príkladov pre učiteľa, ktorý je jedným z hlavných cieľov tejto práce.

Užívateľské rozhranie stránky som navrhol na základe štyroch hlavných častí zápisu šablóny, ktoré predstavujú – text príkladu, premenné, odpovede a otázky. Pri vytváraní príkladu je vhodné aby mal používateľ prehľad o jednotlivých častiach vytváraného príkladu. Z toho dôvodu som dbal na to aby sa jednotlivé elementy zmestili na jednu stránku bez nutnosti posunu. Stránka by bola rozdelená na tieto štyri časti ponúkajúcu logickú postupnosť pri vytváraní príkladu. Prvotný digitálny prototyp stránky pre vytváranie príkladov je možné vidieť na obrázku 6.4.

Na ľavej strane stránky bude pre text príkladu vyčlenené textové pole, ktoré sa bude môcť dynamicky zväčšiť na základe množstva textu. Vpravo od neho bude priestor pre otázky príkladu, ktoré si používateľ pridá pomocou tlačítka s ikonkou. Pretože príklad môže obsahovať neobmedzené množstvo otázok, priestor pod nadpisom „questions“ vyčleňujem pre zoznam otázok daného príkladu. Zoznam premenných a odpovedí zobrazujem pod textovým poľom v podobe menších blokov, ktoré sa budú dať následne rozkliknúť. Po kliknutí sa používateľovi zobrazí dialógové okno, v ktorom si môže nastaviť definíciu a roz-



Obr. 6.4: Ukážka prototypu stránky pre vytváranie príkladov.

sah premennej. Rovnaký princíp aplikujem aj pre odpovede, kde dialógové okno umožní modifikovať správne a nesprávne možnosti pre odpoveď.

Pri vstupe na túto stránku bude používateľovi zobrazené dialógové okno s vysvetlením princípu tvorby príkladov. Pomocou elementov zo stránky bude mať znázornené ako postupovať pri vytváraní a aké možnosti systém ponúka.

6.4.3 Vytváranie testov

Pred samotným návrhom stránky pre tvorbu testov je potrebné definovať postup a všetky nevyhnutné údaje, ktoré učiteľ musí mať k dispozícii pri procese vytvárania testu. Údaje o teste je možné vidieť v ER diagrame na obrázku 6.2.

Najprv je potrebné, aby si učiteľ vybral príklady, ktoré chce v danom teste použiť. Pri tomto kroku je potrebné v návrhu zohľadniť ďalšiu funkcionalitu, ktorú by učiteľ pri tomto kroku očakával. Môže to byť filtrovanie zoznamu príkladov, prípadná rýchla modifikácia príkladu alebo následné zmazanie príkladu zo zoznamu. Pri výbere príkladov musí byť učiteľovi zrejmé, ktoré príklady si vybral a akým spôsobom sa pre študenta zobrazia. Pred uložením si musí učiteľ vedieť určiť názov testu a časový limit pre jeho vyplnenie a po uložení ich musí vedieť rozdať študentom danej triedy.

Na základe týchto požiadaviek je zrejmé, že učiteľ si bude musieť vybrať pomerne veľké množstvo údajov a mohlo by byť jednostránkové zobrazenie pre učiteľa neprehľadné. Pri riešení som sa riadil princípmi pre dizajnovanie procesov z knihy Jenifer Tidwell [18], ktorá odporúča zložité aktivity rozdeliť na niekoľko krokov. Pri prechode jednotlivými krokmi procesu je vhodné užívateľa informovať o aktuálnom kroku, na ktorom sa v procese práve nachádza. Pri jednotlivých krokoch zasa umožniť, aby sa užívateľ mohol vrátiť na predošlý krok ak by si chcel zmeniť vybrané voľby.

Proces tvorby testu som sa rozhodol rozdeliť na štyri kroky, v ktorých budú údaje prezentované vo forme kartičiek. Tie nám pomôžu jasne ohraničiť dané možnosti a jednoduchšie škálovať rozhranie pre rôzne veľkosti zariadení [18].

Prvým krokom bude výber triedy, pre ktorú bude test vytváraný. V druhom kroku bude výber kategórií príkladov, ktoré by mohli byť v teste zahrnuté. Tretí krok umožní výber jednotlivých príkladov zo zoznamu, ktorý zobrazuje dostupné príklady danej triedy a kategórií. Keď si učiteľ vyberie požadované príklady do testu, zobrazia sa mu v podobe náhľadu, ktorý bude odpovedať zobrazeniu viditeľnému pre študenta pri vyplňovaní. V tomto náhľade si bude môcť upraviť typ odpovede pre jednotlivé príklady, rýchlu voľbu pre úpravu šablóny príkladu alebo zmazanie príkladu z testu.

Časový limit určený pre daný test si bude môcť nastaviť pomocou posuvníka na škále od 5 až 45 minút. Maximálna možná hodnota odpovedá jednej vyučovacej hodine na základnej škole.

