

**SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE**  
**FAKULTA ELEKTROTECHNIKY A INFORMATIKY**

Evidenčné číslo: FEI-16605-111184

**MATEMATICKÝ TRENAŽÉR**  
**BAKALÁRSKA PRÁCA**

**2025**

**Bence Bodnár**

**SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE**  
**FAKULTA ELEKTROTECHNIKY A INFORMATIKY**

Evidenčné číslo: FEI-16605-111184

**MATEMATICKÝ TRENAŽÉR**  
**BAKALÁRSKA PRÁCA**

Študijný program: Aplikovaná informatika  
Názov študijného odboru: Informatika  
Školiace pracovisko: Ústav informatiky a matematiky  
Vedúci záverečnej práce: doc. RNDr. Oľga Nánásiová, PhD.

**Bratislava 2025**

**Bence Bodnár**

# SÚHRN

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE  
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Študijný program:	Aplikovaná informatika
Autor:	Bence Bodnár
Bakalárska práca:	Matematický trenažér
Vedúci záverečnej práce:	doc. RNDr. Olga Nánásiová, PhD.
Miesto a rok predloženia práce:	Bratislava 2025

V tejto bakalárskej práci sa zaoberáme vývojom trojvrstvovej webovej aplikácie zameranej na e-learning matematiky, konkrétne pravdepodobnosti a štatistiky. Cieľom práce bolo navrhnúť a implementovať užívateľsky orientovaný frontend pomocou Angular frameworku, pričom sú využívané knižnice Bootstrap a Material UI na zabezpečenie intuitívneho rozhrania. Na druhej strane, backend aplikácie bol vyvinutý pomocou Node.js a frameworku Next.js s cieľom poskytnúť efektívne spracovanie dát a logiky aplikácie. S PostgreSQL databázou sme pracovali na ukladaní a spracovaní užívateľských dát a obsahu. Celá aplikácia je nakoniec nasadená v Docker kontajneroch, čo umožňuje jednoduchšiu distribúciu a nasadenie aplikácie. Výsledkom je komplexná e-learningová platforma, ktorá umožňuje študentom testovať svoje znalosti prostredníctvom testov, úloh a študijných materiálov, a tiež analyzovať ich pokrok a vývoj. Tento projekt predstavuje dôležitý krok smerom k moderným pedagogickým metódam, ktoré využívajú technologické inovácie na zlepšenie vzdelávania.

Kľúčové slová: Docker, PostgreSQL, Framework, Node.js, Angular, Pravdepodobnosť

# ABSTRACT

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA

FACULTY OF ELECTRICAL ENGINEERING AND INFORMATION TECHNOLOGY

Study Programme:	Applied Informatics
Author:	Bence Bodnár
Bachelor's thesis:	Mathematics trainer
Supervisor:	doc. RNDr. Oľga Nánásiová, PhD.
Place and year of submission:	Bratislava 2025

In this bachelor thesis we are developing a three-layer web application focused for e-learning mathematics, specifically probability and statistics. The aim of the work was to design and implement a user-oriented frontend using the Angular framework, using the Bootstrap and Material UI libraries to provide an intuitive editing. On the other hand, the backend of the application was developed using Node.js and the framework Next.js framework in order to provide efficient data processing and application logic. With PostgreSQL database, we worked on storing and processing user data and content. The entire appli- Finally, the entire application is deployed in Docker containers, which allows for easier distribution and deployment of the application. The result is a comprehensive e-learning platform that enables learners to test their knowledge through tests, assignments and study materials, and also analyse their progress and development. This project represents an important step towards modern pedagogical methods that use technological innovation to improve education.

Keywords: Docker, PostgreSQL, Framework, Node.js, Angular, Probability

# Podakovanie

Podakovanie patrí mojej školiteľke doc. RNDr. Oľga Nánásiová, PhD. za poskytnutie poznatkov z oblasti, odborné konzultácie a čas, ktorý mi venovala pri vypracovaní mojej záverečnej práce.

# Zoznam skratiek

<b>AJAX</b>	Asynchronous JavaScript and XML
<b>API</b>	Rozhranie pre programovanie aplikácií
<b>CLI</b>	Command Line Interface
<b>CSS</b>	Kaskádové štýly (Cascading Style Sheets)
<b>DOM</b>	Document Object Model
<b>HTML</b>	Hypertextový značkovací jazyk (HyperText Markup Language)
<b>HTTP</b>	Hypertextový prenosový protokol (Hypertext Transfer Protocol)
<b>LDAP</b>	Lightweight Directory Access Protocol
<b>NPM</b>	Node Package Manager (Správca balíkov pre Node.js)
<b>ORDBMS</b>	Object-Relational Database Management System
<b>RDBMS</b>	Relational Database Management System
<b>REST</b>	Representational State Transfer
<b>RxJs</b>	Reactive Extensions for JavaScript
<b>SCSS</b>	Sassy CSS
<b>SPA</b>	Single Page Application
<b>SQL</b>	Štruktúrovaný dopytovací jazyk (Structured Query Language)
<b>TLS</b>	Transport Layer Security
<b>UI</b>	Užívateľské rozhranie (User Interface)
<b>URL</b>	Uniform Resource Locator
<b>UX</b>	Užívateľský zážitok (User Experience)

# Obsah

<b>Úvod</b>	<b>1</b>
<b>1 Analýza</b>	<b>2</b>
1.1 Brilliant.org . . . . .	2
1.2 Khan Academy . . . . .	2
1.3 Vieme matiku . . . . .	3
1.4 Zhodnotenie . . . . .	3
<b>2 Použité technológie a knižnice</b>	<b>4</b>
2.1 Frontend . . . . .	4
2.1.1 HTML . . . . .	4
2.1.2 CSS . . . . .	5
2.1.3 SCSS . . . . .	5
2.1.4 JavaScript . . . . .	5
2.1.5 AJAX . . . . .	6
2.1.6 jQuery . . . . .	6
2.1.7 java . . . . .	6
2.1.8 MathJax . . . . .	7
2.1.9 MathQuill . . . . .	7
2.1.10 ApexCharts . . . . .	7
2.2 Backend . . . . .	7
2.2.1 API . . . . .	8
2.2.2 REST . . . . .	8
2.2.3 CORS . . . . .	8
2.2.4 Node.js . . . . .	9
2.3 Databázové systémy . . . . .	9
2.3.1 SQL . . . . .	9
2.3.2 PostgreSQL . . . . .	10
2.3.3 DBDiagram . . . . .	10
2.4 Framework . . . . .	10
2.4.1 Frontendové frameworky . . . . .	11
2.4.2 Rozdiel medzi CSS a JavaScript (JS) frameworkami . . . . .	11
2.4.3 Angular . . . . .	11
2.4.4 RxJs . . . . .	11

2.4.5	Backendové frameworky . . . . .	12
2.4.6	Express.js . . . . .	12
2.5	UI a UX . . . . .	13
2.5.1	Figma . . . . .	13
2.5.2	Material UI . . . . .	13
2.6	Gamifikácia . . . . .	13
2.6.1	Gamifikácia v e-learningu . . . . .	14
2.7	Server . . . . .	14
2.7.1	NGINX . . . . .	14
2.7.2	Docker . . . . .	14
2.8	GIT . . . . .	15
<b>3</b>	<b>Návrh webovej aplikácie</b>	<b>16</b>
3.1	Špecifikácia . . . . .	17
3.2	Funkcionálne požiadavky . . . . .	17
3.2.1	Pre študentov . . . . .	18
3.2.2	Pre administrátorov (učiteľov) . . . . .	18
3.3	Nefunkcionálne požiadavky . . . . .	18
3.4	Diagramy prípadov použitia . . . . .	20
3.4.1	Diagramy prípadov použitia pre študenta . . . . .	20
3.4.2	Diagramy prípadov použitia pre administrátora . . . . .	21
3.5	Databáza . . . . .	21
3.6	Používateľské rozhranie . . . . .	23
3.7	Preskúšanie sa . . . . .	26
3.7.1	Gamifikácia . . . . .	27
3.7.2	Generovanie úloh . . . . .	27
3.8	Materialy . . . . .	27
3.8.1	Interaktívne grafy . . . . .	27
3.9	Vytváranie úloh (Admin) . . . . .	27
3.10	Vyhodnocovanie úloh . . . . .	27
<b>4</b>	<b>Implementácia webovej aplikácie</b>	<b>28</b>
4.1	Vývojové prostredie a inštalácia závislostí . . . . .	28
4.2	Frontend – Angular . . . . .	28
4.2.1	Grafická časť tejto PICOVINY . . . . .	31
4.3	Backend . . . . .	31



4.3.1	Autentifikácia . . . . .	33
4.3.2	Posielanie dat ??? . . . . .	36
4.4	DB . . . . .	36
4.5	Docker . . . . .	36
<b>5</b>	<b>Testovanie</b>	<b>37</b>
<b>6</b>	<b>Záver</b>	<b>38</b>
	<b>Záver</b>	<b>39</b>
	<b>Zoznam použitej literatúry</b>	<b>40</b>

# Zoznam obrázkov a tabuliek

Obrázok 1	Diagram prípadov použitia pre študenta. . . . .	20
Obrázok 2	Diagram prípadov použitia pre administrátora. . . . .	21
Obrázok 3	Logický ER diagram (Entity-Relationship Diagram) databázového modelu. . . . .	22
Obrázok 4	Príklad zápisu JSON dát v stĺpci <i>answers</i> . . . . .	23
Obrázok 5	Ukážka LaTeX zápisu zadania úlohy v databáze . . . . .	23
Obrázok 6	Hlavná obrazovka aplikácie. . . . .	24
Obrázok 7	Výrez z konceptuálneho návrhu podstránky Písať test. . . . .	25
Obrázok 8	Používateľské rozhranie stránky pre vytvorenie testovacej sady. .	26
Obrázok 9	Výstup príkazu <code>ng version</code> v termináli. . . . .	29
Obrázok 10	Základná štruktúra Angular projektu. . . . .	30
Obrázok 11	Ukážka štruktúry backendových služieb v aplikácii. . . . .	32
Tabuľka 1	Vzdelávacie platformy . . . . .	4