6.4.4 Prehľad výsledkov testov

Stránka s prehľadom výsledkov jednotlivých študentov bude dostupná po rozdaní a vyplnení testu aspoň jedným študentom. Prehľad bude tvoriť tabuľka, v ktorej budú zobrazené mená študentov a ich percentuálne hodnotenie z testu. Po rozkliknutí získa učiteľ prehľad o odpovediach študenta na jednotlivé otázky príkladov. Prehľad bude obsahovať grafické znázornenie výsledku pomocou farebného kruhu, pod ktorým bude možné vidieť jednotlivé príklady a vyplnené odpovede študenta.

Kapitola 7

Implementácia webového systému

V tejto kapitole popisujem implementáciu webového systému pre vytváranie príkladov a testov. Implementácia systému bola rozdelená na dve hlavné časti. Implementácia najdôležitejších častí na klientskej strane aplikácie je popísaná v podkapitole 7.1 a v podkapitole 7.2 je opísaná implementácia serverovej časti systému.

7.1 Implementácia klientskej časti

Po vytvorení návrhu mohla nasledovať implementácia, ktorú som začal klientskou časťou aplikácie. Konkrétnym prvým krokom bolo vytvorenie nového Vue projektu pomocou nástroja `Vue CLI`. Príkazom `vue create <názov_projektu>` sa v príkazovom riadku spustí sprievodca vytvorením nového projektu. Tu je možné manuálne vybrať jednotlivé súčasti a nastavenia projektu alebo vybrať predkonfigurované voľby pre Vue 2 alebo Vue 3. Konfigurácia zahŕňa výber súčastí projektu ako sú knižnice Vue Router alebo Vuex, výber verzie Vue, výber CSS preprocessora a dalšie. Pre prácu som si vybral konfiguráciu projektu Vue verzie 2 s využitím knižníc Vue Router a Vuex, bez CSS preprocesoru. Po vytvorení projektu som pomocou príkazu `vue add vuetify` do projektu pridal knižnicu Vuetify. Výsledná štruktúra relevantných zdrojových súborov je nasledovná:

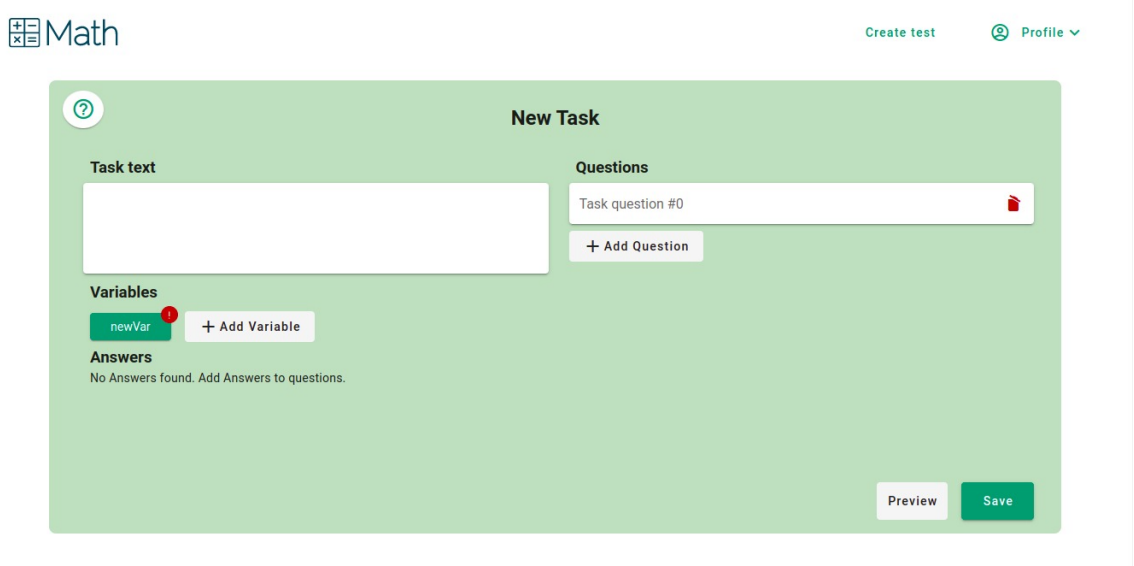
```
dist ..... zkompilované súbory pre nasadenie
node_modules ..... node.js balíky
src
    components
        pages ..... zložka s SFC komponentami stránok
        navbar.vue ..... komponenta hornej navigačnej lišty
        globalSnackbar.vue ..... komponenta oznámení
    languageHandlers ..... zložka so súbormi pre spracovanie zápisu
    plugins/vuetify.js ..... konfiguračný súbor knižnice Vuetify
    services ..... zložka so súbormi pre komunikáciu s API
    store ..... súbor spúšťajúci webové rozhranie
    App.vue ..... komponenta so základným rozložením rozhrania
    main.js ..... súbor inicializujúci framework
server.js ..... súbor spúšťajúci webové rozhranie
axios.js ..... konfiguračný súbor pre knižnicu Axios
package.json
```

7.1.1 Základné rozloženie užívateľského rozhrania

Súbor `App.vue` obsahuje hlavné rozdelenie užívatelského rozhrania. V časti `template` sa nachádza element `v-app` z knižnice Vuetify, ktorý predstavuje povinný element každej aplikácie využívajúcej Vuetify. V nej, podľa odporúčaného postupu v dokumentácii knižnice, definujem základnú štruktúru aplikácie pomocou komponenty `v-app-bar` a `v-main`, v ktorej sa nachádza direktíva `router-view` zobrazujúca aktuálnu stránku. V komponente `Navbar`, zapuzdrujem navigačnú lištu s využitím `v-app-bar`. Zložka `pages` obsahuje súbory s príponou .vue, ktoré implementujú jednotlivé stránky aplikácie ako komponenty typu SFC.

7.1.2 Proces vytvárania príkladov

Najdôležitejšiu funkcionalitu v celom systéme predstavuje vytváranie príkladov a testov. Na základe návrhu som implementoval rozloženie stránky, kde som využil prvky z knižnice Vuetify. Výsledný vzhľad je možné vidieť na obrázku 7.1.



Obr. 7.1: Ukážka implementovaného prototypu stránky pre vytváranie príkladov

Jednotlivé časti sú spracované pomocou modulu `ExtractorModule`, v ktorom definujem triedu `Extractor`. Trieda definuje atribút task, ktorý je typu `Task`. V časti `script` súboru `createTask.vue` definujem inštanciu tejto triedy s názvom `extracted`. Ide o rovnakú štruktúru, ktorú vracia interpreter po spracovaní zápisu. V nej udržiavam informácie o texte príkladu, jednotlivých otázok, premenných a odpovediach.

Pri zmene textu príkladu alebo textu otázky sa po sekunde nečinnosti automaticky spustí funkcia *autoParseText*, ktorá analyzuje aktuálne hodnoty v textových poliach a aktualizuje hodnoty v objekte `extracted`. Ak pri analýze obsahu textových polí narazí premennú v texte, ktorá nie je uložená v objekte pridá ju do nej. Pri zmene akejkoľvek inej hodnoty ako sú premenné, ich atribúty alebo odpovede spôsobenou vstupom od užívateľa sa priamo volá funkcia `parseText` aktualizujúca objekt `extracted` novými hodnotami. Z hladiska MVVM architektúry, objekt extracted využívam ako model, ktorý aktualizujem na základe užívateľského vstupu.

V jednotlivých dialógových oknách, na formulárových komponentoch knižnice Vuetify aplikujem pravidlá, ktoré sa priebežne vyhodnocujú. Ak po pridaní alebo modifikácii jednej

z premenných chýba definícia alebo rozsah, túto skutočnosť indikujem pomocou červeného krúžka s výkričníkom.

Stlačením tlačítka *Preview* sa spúšťa funkcia *generateText*, ktorá z údajov uložených v atribúte task objektu *extracted* vygeneruje zápis šablóny príkladu. Ten je pomocou funkcie *generate* z triedy *GeneratorService* zaslaný na server. Odpoveďou je objekt obsahujúci atribúty `message`, ktorej hodnota je zobrazená pomocou oznámenia a `content` obsahujúca konkrétny vygenerovaný príklad typu `ConcreteTask`. Ten sa následne zobrazuje v dialógovom okne.

Pri stlačení tlačítka *Save* sa obsah objektu `extracted` ešte raz spracuje pomocou funkcie *parseText* a na základe výslednej štruktúry funkcia *generateText* vytvorí zápis príkladu. Ten sa pomocou funkcie *create* z triedy `TaskDataService` posiela pomocou HTTP metody POST na koncový bod API serveru, kde je spracovaný. Počas čakania na odpoveď v tlačítku zobrazujem animáciu, ktorá dáva používateľovi vedieť, že sa vybavuje jeho požiadavka. Po úspešnom spracovaní požiadavky na uloženie príkladu, ktorý sa vyznačuje kódom 200 v HTTP hlavičke, je používateľ presmerovaný na stránku so zoznamom vytvorených príkladov.

Pri voľbe modifikovať už existujúci príklad je volané rovnaké zobrazenie ako pre jeho vytváranie. Ak sa na stránku vytvárania príkladu presmeruje cez tlačítko *Create task* v navigačnej lište alebo tlačítko na kartičke domovskej stránky, v lokálnej pamäti vymažem údaj o identifikačnom čísle príkladu z databázy. Ak bol užívateľ presmerovaný na stránku vytvárania z iného miesta, do lokánej pamäti si ukladám zvolené identifikačné čislo. Na základe tohto čísla sa najprv zavolá funkcia *get* z triedy *taskDataService*, ktorý načíta zápis príkladu z databáze. Zápis je spracovaný pomocou interpreteru a výsledná štruktúra uložená do objektu *extracted*.