# Zoznam algoritmov

# Zoznam výpisov

1	Ukážka základnej konfigurácie Express.js servera . . . . .	31
2	Konfigurácia CORS pre komunikáciu medzi FE a BE . . . . .	32
3	Použitie body-parser pre spracovanie JSON požiadaviek . . . . .	32
4	Funkcia na autentifikáciu používateľa cez LDAP . . . . .	33
5	Spracovanie prihlasovania a uloženie používateľa do session . . . . .	33
6	Implementácia AuthGuard v Angulari . . . . .	34
7	Definícia rout v Angulari s ochranou pomocou AuthGuard . . . . .	35

# Úvod

Štúdium matematickej štatistiky a pravdepodobnosti je kritické pre porozumenie a analyzovanie náhodných javov a dát v rôznych oblastiach, ako napríklad v ekonómii, vedeckom výskume alebo medicíne. Tieto oblasti matematiky zahŕňajú širokú škálu tém, vrátane modusu, mediánu, stredovej hodnoty, náhodných premenných, kombinatoriky a podmienenej pravdepodobnosti. Každá z týchto tém poskytuje unikátne nástroje na kvantifikáciu a analýzu dátových súborov, čo je kľúčové pre predpovedanie a porozumenie rôznym javom a trendom. S cieľom podporiť systematické vzdelávanie v týchto dôležitých oblastiach sme sa rozhodli vyvinúť trojvrstvovú webovú aplikáciu. Táto aplikácia, využívajúca moderné technológie ako Angular framework s knižnicami Bootstrap a Material UI pre frontend, a Node.js s frameworkom Next.js pre backend, sa zameriava na poskytovanie interaktívnych učebných materiálov, testov a študijných materiálov v týchto matematických témach. Cieľom tejto práce je nielen uľahčiť proces učenia sa matematickej štatistiky a pravdepodobnosti, ale aj zlepšiť zrozumiteľnosť a prístupnosť týchto konceptov pre študentov. Naša aplikácia má ambíciu prispieť k zvýšeniu efektivity vzdelávacieho procesu v tejto dôležitej matematickej oblasti a poskytnúť študentom moderný a efektívny nástroj na zlepšenie ich matematických schopností a analytického myslenia.

# 1 Analýza

V tejto kapitole sa venujeme rozboru dostupných platforiem pre e-learning matematiky. Cieľom je identifikovať platformy, porovnať ich funkcie a odhaliť medzery, ktoré naša webová aplikácia môže vyplniť. Na trhu existuje široká škála platforiem pre e-learning rôznych matematických tém, z ktorých každá ponúka rôzne riešenia, funkcie a zameriava sa na odlišné cieľové skupiny.

## 1.1 Brilliant.org

Brilliant.org je online vzdelávacia platforma zameraná na interaktívne kurzy v oblastiach matematiky, vedy a počítačovej vedy. Je navrhnutá tak, aby podporovala aktívne učenie prostredníctvom riešenia problémov a interaktívnych výziev, čím pomáha študentom rozvíjať kritické myslenie a logické schopnosti. Platforma ponúka viac ako 60 kurzov, ktoré sú prispôsobené rôznym úrovňam znalostí, od začiatočníkov po pokročilých.

Medzi jej hlavné výhody patria interaktívne lekcie, ktoré sú navrhnuté tak, aby boli pútavé a vyžadovali aktívnu účasť študentov, čím zvyšujú efektivitu učenia. Umožňuje tiež flexibilné a samostatné štúdium, čo je ideálne pre individuálne potreby. Platforma ponúka denné výzvy na rôzne témy, ktoré pomáhajú udržiavať študentov motivovaných a neustále zapojených do procesu učenia. Nevýhodou je, že táto platforma je platená a dostupná len v anglickom jazyku, čo môže predstavovať prekážku pre niektorých študentov. [1]

## 1.2 Khan Academy

Platforma Khan Academy ponúka bezplatné videokurzy a interaktívne cvičenia z rôznych oblastí matematiky, vrátane vysokoškolskej štatistiky a pravdepodobnosti. Je vhodná pre študentov základných aj vysokých škôl. Medzi jej výhody patrí široká škála obsahu, jednoduché použitie a dostupnosť pre rôzne úrovne znalostí. Dostupné zdroje k daným témam sú prehľadné a dobre štruktúrované. Ponúka taktiež možnosť sledovania pokroku, získavania bodov a odznakov za splnené kapitoly, čím motivuje študentov k učeniu prostredníctvom gamifikácie<sup>1</sup>. Nevýhodou je, že je dostupná len v anglickom jazyku, čo môže byť pre niektorých študentov prekážkou. Používateľské rozhranie môže byť z dôvodu množstva obsahu pre niektorých používateľov neprehľadné, najmä ak sa na platforme nachádzajú prvýkrát. Napriek týmto nedostatkom je platforma považovaná za jeden z najlepších nástrojov na online vzdelávanie a sebarozvoj. [2]

---

<sup>1</sup>Gamifikácia je využitie herných prvkov a mechaník v nehermom prostredí s cieľom zvýšiť motiváciu, zapojenie a efektivitu používateľov.

## 1.3 Vieme matiku

Najpopulárnejším slovenským portálom pre e-learning matematiky je Vieme matiku. Táto platforma ponúka rôzne kurzy a cvičenia z matematiky pre žiakov základných a stredných škôl. Medzi jej výhody patrí dostupnosť pre slovenských žiakov, široký výber tém, rôzne formy precvičovania, do ktorých patrí grafické znázornenie úloh a možnosť sledovania pokroku. Ponúka taktiež hravé prvky, ako sú grafické a zvukové efekty, ktoré môžu zvýšiť motiváciu žiakov. Vyznačuje sa taktiež jednoduchým použitím a prehľadným rozhraním. Nevýhodou je, že nie je dostupná pre študentov mimo Slovenska a je podporovaná len v slovenčine. Platforma slúži na precvičovanie matematických úloh, ale neponúka zdroje pre samostatné štúdium alebo nápovedy. Zároveň, v prípade, že by sme chceli naplno využiť všetky jej funkcie, by bolo potrebné si zakúpiť licenciu. [3]

## 1.4 Zhodnotenie

Počas analýzy existujúcich vzdelávacích platforiem sme zistili, že na trhu chýbajú lokalizované a cenovo dostupné e-learningové riešenia pre stredoškolských a vysokoškolských študentov, ktoré by efektívne kombinovali gamifikáciu, interaktivitu a prehľadné rozhranie. Existujúce platformy, ako Brilliant.org a Khan Academy, ponúkajú kvalitné vzdelávacie materiály, ale ich dostupnosť je limitovaná anglickým jazykom a v prípade Brilliant.org aj plateným modelom. Vieme Matiku síce poskytuje lokalizovaný obsah, ale nezohľadňuje pokročilé potreby samostatného štúdia a je obmedzená na úzky okruh používateľov. Analyzované platformy ukazujú širokú škálu prístupov, pričom mnohé sa zameriavajú na riešenie komplexných úloh alebo tradičné formy vzdelávania. Tieto prístupy však často nekladú dôraz na intuitívne osvojovanie matematických konceptov a podporu samostatného učenia. Tieto poznatky nám umožňujú identifikovať medzery a formulovať jasné požiadavky na vývoj novej aplikácie, ktorá by ponúkala lokalizovaný obsah, interaktívne učenie a dostupnosť pre rôzne cieľové skupiny.

Tabuľka 1: Vzdelávacie platformy

Platforma	Funkcie	Cieľová skupina	Cena
Khan Academy	Videokurzy, cvičenia	Všetky úrovne	Bezplatná
Brilliant.org	Gamifikované kurzy	Stredné a Vysoké školy	Platená
Vieme Matiku	Online kurzy matematiky	Základné a Stredné školy	Čiastočne bezplatná

## 2 Použité technológie a knižnice

V tejto kapitole sa podrobne venujeme technológiám a knižniciam, ktoré plánujeme použiť na vývoj webovej aplikácie pre e-learning matematickej štatistiky a pravdepodobnosti. Výber technológií je založený na princípoch flexibility, kompatibility, bezpečnosti a aktívnej komunity vývojárov.

### 2.1 Frontend

Frontend je časť softvérového vývoja, ktorá sa zaoberá tým, čo používateľ vidí a s čím interaguje pri práci s aplikáciou alebo webovou stránkou. Ide o viditeľnú vrstvu aplikácie, ktorá zahŕňa všetky prvky používateľského rozhrania (UI) a je priamo zodpovedná za používateľskú skúsenosť (UX).

V kontexte nášho webového vývoja predstavuje frontend technológie a nástroje používané na tvorbu webových stránok, ktoré sú dostupné a vykresľované v internetových prehliadačoch. Zahŕňa návrh, implementáciu a optimalizáciu používateľského rozhrania tak, aby bolo esteticky príťažlivé, funkčné a dostupné na rôznych zariadeniach a platformách.