Pomocou funkcie *beforeRouteEnter*, ktorá sa vykoná pred dokončením presmerovania, rozlišujem z akej stránky bol používateľ presmerovaný a ukladám si túto informáciu pomocou localStorage API do lokálnej pamäte prehladača. Pri uložení príkladu sa na základe tejto hodnoty rozhoduje, či presmerovať na stránku so zoznamom príkladov alebo späť na stránku pre vytváranie testov.

7.1.3 Tvorba testov

Na stránke pre vytváranie testov využívam komponentu `v-stepper` z knižnice Vuetify, ktorá ponúka funkcionalitu popísanú v návrh pod kapitolou 6.4. Jednotlivé kroky pri vytváraní sú zobrazené v samostatných sekciách a dátový model, ktorý uchováva všetky vybrané údaje ako sú ročník, kategórie príkladov, vybrané príklady a nastavenia typu odpovedí pre príklady je uložený pomocou objektov z knižnice Vuex.

Pre ukladanie a načítanie dát využívam doporučený postup práce s knižnicou Vuex. Súbor *src/stores/modules/testDataModule* definuje všetky potrebné objekty a funkcie na ich nastavovanie.

Po zvolení si príkladov do testu, sú tieto príklady pomocou funkcie *generateMultiple* z triedy `GeneratorService` doplnené o konkrétne hodnoty a prezentované pri poslednom kroku vytvárania testu.

Ukladanie testu prebieha pomocou funkcie *createOrSaveTest*, ktorá volá funkciu *create* alebo *update* z triedy testDataService. Funkcia *create* najprv vytvorí nový test a pomocou *metódy* addTask pridá každý príklad k danému testu. Metóda *update* najprv vymaže všetky príklady, ktoré sa viažu k danému testu a následne pridá všetky príklady, ktoré boli vybraté.

Takto uložené testy v databázi predstavujú iba šablónu, konkrétny test sa vygeneruje až pri rozdávaní testu študentom.

V zozname vytvorených testov po kliknutí na ikonku pod stĺpcom *Hand out* sa v dialógom okne načíta zoznam všetkých žiakov daného ročníka, pre ktorý bol test vytvorený. Po potvrdení výberu žiakov, ktorým sa má test rozdať a kliknutí na tlačítko *Hand out* sa volá funkcia *handoutTest*. V nej sa pre každého študenta vygeneruje a priradí test s konkrétnymi hodnotami a to nasledujúcim spôsobom. Pre všetky príklady, ktoré test obsahuje sa zavolá metóda *generateMultiple* a výsledný zoznam sa uloží vo formáte JSON ako hodnota do atribútu content pre model konkrétneho testu. Model konkrétneho testu je definovaný pod názvom `ConcreteTest` a na klientskej strane s ním komunikuje trieda *concreteTestDataService*. Pomocou metódy *create* vytvára nový konkrétny test.

7.1.4 Vyplnenie testu študentom

Pred vypĺňaním testu študentom sa aktualizuje položka v databáze reprezentujúca konkrétny test pridelený študentovi. Atribút *startedAt* sa nastaví na aktuálny čas a atribút *shouldFinishBefore* bude mať hodnotu aktuálneho času, ku ktorému sa pričíta časový limit pre daný test. Po tejto aktualizácií sa test zobrazí študentovi a môže vidieť zostávajúci čas, ktorý má na vyplnenie. Hodnota zostávajúceho času sa získava každú sekundu pomocou funkcie *syncTime*. Tá z databáze požiada o záznam konkrétneho testu, z ktorého využíva atribút *shouldFinishBefore*, ktorý porovnáva s aktuálnym časom. Ak je aktuálny čas menší v porovnaní s hodnotou *shouldFinishBefore* funkcia sa automaticky zavolá o dalšiu sekundu pomcou funkcie *setTimer*. V prípade, že aktuálny čas je rovný alebo vyšší než čas v atribúte *shoudlFinishBefore* test sa pre žiaka automaticky ukončí.

7.1.5 Navigácia a autentizácia

Navigáciu medzi stránkami som implementoval s využitím knižnice Vue Router. V súbore `main.js` si vytváram globálnu premennú `$router` a pomocou jej funckie *push* mením aktuálne zobrazenú stránku v direktíve `router-view`. Mapovanie jednotlivých URL ciest a komponent som si definoval v súbore `main.js`. Súbor `main.js` ďalej obsahuje definície pravidiel pre autentizáciu užívateľov. Pred každým presmerovaním sa vykoná funkcia *authorizeUserType*, ktorá skontroluje či je daný užívateľ prihlásený. Na overenie využívam špeciálny token, ktorý je uložený na strane klienta po úspešnom prihlásení sa do systému.