#### 2.1.1 HTML

Hypertextový značkovací jazyk (HyperText Markup Language) (HTML) je značkovací jazyk používaný na tvorbu a štruktúrovanie obsahu webových stránok. Umožňuje definovať rôzne prvky, ako sú nadpisy, odseky, obrázky či odkazy, čím určuje základnú kostru a vzhľad webovej stránky. Napriek častým mylným predstavám, HTML nie je programovací jazyk, keďže neumožňuje vytvárať podmienené logické operácie alebo funkcie. Jeho hlavnou úlohou je prezentácia a organizácia obsahu pre webové prehliadače. [4]



### 2.1.2 CSS

Kaskádové štýly (Cascading Style Sheets) (CSS) [5] je štýlovací jazyk používaný na definovanie vzhľadu a formátovania webových stránok. Umožňuje oddeliť vizuálnu prezentáciu od štruktúry obsahu definovanej v HTML, čím zjednodušuje údržbu a aktualizáciu dizajnu. Pomocou CSS je možné nastaviť rôzne vizuálne vlastnosti, ako sú farby, písma, veľkosti, rozloženie prvkov a ďalšie aspekty dizajnu. Taktiež podporuje tvorbu responzívnych dizajnov, ktoré sa prispôbujú rôznym zariadeniam a veľkostiam obrazoviek. Moderné techniky, ako flexbox a grid, umožňujú presné rozmiestnenie a zarovnanie prvkov na stránke, čo je užitočné pri tvorbe komplexných rozložení.

### 2.1.3 SCSS

Sassy CSS (SCSS) je rozšírenie jazyka CSS, ktoré pridáva pokročilé funkcie pre efektívnejšie štýlovanie webových stránok. SCSS umožňuje používať premenné, vnáranie selektorov, mixiny, funkcie a operácie, čím zjednodušuje správu a údržbu štýlov. Vďaka svojim vlastnostiam podporuje modulárny prístup k tvorbe štýlov, čím zlepšuje čitateľnosť kódu a urýchľuje vývoj.

SCSS používa štandardnú CSS syntax s doplnením nových funkcií, čo zabezpečuje spätnú kompatibilitu. Kód napísaný v SCSS sa následne kompiluje do klasického CSS, ktoré podporujú všetky moderné prehliadače. Tento proces zvyšuje flexibilitu vývoja a umožňuje tvorbu komplexných štýlových štruktúr.[6]

### 2.1.4 JavaScript

JavaScript [7] je interpretovaný programovací jazyk ktorý umožňuje dynamickú interakciu s používateľom a zmeny obsahu webových stránok bez nutnosti ich opätovného načítania.

Podporuje objektovo orientované programovanie s triedami, objektmi a metódami, čo umožňuje tvorbu komplexných aplikácií. Vďaka svojej dynamickej povahe dokáže meniť obsah a štruktúru stránky počas jej behu.

Medzi jeho funkcie patrí funkcionálne programovanie, kde sú funkcie považované za prvotriedne objekty, a programovanie riadené udalosťami, ktoré umožňuje reagovať na interakcie používateľa, napríklad na kliknutia.

Je multiplatformový a podporuje rôzne zariadenia, ako sú počítače, smartfóny a tablety. Populárne knižnice a rámce ako jQuery, React, Angular a Vue výrazne uľahčujú vývoj aplikácií.

Medzi jeho vlastnosti patrí manipulácia s Document Object Model (DOM), spracovanie

udalostí, manipulácia s dátami a podpora asynchrónnych volaní na server pomocou techniky Asynchronous JavaScript and XML (AJAX). Tieto vlastnosti z neho robia základný nástroj na tvorbu moderných webových aplikácií.

### **2.1.5 AJAX**

AJAX (Asynchronous JavaScript and XML) je technológia, ktorá umožňuje webovým aplikáciám načítavať a odosielať dáta na pozadí bez nutnosti obnoviť celú stránku. Funguje na kombinácii známych technológií – HTML, CSS, JavaScript a XMLHttpRequest (alebo moderného fetch) – a výrazne zlepšuje interaktivitu a používateľskú skúsenosť. AJAX sa využíva napríklad v ankete bez reloadu, našepťávačoch alebo moderných single-page aplikáciách.[8]

### **2.1.6 jQuery**

jQuery je open-source knižnica napísaná v JavaScripte, ktorá uľahčuje prácu s HTML dokumentom, manipuláciu s DOM, prácu s udalosťami, animáciami a technológiou AJAX. Bola vytvorená v roku 2006 s cieľom zjednodušiť vývoj v čase, keď neexistovala jednotná podpora v prehliadačoch. Aj napriek tomu, že je dnes považovaná za technicky zastaranú a dátovo náročnú, stále sa používa na mnohých weboch vďaka svojej jednoduchosti a veľkému množstvu dostupných pluginov.[9]

### **2.1.7 java**

TypeScript [10] je programovací jazyk vyvinutý spoločnosťou Microsoft, ktorý rozširuje možnosti JavaScriptu pridaním statického typovania a pokročilých objektovo orientovaných prvkov. Tým umožňuje vývojárom identifikovať chyby už počas vývoja, čo zvyšuje spoľahlivosť a udržiavateľnosť kódu. TypeScript je nadmnožinou JavaScriptu, čo znamená, že všetok platný kód v JavaScripte je kompatibilný s TypeScriptom. Po napísaní sa kód v TypeScripte transpiluje do štandardného JavaScriptu, ktorý je podporovaný vo všetkých moderných prehliadačoch. Tento prístup umožňuje využívať výhody moderných programovacích techník pri zachovaní širokej kompatibility a flexibility, ktorú JavaScript ponúka.

### 2.1.8 MathJax

MathJax [11] je open-source<sup>2</sup> JavaScriptový engine určený na zobrazovanie matematickej notácie, ako sú LaTeX, MathML a AsciiMath, v moderných webových prehliadačoch. Je navrhnutý tak, aby konsolidoval pokroky vo webových technológiách do jednotnej platformy pre matematiku na webe, podporujúc hlavné prehliadače a operačné systémy, vrátane mobilných zariadení. Používatelia nemusia inštalovať žiadne doplnky ani softvér; stačí, aby autor stránky zahrnul MathJax a matematický obsah do webovej stránky, a MathJax sa o zvyšok postará.

### 2.1.9 MathQuill

MathQuill [12] je open-source webový editor matematických výrazov navrhnutý tak, aby umožňoval jednoduché a estetické zadávanie matematiky priamo v prehliadači. Umožňuje používateľom zadávať a zobrazovať matematické výrazy v známej vizuálnej forme pomocou syntaxe podobnej LaTeXu.

Je obľúbený v edukatívnych a interaktívnych webových aplikáciách, pretože poskytuje dynamické pole, v ktorom možno editovať matematické vzorce s okamžitým vizuálnym výstupom. MathQuill podporuje rôzne režimy – ako zobrazenie statických výrazov, interaktívne polia pre vstup používateľa alebo čítanie/zápis LaTeX kódu.

### 2.1.10 ApexCharts

ApexCharts[13] je open-source JavaScriptová knižnica určená na vizualizáciu dát v podobe interaktívnych grafov. Poskytuje širokú škálu grafických typov a umožňuje ich jednoduchú integráciu do webových aplikácií s podporou populárnych frameworkov ako React, Angular či Vue. Vďaka svojej flexibilitě, responzivnosti a podpore dynamických dát sa často využíva pri tvorbe analytických nástrojov a interaktívnych dashboardov.

## 2.2 Backend

Backend predstavuje časť softvérovej aplikácie, ktorá nie je priamo prístupná používateľom; ide o serverovú stranu v klient-server architektúre. Zodpovedá za hlavnú funkcionálnu aplikáciu, vrátane spracovania webových požiadaviek, manipulácie s dátami a ich ukladania v databázach. Backend spolupracuje s frontendom, ktorý tvorí prezentačnú vrstvu aplikácie, s cieľom zabezpečiť komplexnú používateľskú skúsenosť. Dáta generované na backende sú

---

<sup>2</sup>Open-source je softvér, ktorého zdrojový kód je voľne dostupný, môže byť používaný, upravovaný a distribuovaný kýmkoľvek.

odosielané na frontend, kde sú prezentované používateľovi. Hoci sú často backend a frontend vyvíjané oddelenými tímami, hranica medzi nimi môže byť nejasná, čo vedie k prístupu známemu ako full-stack<sup>3</sup> development, kde programátori pracujú na oboch stranách aplikácie. V minulosti backend pozostával prevažne z jednoduchých serverových skriptov, avšak s rozvojom webových technológií sa využívajú pokročilé frameworky umožňujúce dynamickú generáciu obsahu. Efektívny backendový kód je kľúčový pre optimalizáciu výkonu aplikácie, minimalizáciu zataženia servera a databázy, a zabezpečenie rýchlej odozvy pre používateľov. [14]

### 2.2.1 API

Rozhranie pre programovanie aplikácií (API) je rozhranie, ktoré umožňuje jednej aplikácii komunikovať s inou aplikáciou alebo systémom. Umožňuje vývojárom využívať už existujúce funkcionality bez toho, aby ich museli znova vytvárať, čím urýchľuje a zjednodušuje vývoj softvéru. API definuje spôsob výmeny dát medzi klientom a serverom – napríklad keď mobilná aplikácia zobrazuje údaje o počasí získané zo servera.[15]

### 2.2.2 REST

REST API (Representational State Transfer (REST) acrfullapi) je typ webového API, ktorý umožňuje aplikáciám komunikovať medzi sebou pomocou štandardných HTTP protokolov. REST API využíva jednoduché operácie ako čítanie, zápis, úprava a mazanie dát prostredníctvom URL adries a HTTP metód.

Jeho výhodou je jednoduchá štruktúra, škálovateľnosť a nezávislosť medzi klientom a serverom – klient nemusí poznať vnútornú logiku servera a server nemusí vedieť nič o klientskom rozhraní. REST API je široko používané pri vývoji moderných webových a mobilných aplikácií, pretože umožňuje flexibilnú a efektívnu výmenu dát.

### 2.2.3 CORS

CORS (Cross-Origin Resource Sharing) predstavuje mechanizmus, ktorý umožňuje realizáciu HTTP požiadaviek medzi rôznymi doménami (tzv. cross-origin požiadavky). Ide o spôsob, ako prehliadače môžu bezpečne vykonávať požiadavky na zdroje umiestnené na iných serveroch, než odkiaľ pochádza pôvodná webová stránka. V minulosti takéto požiadavky bránila politika rovnakého pôvodu (same-origin policy), ktorá obmedzovala skripty bežiacie v prehliadači výhradne na komunikáciu so serverom tej istej domény.

---

<sup>3</sup>Full-stack vývoj je proces, kde vývojár pracuje na frontende aj backende, teda na používateľskom rozhraní aj serverovej logike.

Mechanizmus CORS funguje tak, že server explicitne definuje, kto môže vytvárať požiadavky na jeho API, aké typy požiadaviek sú povolené a aké HTTP metódy sú podporované. Toto sa realizuje prostredníctvom špeciálnych HTTP hlavičiek, ktoré umožňujú prehliadaču určiť, či konkrétna požiadavka medzi rôznymi doménami môže byť bezpečne vykonaná. [16]

### 2.2.4 Node.js

Node.js je runtime<sup>4</sup> prostredie umožňujúce použitie JavaScriptu na strane servera, pričom využíva flexibilitu a jednoduchosť tohto programovacieho jazyka. Výhodou JavaScriptu sú jeho pokročilé koncepty ako „first-class functions“ (funkcie prvej triedy) a closures, ktoré umožňujú vytváranie efektívnych webových aplikácií. Hoci JavaScript býva kritizovaný za nespoľahlivosť, táto kritika vyplýva predovšetkým zo zvláštností DOM-u v prehliadačoch, nie zo samotného jazyka.

## 2.3 Databázové systémy

Databázový systém je softvérové riešenie, ktoré slúži na efektívne uchovávanie, správu a vyhľadávanie dát. Umožňuje vytvárať databázy, ktoré obsahujú štruktúrované dáta usporiadané v tabuľkách, a poskytuje nástroje na ich úpravu, triedenie, filtrovanie a prezentáciu.

Databáza samotná je organizovaný súbor údajov, často rozdelený do tabuliek, ktoré spolu súvisia a medzi ktorými môžu byť definované vzťahy. Cieľom databázového systému je zabezpečiť integritu, konzistenciu a dostupnosť dát, pričom používateľ môže s dátami pracovať pomocou formulárov, dotazov či zostáv.[17]

### 2.3.1 SQL

Štruktúrovaný dopytovací jazyk (Structured Query Language) (SQL) je jazyk navrhnutý na prácu s dátami v relačných databázach, pričom umožňuje ich efektívne ukladanie, organizáciu a získavanie. V SQL sa údaje ukladajú do tabuliek, ktoré fungujú ako dvojrozmerné štruktúry zložené z riadkov a stĺpcov. Každá tabuľka predstavuje určitý typ objektu (napr. zákazníci, produkty) a každý riadok v tabuľke predstavuje jeden záznam s konkrétnymi údajmi.

Pomocou dotazov (queries) je možné z databázy získať presne tie údaje, ktoré používateľ potrebuje – napríklad filtrovať podľa podmienok, zoskupovať dáta alebo ich spájať z

---

<sup>4</sup>Runtime prostredie je prostredie poskytujúce aplikácii všetky potrebné zdroje, knižnice a služby, ktoré sú potrebné na jej spustenie a beh.

viacerých tabuliek. Dotazy sú základom práce s SQL a umožňujú analyzovať, transformovať a prezentovať uložené informácie v požadovanej forme.[18]

### 2.3.2 PostgreSQL

PostgreSQL je pokročilý open-source objektovo-relačný databázový systém (ORDBMS), ktorý kombinuje tradičné vlastnosti relačných databáz so schopnosťou pracovať s objektovo orientovanými prvkami. To znamená, že okrem práce s tabuľkami a vzťahmi (typických pre Relational Database Management System (RDBMS)) podporuje aj vlastné dátové typy, dedičnosť, či ukladanie komplexných štruktúr, ako sú napríklad JSON či geografické dáta.

Ako ORDBMS, PostgreSQL umožňuje definovať vlastné funkcie, procedúry a typy, čím ponúka vysokú mieru prispôbitelnosti pre náročné aplikácie. Je navrhnutý s dôrazom na integritu dát, rozšíriteľnosť a štandardy SQL, a preto je široko používaný v podnikových riešeniach, výskumných systémoch aj moderných webových aplikáciách.[19]

### 2.3.3 DBDiagram

dbdiagram.io je bezplatný online nástroj, ktorý umožňuje vývojárom a dátovým analytikom jednoducho vytvárať diagramy a databázových štruktúr pomocou jednoduchého textového zápisu. Jeho hlavnou výhodou je schopnosť rýchlo a efektívne navrhovať a vizualizovať databázové schémy, čo uľahčuje plánovanie a komunikáciu v tímoch. Používatelia môžu definovať tabuľky, stĺpce a vzťahy medzi nimi v jednoduchom jazyku, pričom nástroj automaticky generuje zodpovedajúci diagram. Okrem toho dbdiagram.io podporuje import a export SQL skriptov, čo umožňuje integráciu s existujúcimi databázami a uľahčuje migráciu alebo dokumentáciu databázových štruktúr. Vďaka týmto funkciám je dbdiagram.io obľúbeným nástrojom pre rýchly návrh databáz a spoluprácu na databázových projektoch.

## 2.4 Framework

Framework je softvérová platforma poskytujúca vývojárom preddefinovanú štruktúru a nástroje, ktoré urýchľujú a zjednodušujú proces vývoja aplikácií. Existujú rôzne typy frameworkov podľa oblasti použitia, napríklad webové, desktopové či mobilné. Používanie frameworkov prináša výhody ako rýchlejší vývoj, lepšiu kvalitu kódu a standardizáciu práce vývojárov. [20]

### 2.4.1 Frontendové frameworky

Frontend framework je špecializovaný typ frameworku určený na vývoj používateľských rozhraní webových aplikácií. Na rozdiel od všeobecných frameworkov, ktoré môžu pokrývať celý vývojový cyklus aplikácie, frontend framework sa sústreďuje výhradne na klientskú časť – teda na to, čo používateľ vidí a s čím interaguje v prehliadači. Poskytuje nástroje a štruktúru na organizáciu komponentov, správu stavu aplikácie a manipuláciu s DOM-om, čím umožňuje tvorbu dynamických, responzívnych a udržiavateľných rozhraní.

### 2.4.2 Rozdiel medzi CSS a JavaScript (JS) frameworkami

- CSS frameworky (napr. Bootstrap, Tailwind CSS, Materialize) sú zamerané na vizuálnu stránku webu – poskytujú hotové štýly, rozloženia, komponenty (tlačidlá, formuláre, mriežky), ktoré zjednodušujú tvorbu konzistentného a estetického dizajnu. Ide predovšetkým o „vzhľad“ aplikácie.
- JS frameworky (napr. React, Angular, Vue.js) naopak poskytujú funkčnú logiku a štruktúru aplikácie. Umožňujú pracovať s dátami, dynamicky meniť obsah stránky bez opätovného načítania (tzv. SPA), pracovať s API a definovať správanie komponentov.[21]

### 2.4.3 Angular

Angular je open-source framework vyvinutý spoločnosťou Google, určený na tvorbu dynamických a responzívnych webových aplikácií. Umožňuje vývojárom vytvárať aplikácie s bohatou funkcionalitou prostredníctvom komponentovo orientovanej architektúry, ktorá podporuje opätovné použitie kódu a zjednodušuje údržbu aplikácií. Angular využíva TypeScript, nadmnožinu JavaScriptu, ktorá pridáva statické typovanie a ďalšie funkcie zlepšujúce vývojový proces. Medzi kľúčové vlastnosti Angularu patrí obojsmerná väzba dát, ktorá synchronizuje model a zobrazenie, a modulárny systém, ktorý umožňuje rozdelenie aplikácie na menšie, ľahko spravovateľné časti. Angular tiež obsahuje nástroje na správu formulárov, komunikáciu so serverom a smerovanie, čo umožňuje vytvárať komplexné aplikácie s minimálnym úsilím. [22]

### 2.4.4 RxJs

RxJs Reactive Extensions for JavaScript (RxJs) je knižnica určená na reaktívne programovanie, ktorá umožňuje efektívne spracovanie asynchrónnych a udalostne založených dátových tokov pomocou tzv. observables. Tieto observables predstavujú prúdy dát (napr.

kliknutia, HTTP odpovede), ktoré môžu byť pozorované a spracovávané rôznymi operátormi ako map, filter či reduce.

RxJS umožňuje prehľadnejšie a konzistentnejšie riadiť komplexné interakcie v aplikáciách, najmä tam, kde dochádza k častým zmenám stavu alebo udalostiam v čase. Využíva sa najmä v moderných frontend frameworkoch, ako je Angular, kde zjednodušuje prácu s dátami a udalosťami.[23]

### **2.4.5 Backendové frameworky**

Back-end framework je softvérový nástroj, ktorý uľahčuje vývoj serverovej časti webovej aplikácie. Poskytuje štruktúru, komponenty a funkcie potrebné na spracovanie požiadaviek, prácu s databázou, autentifikáciu a ďalšie serverové operácie. Cieľom back-end frameworku je zefektívniť vývoj, zabezpečiť konzistentnosť kódu a podporiť štandardizované riešenia. Rôzne frameworky (napr. Django, Spring, Laravel, Ruby on Rails, Express) sa líšia podľa náročnosti projektu, škálovateľnosti, dokumentácie a vhodnosti pre začiatočníkov či veľké podnikové aplikácie.[24]

Node.js tak využíva práve dobre definované vlastnosti JavaScriptu, ktoré umožňujú vytvárať vysoko výkonné platformy pre webové aplikácie.[25]

### **2.4.6 Express.js**

Express je JavaScriptový framework fungujúci ako ľahká nadstavba nad Node.js, ktorý zjednodušuje tvorbu webových aplikácií a API. Zatiaľ čo Node.js poskytuje základné prostredie pre spúšťanie JavaScriptu na serveri, Express pridáva štruktúru a pomáha vývojárom vyhnúť sa opakovanému a zdĺhavému písaniu nízkoúrovňového kódu. Express je teda nástroj, ktorý zjednodušuje a zrýchľuje vývoj tým, že poskytuje vyššiu úroveň abstrakcie oproti základným funkciám dostupným v čistom Node.js.