7.1.6 Oznámenia

Využitím prvku `v-snackbar` a knižnice Vuex som vytvoril komponentu, ktorú je možné globálne ovládať z ľubovoľnej komponenty v aplikácii. Pre zobrazenie stačí zavolať metódu showMessage, s príslušnými nastaveniami, pomocou funkcie dispatch z globálneho Vuex objektu. Nastavenia predstavuje objekt s nasledovnými atribútmi: message a suceess. Túto komponentu využívam pre oznámenie kladných aj záporných správ. Farbu pozadia určuje boolovský atribút `success`.

7.2 Implementácia serverovej časti

Serverová časť systému je implementovaná pomocou Node.js a aplikačného rámca Express. Vstupný bod aplikácie predstavuje súbor `server.js` spúšťaný pomocou príkazu `node server.js`, ktorý spúšťa samotný server. V tomto súbore si najprv nastavujem základnú

konfiguráciu serveru. Do objektu `db` načítam objektové dátové modely vytvorené knižnicou Sequelize zo zložky models. Volaním metódy *sync*, synchronizujem objektové dátové modely s databázou. Po synchronizácii pomocou Sequelize tried v objekte `db` vyplňujem databázu niekoľkými ukážkovými užívateľmi, príkladmi, kategóriami a testami.

Server predstavuje aplikačné rozhranie, ktoré je dostupné cez `názov_domény/api/`. Atribúty modelov sú definované v jednotlivých súboroch v adresári `models`. V rámci súboru `models/index.js` nastavujem vzťahy a väzby medzi jednotlivými modelmi, vytvorené na základe ER diagramu na obrázku 6.2.

Výsledná štruktúra relevantných zdrojových súborov je nasledovná:

```
├── routes ..... adresár so súbormi definujúce koncové body API
├── models ..... adresár so súbormi dátových modelov definovaných pomocou Sequelize
├── controllers ..... adresár so súbormi defunujúcimi aplikačnú logiku
├── verifications ..... adresár s middleware pre overenie užívateľa
├── package.json ..... konfiguračný súbor projektu, zoznam závislostí
├── database.config.js ..... konfiguračný súbor pre pripojenie k databáze
└── server.js ..... súbor spúšťajúci webové aplikačné rozhranie
```

7.2.1 Komunikácia s databázou pomocou Sequelize

Jednou z primárnych funkcií serverovej časti je dotazovanie sa na databázu. V prvom rade je potrebné pripojiť sa k danej databáze. Vytvorenie spojenia pomocou Sequelize prebehne ihneď po vytvorení novej inštancie Sequelize. Konštruktor si vyžaduje nasledujúce údaje – názov databáze, meno a heslo používateľa a konfiguráciu servera. Konfigurácia servera sa skladá z hosťovskej domény, typu databáze a ďalších nastavení pre daný typ databáze.

V mojej implementácii som využíval dve rôzne konfigurácie. Prvá bola určená na lokálne testovanie aplikácie, kým tú druhú využívam už pri nasadenej aplikácii. Nasadená aplikácia využíva MySQL databázu, ktorá je hosťovaná cez službu Microsoft Azure[1]. Po prechode na databázu hosťovanú cez Azure Database for MySQL[2], bolo potrebné rozšíriť konfiguračný súbor o nastevenie posunu času voči serverovému o dve hodiny.

Pužitím knižnice Sequelize je možné využívať už predpripravené, štandardné funkcie na dotazovanie sa na databázu. Funkcie pre vytvorenie, aktualizáciu či zmazanie sú štandardne dostupné pre každý definovaný model.

7.2.2 Koncové body API

Server ponúka pre každý model sadu koncových bodov umožňujúcich vykonávať CRUD (Create, Read, Update, Delete) operácie nad databázou a ďalšie funkcie definované v súbore `controllers/názov_modelu.controller.js`. Koncové body popisujú nasledovné možnosti v rámci funkcionality:

- `POST api/názov_modelu/` – Vytvorí nový záznam v tabuľke.
- `GET api/názov_modelu/` – Vráti všetky záznamy v tabuľke.
- `GET api/názov_modelu/:id` – Vráti jeden konkrétny záznam z tabuľky.
- `PUT api/názov_modelu/` – Aktualizuje existujúci záznam v tabuľke.

[1] Microsoft Azure – https://azure.microsoft.com/en-us/
[2] Azure Database for MySQL – https://azure.microsoft.com/en-us/services/mysql/#overview

- `DELETE api/názov_modelu/:id` – Vymaže jeden konkrétny záznam v tabuľke.
- `DELETE api/názov_modelu/` – Vymaže všetky záznamy v tabuľke.