Jeho minimalistická architektúra ponúka veľkú mieru flexibility – vývojári nie sú nútení dodržiavať presne stanovenú štruktúru projektu, no zároveň majú k dispozícii nástroje na efektívne spracovanie HTTP požiadaviek, tvorbu routovacích pravidiel a využívanie middleware vrstiev. Express umožňuje jednoduché vytváranie RESTful API a dynamických webových aplikácií. Pre svoju jednoduchosť, rozsiahlu dokumentáciu a silnú komunitu patrí medzi najpopulárnejšie riešenia pre serverovú časť aplikácií postavených na JavaScripte.[26][24]



## 2.5 UI a UX

Používateľské rozhranie predstavuje vizuálnu časť digitálneho produktu, s ktorou používateľ priamo interaguje. Zahŕňa prvky ako tlačidlá, ikony, typografiu a farebné schémy. Cieľom UI dizajnu je vytvoriť esteticky príjemné a funkčné prostredie, ktoré uľahčuje používateľovi navigáciu a interakciu s produktom.

Používateľský zážitok sa zameriava na celkovú skúsenosť používateľa pri interakcii s produktom alebo službou. Ide o to, ako intuitívne a efektívne dokáže používateľ dosiahnuť svoje ciele. UX dizajn zahŕňa analýzu potrieb používateľov, návrh informačnej architektúry, prototypovanie a testovanie, s cieľom zabezpečiť, aby bol produkt nielen funkčný, ale aj príjemný na používanie.

Hoci sú UI a UX odlišné disciplíny, úzko spolupracujú s cieľom vytvoriť digitálne produkty, ktoré sú nielen vizuálne atraktívne, ale aj používateľsky prívetivé a efektívne.[27]

### 2.5.1 Figma

Figma je cloudová platforma na návrh a prototypovanie používateľských rozhraní pre webové a mobilné aplikácie. Umožňuje tímovú spoluprácu v reálnom čase prostredníctvom webového prehliadača alebo desktopovej aplikácie. Podporuje vektorové ilustrácie, interaktívne prototypy a dizajnové systémy. Všetky projekty sú uložené v cloude, čo zjednodušuje prístup a zdieľanie. [28]

### 2.5.2 Material UI

Angular Material UI je oficiálna knižnica komponentov pre Angular, ktorá implementuje dizajnové princípy Material Design od spoločnosti Google. Poskytuje predpripravené a prispôsobiteľné komponenty, ako sú tlačidlá, formuláre, tabuľky či navigačné panely, ktoré umožňujú vývojárom rýchlo vytvárať moderné, responzívne a esteticky príjemné používateľské rozhrania. Angular Material UI podporuje integráciu s Angular frameworkom a zabezpečuje konzistentný dizajn a vysokú úroveň použiteľnosti v rámci aplikácií. [29]

## 2.6 Gamifikácia

Gamifikácia je aplikácia herných prvkov a princípov v neherných kontextoch s cieľom zvýšiť angažovanosť a motiváciu jednotlivcov pri vykonávaní určitých aktivít. Tento prístup sa využíva v rôznych oblastiach, ako sú vzdelávanie, marketing, podnikanie a osobný rozvoj. Vo vzdelávacom prostredí to znamená implementáciu mechanizmov, ako sú zbieranie bodov, získavanie odmien, porovnávanie sa s ostatnými či postupovanie na vyššie úrovne, aby sa

proces učenia stal interaktívnejším a pútavejším.

### **2.6.1 Gamifikácia v e-learningu**

Gamifikácia v e-learningu zahŕňa integráciu herných prvkov, ako sú body, odznaky, rebríčky a úrovne, do vzdelávacích online prostredí s cieľom zvýšiť motiváciu a angažovanosť študentov. Podľa príspevku v zborníku z medzinárodnej vedeckej konferencie "Vzdělávání dospělých 2021"[30] je gamifikácia efektívnym nástrojom na podporu aktívneho učenia, pretože využíva prirodzenú ľudskú tendenciu k hre a súťaživosti. Implementácia herných mechanizmov v e-learningových kurzoch môže viesť k zlepšeniu zapojenia študentov, zvýšeniu ich motivácie a následne k lepším vzdelávacím výsledkom.

## **2.7 Server**

Server je centrálny počítač v sieti, ktorý poskytuje služby a zdroje iným zariadeniam – tzv. klientom. Umožňuje uchovávať a sprístupňovať dáta, spúšťať aplikácie, spracovávať požiadavky používateľov alebo poskytovať webové stránky či e-mailové služby. Môže ísť o fyzický počítač alebo virtuálny stroj a jeho konkrétna funkcia závisí od typu – napríklad súborový, databázový, aplikačný alebo webový server. Správne nastavený server je základom spoľahlivej a bezpečnej IT infraštruktúry.[31]

### **2.7.1 NGINX**

Nginx[32] je výkonný a spoľahlivý webový a reverzný proxy server, ktorý je navrhnutý na efektívne spracovanie veľkého množstva súčasných pripojení pri minimálnom využití systémových zdrojov. Najčastejšie sa používa na poskytovanie statického obsahu, ako sú HTML, CSS, JavaScript alebo obrázky, a zároveň ako sprostredkovateľ medzi klientom a backend aplikáciou, čím preposiela požiadavky na servery bežiacie napríklad v Node.js alebo PHP. Nginx dokáže tiež rozdeľovať záťaž medzi viaceré servery, čím zabezpečuje vyššiu dostupnosť a škálovateľnosť aplikácií, a môže slúžiť ako cache server pre rýchlejšie doručovanie často používaného obsahu. Vďaka svojej asynchrónnej architektúre a podpore moderných protokolov, ako sú HTTP/3 a TLS, je ideálnym riešením pre hosting moderných webových aplikácií s vysokými nárokmi na výkon.

### **2.7.2 Docker**

Docker je open-source platforma, ktorá umožňuje vývoj, distribúciu a správu aplikácií pomocou tzv. kontajnerov. Kontajner je ľahké a izolované prostredie, ktoré obsahuje všetko potrebné na beh aplikácie – vrátane kódu, knižníc, runtime prostredia a nastavení.

Vďaka tomu môže aplikácia bežať rovnakým spôsobom na rôznych miestach, napríklad na vývojárskom počítači, v testovacom prostredí aj na produkčnom serveri. Docker využíva virtualizáciu na úrovni operačného systému, čo mu umožňuje spúšťať viacero kontajnerov efektívne a s nízkymi nárokmi na systémové prostriedky.[33]

## 2.8 GIT

Git je distribuovaný systém na správu verzií, ktorý umožňuje sledovať a zaznamenávať zmeny v súboroch počas vývoja softvéru. Umožňuje viacerým vývojárom pracovať súčasne na tom istom projekte, pričom každá kópia projektu obsahuje úplnú históriu zmien. Git je známy svojou rýchlosťou, flexibilitou a podporou nelineárnych pracovných tokov. Je open-source, bezplatný a vďaka svojim vlastnostiam sa stal štandardom v oblasti verzionovania kódu.[34]

### 3 Návrh webovej aplikácie

Naša webová aplikácia je koncipovaná ako dvojvrstvový systém, pozostávajúci z frontendovej časti, implementovanej pomocou frameworku Angular, a backendovej časti, vytvorenej pomocou platformy Express. Obe časti sú navrhnuté ako samostatné a na sebe nezávislé aplikácie, čo zvyšuje flexibilitu a umožňuje ich samostatný vývoj, testovanie a nasadzovanie.

Backendová časť je zodpovedná za spracovanie údajov, manipuláciu s databázou, LDAP autentifikáciu používateľov a spracovanie požiadaviek zo strany klienta. Slúži ako integračná vrstva medzi používateľským rozhraním a databázou, pričom zabezpečuje bezpečný a efektívny prenos údajov. Backend sme implementovali pomocou frameworku Express.js, ktorý je postavený na Node.js a umožňuje jednoducho vytvárať REST API rozhrania. Medzi jeho hlavné výhody patrí jednoduchosť, flexibilita, rozsiahla komunita a množstvo dostupných middleware modulov, čo výrazne urýchľuje vývoj serverovej logiky.

Express.js je zároveň ideálne riešenie pre aplikácie typu SPA, akú predstavuje aj náš frontend vytvorený v Angulari. V tomto architektonickom modeli Express plní úlohu backendovej vrstvy, ktorá poskytuje API endpointy, prostredníctvom ktorých frontend získava dynamicky obsah a dáta. Táto kombinácia umožňuje rýchlu a plynulú interakciu používateľa s aplikáciou bez potreby neustáleho obnovovania stránky, čím sa zvyšuje používateľský komfort a celkový výkon systému.

Pomocou LDAP autentifikácie dokážeme navyše overovať identitu používateľov a priradovať im prístupové práva na základe ich roly – napríklad ako študent alebo administrátor (učiteľ). Tento prístup umožňuje efektívne riadenie oprávnení a zabezpečenie rôznych úrovní prístupu k funkcionalitám aplikácie.

Frontendová časť aplikácie je navrhnutá ako Single Page Application (SPA). Použitie Angular frameworku bolo motivované jeho výbornou škálovateľnosťou a komponentovým prístupom, ktorý umožňuje vytvárať zapuzdrené (encapsulované) komponenty. Tento prístup vedie k lepšej organizácii a čitateľnosti kódu, a zároveň uľahčuje jeho údržbu a opätovné použitie.

Komunikácia medzi frontendom a backendom je sprostredkovaná prostredníctvom tzv. servisných tried (Services), ktoré Angular poskytuje ako jednu zo svojich kľúčových funkcionalít. Tieto služby slúžia na získavanie údajov zo servera, validáciu používateľských vstupov, uchovávanie stavu aplikácie, ako aj na centrálnu spracovanie logiky a dát. Každá Angular služba je singleton<sup>5</sup> objekt, čo umožňuje zdieľanie dát a funkcionality

---

<sup>5</sup>Návrhový vzor, ktorý zabezpečuje, že z určitej triedy existuje v aplikácii práve jedna inštancia, ktorá je globálne dostupná.

naprieč celou aplikáciou. Tento prístup významne zvyšuje modularitu systému a podporuje znovupoužiteľnosť kódu.

Pri asynchrónnej komunikácii a práci s údajmi využívame v Angulari mechanizmus Observables, ktorý je súčasťou knižnice RxJs. Observables umožňujú efektívne pracovať s dátovými tokmi, ako sú napríklad HTTP požiadavky alebo používateľské vstupy, pričom poskytujú pokročilé možnosti ako je reaktívne spracovanie dát, zrušenie požiadaviek, spájanie viacerých streamov alebo transformácia údajov. Vďaka tejto reaktívnej paradigme dokáže aplikácia flexibilne reagovať na zmeny v dátach a zvyšuje sa jej interaktivita, výkon a stabilita.

### 3.1 Špecifikácia

Primárnym cieľom našej webovej aplikácie je poskytnúť efektívnu a modernú platformu pre študentov, ktorí si chcú prehĺbiť svoje znalosti z matematiky, špeciálne z oblasti teórie pravdepodobnosti. V reakcii na obmedzenia a nedostatky súčasných vzdelávacích systémov a aplikácií je naša platforma navrhnutá tak, aby nebola len nástrojom na tvorbu a vyhodnocovanie testov, ale zároveň poskytovala interaktívne študijné materiály a vizualizácie, ktoré zlepšujú pochopenie teoretických konceptov.

Aplikácia kladie dôraz na rozdelenia pravdepodobnosti, pričom podporuje diskrétna rozdelenia ako rovnomerné, binomické, hypergeometrické, poissonovo a geometrické rozdelenie. Každé z rozdelení je v aplikácii reprezentované interaktívnymi vizualizáciami a príkladmi, čo študentom umožňuje lepšie porozumieť danej problematike.

Systém zároveň rozlišuje dve hlavné role používateľov – rolu *študenta* a rolu *administrátora (učiteľa)*. Každá z týchto rolí disponuje špecifickými funkcionalitami, ktoré sú prístupné iba na základe autentifikácie používateľa. Študentom aplikácia umožňuje prístup k študijným materiálom, generovanie testov a sledovanie svojho pokroku. Administrátori majú k dispozícii rozšírené možnosti správy obsahu, vrátane tvorby testov, ako aj monitorovania aktivít a pokroku jednotlivých študentov.

Pre systematické znázornenie interakcií používateľov so systémom, sú vytvorené diagramy prípadov použitia zvlášť pre rolu študenta (obrázok č. 1) a administrátora (obrázok č. 2), ktoré sú súčasťou ďalších kapitol tejto práce.

### 3.2 Funkcionálne požiadavky

Funkcionálne požiadavky sú špecifikácie, ktoré definujú, čo by systém mal robiť a aké funkcie by mal poskytovať používateľom. Tieto požiadavky sa zameriavajú na konkrétne

správanie systému, interakcie s používateľmi a spracovanie údajov. V našej aplikácii sme identifikovali nasledujúce funkcionálne požiadavky pre študentov a administrátorov:

### **3.2.1 Pre študentov**

- Prihlásenie do systému pomocou LDAP autentifikácie.
- Prehliadanie študijných materiálov z oblasti pravdepodobnosti a štatistiky.
- Generovanie testov podľa vybraných tém.
- Riešenie úloh v rámci testovej sady s možnosťou využitia nápoved.
- Zobrazenie výsledkov testov vrátane získaného počtu bodov a použitých nápoved.
- Sledovanie vlastného pokroku a úspešnosti pri riešení úloh.
- Export testových úloh a výsledkov vo formáte vhodnom na uloženie alebo tlač.

### **3.2.2 Pre administrátorov (učiteľov)**

- Prihlásenie do systému pomocou LDAP autentifikácie ako učiteľ.
- Zobrazovanie výsledkov všetkých študentov.
- Prístup k všeobecným štatistikám a analýzam výkonnosti.
- Pridávanie nových testových úloh.
- Správa existujúcich testových úloh vrátane ich úpravy a mazania.
- Zobrazovanie histórie testov a pokroku jednotlivých študentov.

## **3.3 Nefunkcionálne požiadavky**

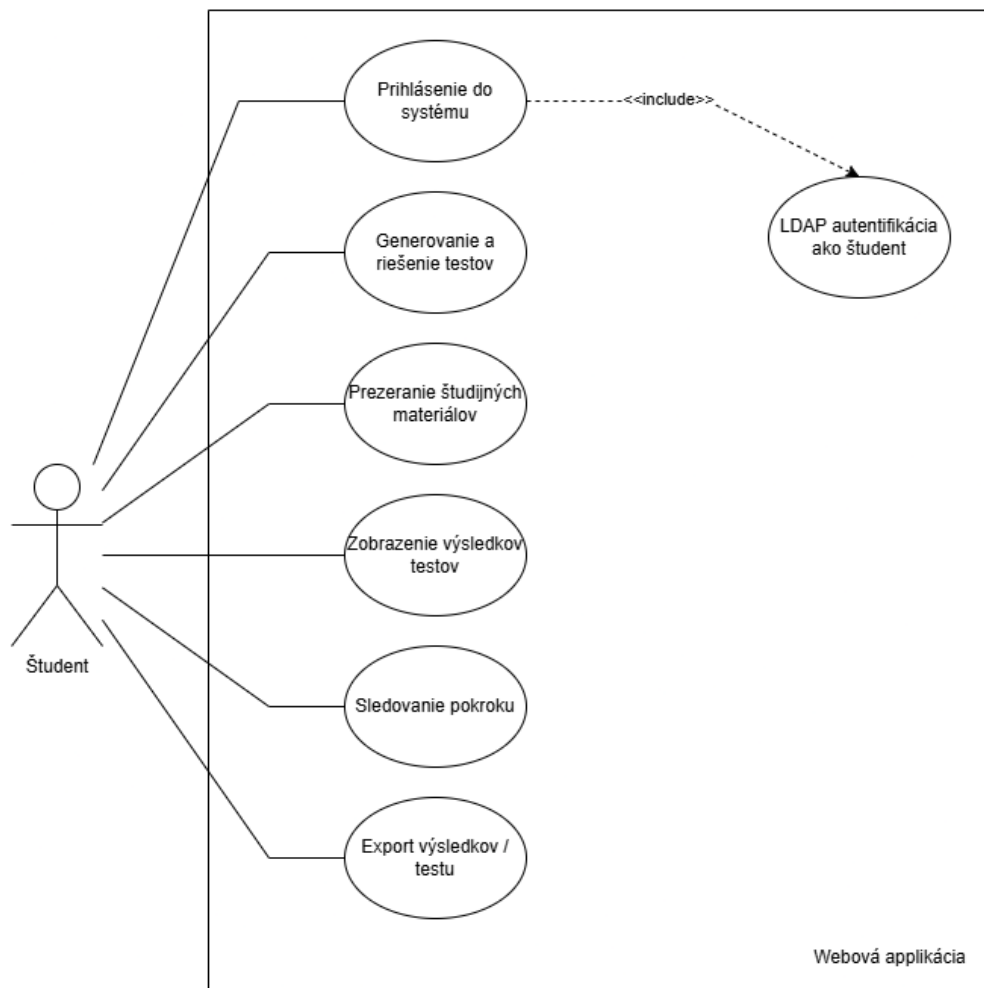
Nefunkcionálne požiadavky sú špecifikácie, ktoré sa zameriavajú na vlastnosti a kvalitu systému, ako sú výkon, bezpečnosť, použiteľnosť a údržba. Tieto požiadavky sa netýkajú konkrétnych funkcií, ale skôr toho, ako by mal systém fungovať a aké vlastnosti by mal mať. V našej aplikácii sme identifikovali nasledujúce nefunkcionálne požiadavky:

- Jednoduchosť a prehľadnosť používateľského rozhrania – rozhranie musí byť intuitívne, prispôsobené aj menej technicky zdatným používateľom.
- Spoľahlivé a overené študijné materiály – obsah aplikácie je čerpaný z dôveryhodných odborných zdrojov.

- Rýchla odozva pri generovaní testov a spracovaní výsledkov – činnosti musia byť vykonané v čo najkratšom čase bez zbytočného čakania.
- Podpora rôznych zariadení – aplikácia musí byť plne funkčná na počítačoch, tabletoch aj mobilných zariadeniach.
- Modulárnosť a rozširiteľnosť systému – systém je navrhnutý tak, aby bolo možné v budúcnosti pridávať nové moduly, úlohy či témy bez nutnosti zásahu do existujúceho jadra aplikácie.