Okrem týchto základných koncových bodov je v API implementovaných niekoľko ďalších zaujímavých, ktoré sú potrebné pre funkcionalitu systému:

- `GET api/concreteTests/:id/results` – Vráti výsledky študenta z konkrétneho rozdaného testu.
- `POST api/tasks/add_category` – Priradí danú kategóriu pre konkrétny príklad.
- `POST api/tasks/add_task` – Priradí daný príklad k testu.
- `GET api/users/:id/assignments` – Vráti všetky pridelené testy danému študentovi.
- `POST api/users/studentsByGrade` – Vráti všetkých študentov podľa zadaného ročníka.

7.2.3 Generovanie príkladov

Serverová časť systému je ďalej zodpovedná za generovanie konkrétnych príkladov zo šablón zápisu. Dôvodom je, aby sme ušetrili na výkone u menej výkonných zariadeniach atd.

Koncové body, ktoré sú zodpvedné za generovanie:

- `POST api/generate` – Vygeneruje jeden konkrétny príklad na základe šablóny príkladu.
- `POST api/generate/multiple` – Vygeneruje zoznam konkrétnych príkladov na základe poskytnutých informácií.

7.2.4 Autentifikácia užívateľa

Autentifikácia užívateľa využíva knižnicu `jsonwebtoken` a `bcryptjs`. Koncový bod

`POST api/users/login`

prihlasuje užívateľa na základe poskytnutých údajov. Pri úspešnom overení, že sa mailová adresa nachádza v databáze, overím pomocou funkcie *bcrypt.compareSync*, či sa zhoduje používateľom poskytnuté heslo s tým heslom, ktoré má uložené v databáze. Ak sa heslá zhodujú použijem funkciu *jwt.sign* pre vytvorenie unikátneho tokenu. Odpoveď na požiadavku obsahuje atribúty message a v objekte data, atribút token, ktorý sa následne uloží do lokálneho úložiska cez `localStorage` API.

Kapitola 8

Testovanie

Testovanie je neoddeliteľnou časťou vývoja softvéru a produktov vo všeobecnosti. V mojej práci som vyvíjaný systém a jednotlivé jeho časti testoval postupne počas konkrétnych fáz implementácie.

V prvej fáze práce, ktorá zahrňovala návrh jazyka pre zápis príkladov a implementáciu interpretera, som testoval výsledný interpreter pomocou zápisov jednoduchých príkladov. V druhej fáze práce, kde sa pre vstup do interpretera začal využívať výstup ďalšieho modulu bolo nutné otestovať hlavne spoluprácu týchto dvoch častí.

Implementácia serverovej časti a aplikačného rozhrania, ktorý komunikuje s databázou boli v prvom rade testovaná s využitím nástroja Postman[1]. Nástrojom je možné zasielanie požiadaviek na koncové body aplikačných rozhraní, čím je možné postupne otestovať ich požadovanú funkcionalitu. Odpovede na požiadavky sa zobrazujú vo formáte JSON.

Po samostatnom otestovaní odpovedí jednotlivých častí API a následnej implementácie komunikácie medzi klientskou časťou bola testovaná funkcionalita koncových bodov aj pomocou volaní z klientskej časti systému. Pred nasadením aplikácie bola lokálna databáza presunutá na Microsoft Azure. Testovanie koncových bodov v tomto prípade odhalilo chybu synchronizácie času na serveroch databáze voči času webového systému.

Serverová aj klientska časť systému bola nasadená pomocou platformy Heroku[2], ktorá ponúka bezplatné hosťovanie a správu Node.js aplikácií. Klientská časť je pre demonštračné a testovacie účely prístupná cez voľne dostupnú webovú adresu[3].

8.1 Užívateľské testovanie

Užívateľské testovanie zamerané na použiteľnosť, odpovedá na otázky ako užívateľ pracuje s testovaným produktom. Faktor, ktorým je možné merať použiteľnosť predstavuje mentálne úsilie vynaložené používateľom pri práci s aplikáciou. Inými slovami, čím menej musí používateľ premýšľať pri práci s produktom, o to lepšia je jeho výsledná použiteľnosť [12].

Pri testovaní použiteľnosti je najpopulárnejšou metódou produkt testovať priamo s užívateľom. Vidieť ako užívateľ pracuje s produktom na vlastné oči vedie k lepšiemu pochopeniu kľúčových aspektov na zlepšenie. V dnešnej dobe je čoraz populárnejšie vzdialené testovanie, pri ktorom užívateľ pracuje s produktom bez priamej asistencie. Proces a postup je riadený pokynmi z predom napísaného scenára alebo formulára.

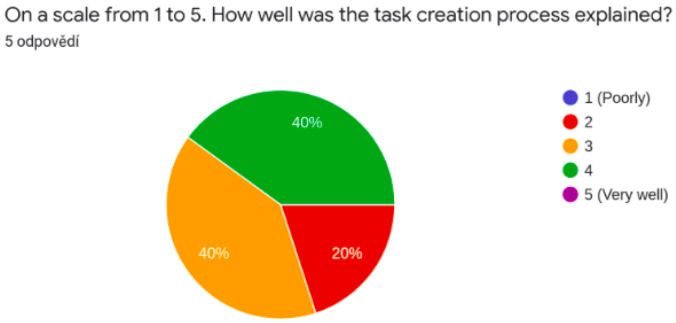
[1] Postman – https://www.postman.com/
[2] Heroku – https://www.heroku.com/
[3] https://xcziro00-bp.herokuapp.com/

Steve Krug vo svojej knihe [9], popisuje spôsob vzdialeného testovania stránkou, na ktorej je používateľovi sprístupnený formulár s pokynmi a pomocou nahrávania obrazu je zachytená práca užívateľa s produktom.

V práci som sa rozhodol časť nahrávania obrazovky vynechať z dôvodu minimalizácie pípravy potrebnej zo strany užívateľov. Zameral som sa na vzdialené užívateľské testovanie pomocou formulára, v ktorom užívateľ otestuje základnú funkcionalitu systému. Úvod obsahuje krátke predstavenie produktu pre užívateľa, stručný popis postupu pri testovaní a odkaz na stránku. Po otvorení odkazu sa užívateľ ďalej riadi pokynmi vo formulári a odpovedá na otázky. Za každým pokynom nasleduje otázka na zhodnotenie náročnosti danej akcie v systéme. Testovací formulár nevyžaduje vyplnenie žiadnych osobných údajov.

8.2 Výsledky a smer ďalšieho vývoja

Užívateľské testovanie, ktorého sa zúčastnilo 5 osôb rozličnej úrovni technickej zdatnosti, neprinieslo príliš pozitívne výsledky. Najmä z hľadiska užívateľskej prívetivosti procesu vytvárania príkladov. Ukážka 8.1 znázorňuje výsledky hodnotenia náročnosti vytvárania príkladov. Hlavným nedostatkom bol ťažko popísaný postup pri vytváraní príkladov. Avšak aj po detailnejšom vysvetlení základného príkladu, užívateľovi nebol jasný koncept vytvárania príkladov.



Obr. 8.1: Výsledný graf hodnotenia pre otázku "Na škále od 1 po 5. Ako by ste hodnotili vysvetlenie procesu vytvárania príkladu?"

Počas testovania bolo zistených niekoľko menších chýb, ktoré boli následne upravené. Jednému užívateľovi nebolo jasné, ktoré tlačítko použiť pri uložení výsledkov testu pri jeho ukončení, čo bolo vyriešené odstránením tlačítka pre uloženie odpovedí a bolo ponechané iba tlačítko pre ukončenie testu, ktoré zároveň ukladá odpovede študenta. Pri vytváraní príkladu bolo ďalším užívateľom zistené, že nie je možné zadať zápornú hodnotu pre rozsah premennej a nekontroluje sa prekrytie hodnôt rozsahu premennej. Tieto nedostatky boli následne v systéme upravené.

Ďalší smer vývoja klientskej časti systému vidím vo vylepšení užívateľskej skúsenosti. Konkrétne sa jedná o vylepšenie užívateľského rozhrania a doplnenie pomocných funkcií ako je automatické dopĺňanie názvov premenných, lepšie popísanie procesu tvorby príkladu v nápovede a pridanie funkcionality ako rýchla voľba vytvoriť nový príklad na základe už existujúceho príkladu. Na serverovej strane systému, by sa mohol vylepšiť spôsob zápisu príkladu, prejsť na databázu typu NoSQL alebo previesť JavaScriptový kód interpretera na kód v jazyku TypeScript.

Kapitola 9

Záver

Cieľom práce bolo uľahčiť tvorbu a vyhodnocovanie matematických online testov automatizovaním procesu vytvárania ich príkladov a následného využitia pri zostavovaní testov. Výsledkom práce je jednoduchý webový systém pre vytváranie matematických príkladov a testov pre základné školy. Zápis a vytváranie príkladov umožňuje navrhnutý jazyk a jeho interpreter, implementovaný v jazyku JavaScript.