### 3.4 Diagramy prípadov použitia

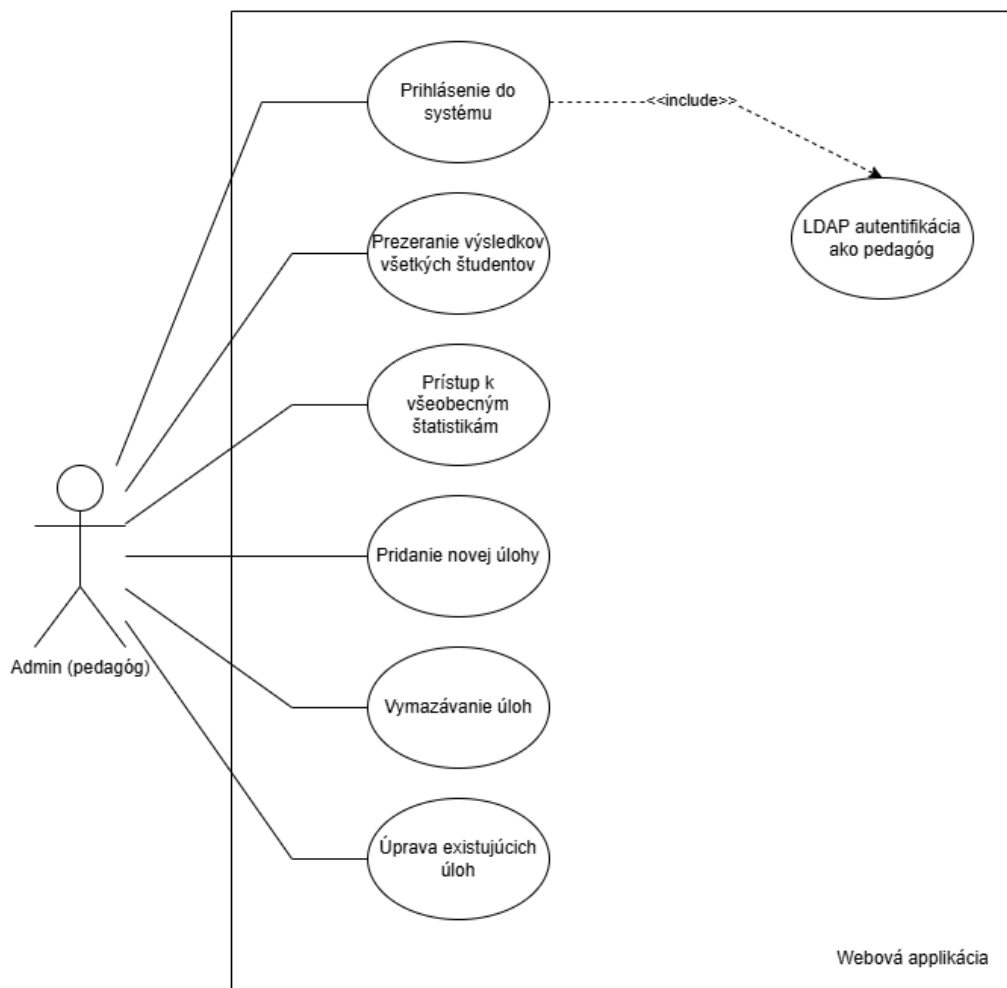
#### 3.4.1 Diagramy prípadov použitia pre študenta



Obr. 1: Diagram prípadov použitia pre študenta.



### 3.4.2 Diagramy prípadov použitia pre administrátora



Obr. 2: Diagram prípadov použitia pre administrátora.

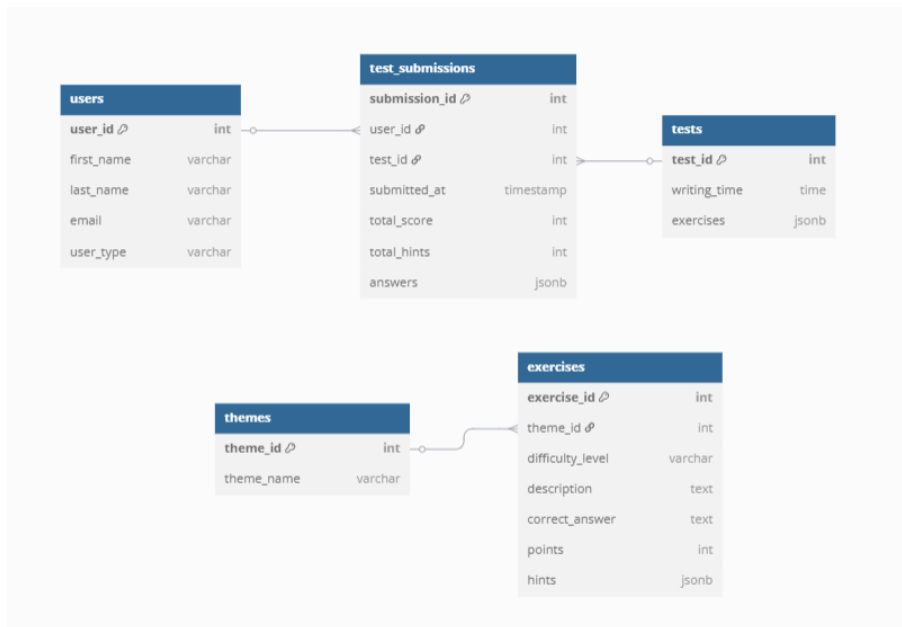
## 3.5 Databáza

Databázovým systémom našej webovej aplikácie je PostgreSQL, ktorý sme si zvolili pre jeho flexibilnú prácu s dátami. Využívame najmä podporu pre JSONB, ktorá nám umožňuje efektívne ukladať a vyhľadávať pološtruktúrované dáta, ako sú odpovede či nápovedy.

Ďalšou výhodou je podpora polových typov (ARRAY), ktoré zjednodušujú štruktúru databázy tým, že umožňujú uchovávať viacero hodnôt v jednom stĺpci. PostgreSQL nám

tak poskytuje výkonné a praktické riešenie pre prácu s komplexnými údajmi.

Na obrázku č 3. je zobrazený logický ER diagram (Entity-Relationship Diagram) databázového modelu našej aplikácie. Tento diagram zobrazuje vzťahy medzi jednotlivými entitami a znázorňuje, ako sú údaje v databáze navzájom prepojené.



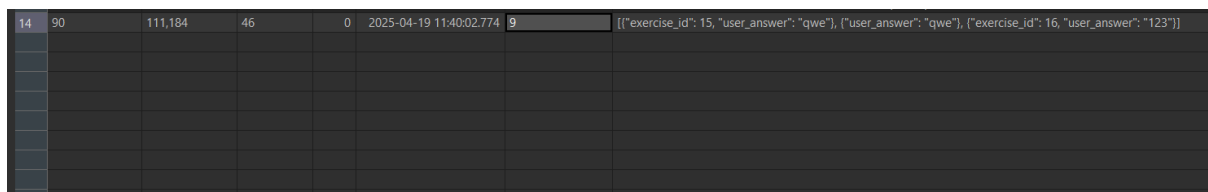
Obr. 3: Logický ER diagram (Entity-Relationship Diagram) databázového modelu.

Databáza sa skladá z nasledovných tabuliek:

- *users* – uchováva informácie o prihlásených používateľoch. Keďže autentifikáciu zabezpečujeme prostredníctvom akademického LDAP systému, neukladáme žiadne citlivé dáta, ktoré by neboli dostupné priamo z LDAP. Okrem základných údajov rozlišujeme aj typ používateľa (študent alebo administrátor/učiteľ), ktorý taktiež získavame z LDAP a ktorý ovplyvňuje rozsah oprávnení v aplikácii.
- *themes* - obsahuje zoznam matematických tém, primárne z oblasti matematickej štatistiky, ktoré slúžia na kategorizáciu úloh.
- *exercises* - uchováva údaje o jednotlivých úlohách, ako sú zadanie, úroveň obtiažnosti, počet bodov, správna odpoveď a dostupné nápovedy. Každá úloha je priradená k jednej tematickej oblasti (theme\_id).
- *tests* - reprezentuje sadu úloh, ktorá sa skladá z kombinácie úloh z databázy a automaticky generovaných úloh. Zároveň obsahuje aj časový limit na vypracovanie testu.

- *test\_submissions* - zaznamenáva konkrétne podania testov jednotlivými používateľmi. Obsahuje informácie o tom, kto test riešil, akú testovú sadu dostal, kedy test odovzdal, aké odpovede zadal, koľko bodov získal a koľko nápoved použil.

Polia *answers* a *hints* sú uložené vo formáte JSONB, čo umožňuje flexibilne uchovávať komplexné štruktúry dát, ako napríklad zoznam odpovedí alebo počty použitých nápoved ku konkrétnym úlohám. Príklad zápisu JSON dát v tomto stĺpci je znázornený na obrázku č. 4.

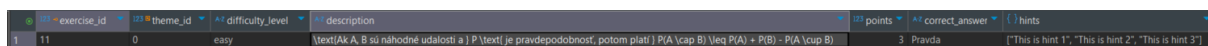


14	90	111,184	46	0	2025-04-19 11:40:02.774	9	[{"exercise_id": 15, "user_answer": "qwe"}, {"user_answer": "qwe"}, {"exercise_id": 16, "user_answer": "123"}]

Obr. 4: Príklad zápisu JSON dát v stĺpci *answers*.

Každé zadanie úlohy (description) je uložené ako reťazec vo formáte LaTeX, čo umožňuje pohodlný zápis matematických vzorcov a výrazov. Tento spôsob zápisu je výhodný najmä pre potreby zobrazovania úloh vo webovom rozhraní pomocou renderovacích nástrojov ako je napríklad MathJax, ktoré dokážu LaTeX premeniť na kvalitne vykreslené rovnice.