V práci som v teoretickej časti previedol analýzu a porovnanie podobných webových systémov a nástrojov ktoré umožňujú generovať matematické príklady a testy. Po tejto fáze som navrhol jazyk, ktorý umožňuje vytvárať zápisy predstavujúce šablóny matematických príkladov a implementoval interpreter, ktorý dané šablóny spracováva a generuje z nich príklady s rôznymi hodnotami. Ďalej som v práci naštudoval v súčasnosti používané webové technológie. Následne som navrhol a zrealizoval jednoduchý webový systém, umožňujúci učiteľom vytvárať matematické príklady a testy. Testy môžu rozdať študentom a sledovať ich dosiahnuté výsledky. Žiaci si v systéme môžu pridelené testy vyplniť, ktoré sa automaticky vyhodnotia. Implementácia klientskej časti využíva technológie z ekosystému Vue.js a knižnicu Vuetify. Serverová časť systému, predstavujúca aplikačné rozhranie, využíva technológie Node.js a komunikuje s klientskou časťou a MySQL databázou. Po implementácii bol systém sprístupnený na verejnej URL adrese a testovaný malou skupinou užívateľov formou dotazníka. Na základe ich spätnej väzby boli prevedené opravy menších chýb systému.

Systém ponúka množstvo potenciálu pre jeho rozšírenie a vylepšenie v budúcnosti. Práca by mohla pokračovať rozšírením jazyka o zápis viacerých typov príkladov, pridaním pokročilejších funkcií do procesu tvorby príkladov a testov, ako napríklad automatické dopĺňanie názvov premenných v texte príkladu, ohraničenie rozsahu správnych výsledkov alebo doplnenie komplexnejších možností pre vytváranie testov. Možnosť vylepšenia práce vidím aj vo využití jazyka TypeScript pre implementáciu interpreteru a užívateľsky prívetivejšie formátovanie zadaní príkladov a odpovedí.

Napriek tomu je výsledný systém v súčasnom stave využiteľný na stanovené účely a počas práce ma obohatil o cenné znalosti z oblasti vývoja webových aplikácií a kritérií pre dobrú použiteľnosť.

Literatúra

[1] *Lodash* [online]. 2022 [cit. 2022-04-24]. Dostupné z: https://lodash.com/.

[2] Aggarwal, S. Modern web-development using reactjs. *International Journal of Recent Research Aspects*. 2018, zv. 5, č. 1, s. 133–137.

[3] Bauer, M. *Hello Framework! A heuristic method for choosing front-end JavaScript frameworks*. 2021.

[4] De Jong, J. a Mansfield, E. Math. js: An advanced mathematics library for javascript. *Computing in Science & Engineering*. IEEE. 2018, zv. 20, č. 1, s. 20–32.

[5] Filipova, O. *Learning Vue. js 2*. Packt Publishing Ltd, 2016.

[6] Flanagan, D. *JavaScript: the definitive guide*. O'Reilly Media, Inc, 2013.

[7] Halliday, P. *Vue. js 2 Design Patterns and Best Practices: Build enterprise-ready, modular Vue. js applications with Vuex and Nuxt*. Packt Publishing Ltd, 2018.

[8] Krug, S. *Don't make me think!: a common sense approach to Web usability*. Pearson Education India, 2000.

[9] Krug, S. *Rocket surgery made easy: The do-it-yourself guide to finding and fixing usability problems*. New Riders, 2009.

[10] Macrae, C. *Vue. js: up and running: building accessible and performant web apps*. Ö'Reilly Media, Inc.", 2018.

[11] Mak, R. *Writing compilers and interpreters: a software engineering approach*. John Wiley & Sons, 2011.

[12] Marsh, J. *UX for beginners: A crash course in 100 short lessons*. Ö'Reilly Media, Inc.", 2015.

[13] Oksanen, M. Cross-platform UI Development: React vs Svelte. 2021.

[14] Passaglia, A. *Vue. js 2 Cookbook*. Packt Publishing Ltd, 2017.

[15] Pereira, C. R. Working with SQL Databases. In: *Building APIs with Node. js*. Springer, 2016, s. 27–36.

[16] Saks, E. JavaScript Frameworks: Angular vs React vs Vue. 2019.

[17] Sarjeant, J. J. J. *Axios* [online]. 2022 [cit. 2022-04-24]. Dostupné z: https://axios-http.com/.

[18] Tidwell, J. *Designing interfaces: Patterns for effective interaction design.* Ö'Reilly Media, Inc.", 2010.

[19] Vuetify. *Vuetify — A Material Design Framework for Vue.js.* 2022 [cit. 2022-04-24]. Dostupné z: https://vuetifyjs.com/en.

[20] Watson, D. Compilers and Interpreters. In: *A Practical Approach to Compiler Construction.* Springer, 2017, s. 13–36.

Príloha A

Plagát

Obr. A.1: Plagát webového systému

Príloha B

Obsah priloženého pamäťového média

Priložený DVD disk obsahuje:

- `src/` - zložka so zdrojovými súbormi.
- `text_src/` - zložka so zdrojovými súbormi tejto technickej správy.
- `plagat.svg` - A2 plagát práce vo formáte SVG.
- `plagat.png` - A2 plagát práce vo formáte PNG.
- `bp30sec.mp4` - krátke video demonštrujúce prácu.
- `xcziro00-bp.pdf` - technická správa práce v PDF formáte.
- `README.md` - návod k lokálnemu spusteniu