Príklad zápisu jednej úlohy s použitím LaTeX zápisu je znázornený na obrázku č. 5.



exercise_id	theme_id	difficulty_level	description	points	correct_answer	hints
11	0	easy	Text(Ak A, B sú náhodné udalosti a $P$ text) je pravdepodobnosť, potom platí $P(A \cup B) \leq P(A) + P(B) - P(A \cap B)$	3	Pravda	[ "This is hint 1", "This is hint 2", "This is hint 3" ]

Obr. 5: Ukážka LaTeX zápisu zadania úlohy v databáze

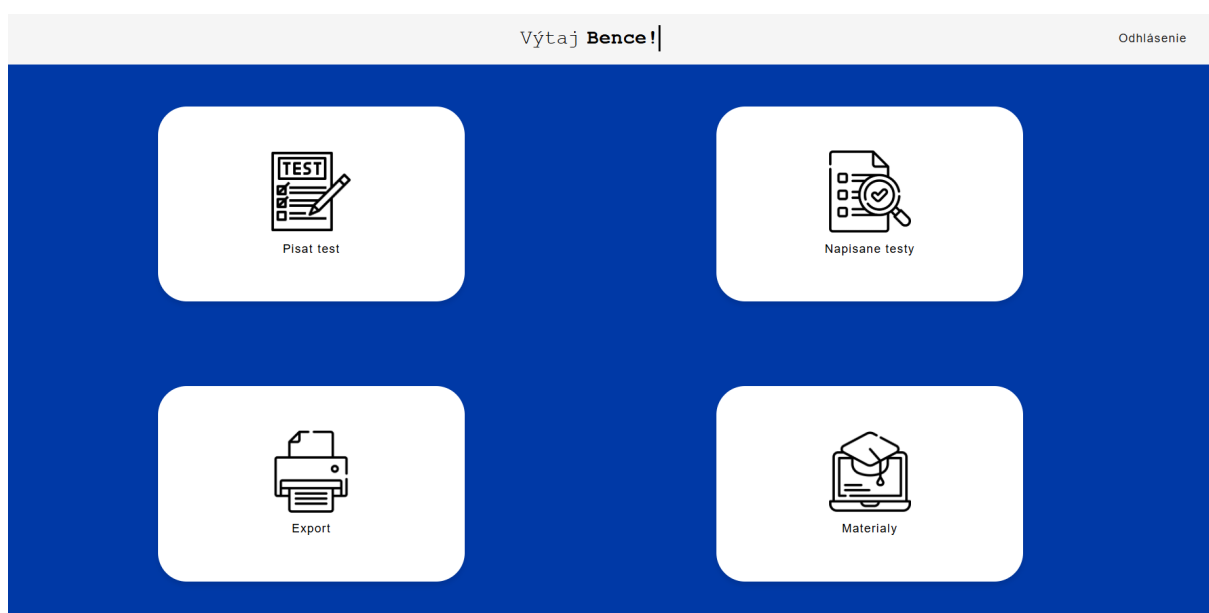
### 3.6 Používateľské rozhranie

Používateľské rozhranie aplikácie bolo navrhnuté s dôrazom na prehľadnosť, jednoduchosť a intuitívnu navigáciu, čím sa znižuje kognitívna záťaž používateľa a zvyšuje sa celková používateľská skúsenosť (UX). Cieľom návrhu bolo zabezpečiť, aby sa aj používateľ s minimálnymi digitálnymi zručnosťami dokázal v prostredí jednoducho orientovať a efektívne pracovať so všetkými funkciami aplikácie.

Základná štruktúra dizajnu bola vytvorená v nástroji Figma, ktorý umožnil vizuálny návrh používateľského rozhrania ešte pred samotnou implementáciou. Rozhranie je koncipované ako hlavné menu s veľkými, vizuálne výraznými blokmi, ktoré reprezentujú jednotlivé kľúčové funkcie aplikácie:

- Písanie testu
- Zobrazenie napísaných testov
- Export testov a výsledkov
- Prístup k študijným materiálom

Ako môžeme vidieť na obrázku č. 6, ide o hlavnú obrazovku aplikácie, ktorá slúži ako východiskový bod pre všetky hlavné činnosti používateľa.



Obr. 6: Hlavná obrazovka aplikácie.

Navigácia v aplikácii je zabezpečená pomocou horného navigačného panela, ktorý obsahuje možnosť návratu na hlavnú obrazovku a tlačidlo pre odhlásenie zo systému. Tento prvok je dostupný na všetkých podstránkach aplikácie, čím sa zabezpečuje jednotný a konzistentný spôsob navigácie naprieč celým systémom.

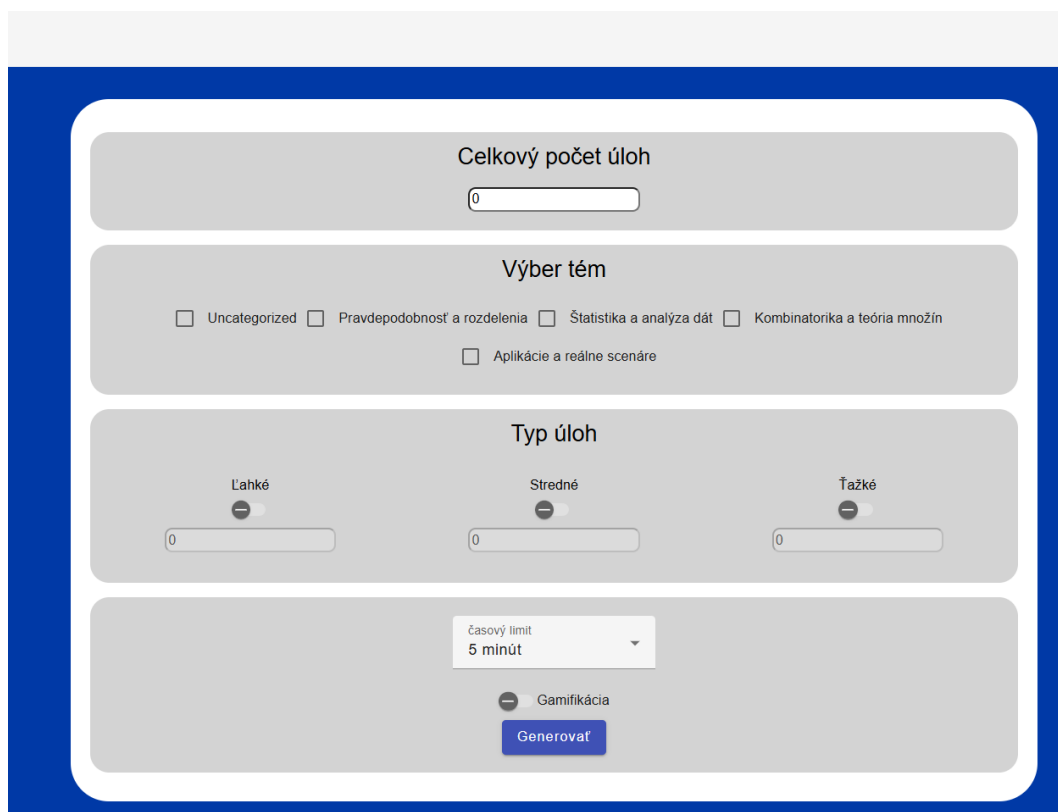
Na obrázku č. 7 je zobrazený výrez z konceptuálneho návrhu podstránky *Písať test*, ktorý bol súčasťou úvodného vizuálneho návrhu. V porovnaní s finálnou implementáciou prešli niektoré prvky úpravami zameranými na lepšiu prehľadnosť, responzivnosť a používateľskú použiteľnosť.



Obr. 7: Výrez z konceptuálneho návrhu podstránky Písať test.

### 3.7 Preskúšanie sa

Súčasťou navrhovanej webovej aplikácie je stránka umožňujúca používateľom vytvárať vlastné testovacie sady úloh na precvičenie a preverenie vedomostí z oblasti pravdepodobnosti a štatistiky. Používateľské rozhranie tejto stránky (obrázok 8) bolo navrhnuté tak, aby bolo intuitívne a aby zabezpečovalo príjemnú používateľskú skúsenosť (UX).



Obr. 8: Používateľské rozhranie stránky pre vytvorenie testovacej sady.

Na tejto stránke má používateľ možnosť presne definovať parametre generovaného testu, ako napríklad:

- **Výber tém** – používateľ si môže označiť matematické témy, z ktorých chce byť testovaný. Ide najmä o témy ako pravdepodobnosť a rozdelenia, štatistika a analýza dát, kombinatorika a teória množín, a tiež aplikácie a reálne scenáre.
- **Typ a obtiažnosť úloh** – umožňuje nastaviť počet ľahkých, stredne ťažkých a ťažkých úloh.
- **Časový limit** – určuje maximálny čas na vypracovanie celého testu.

- **Gamifikácia** – voliteľná funkcionálna, ktorej účelom je zvýšiť motiváciu používateľov prostredníctvom herných prvkov, bližšie vysvetlená v nasledujúcej podkapitole.

Po úspešnom vygenerovaní testovej sady je používateľ automaticky presmerovaný na stránku samotného preskúšania (obrázok ??). Pri návrhu tejto stránky sme kládli dôraz na vizuálnu spätnú väzbu pre používateľa – už zodpovedané úlohy sú jasne vizuálne označené, čím sa zlepšuje prehľadnosť testu. Súčasťou rozhrania je aj stručný popis používania aplikácie, ktorý sa zobrazí kliknutím na príslušnú ikonu s informáciami.

Po uplynutí časového limitu sa používateľovi automaticky zobrazí modálne okno, ktoré informuje o ukončení testu a odoslaní odpovedí na spracovanie a vyhodnotenie.

Na správne a čitateľné vykresľovanie matematických výrazov a rovníc bola použitá JavaScriptová knižnica **MathJax**. Tá zabezpečuje bezproblémové zobrazenie matematických symbolov a vzorcov priamo vo webovom prehliadači používateľa.

### 3.7.1 Gamifikácia

Podsekcia bude venovaná detailnému opisu gamifikačných prvkov použitých v aplikácii, ich cieľom a spôsobom implementácie. (Obsah bude dopísaný v nasledujúcich krokoch práce.)

### 3.7.2 Generovanie úloh

Táto podsekcia sa venuje technickému procesu generovania úloh pre testy na základe používateľských preferencií, výberu tém a obtiažností. (Obsah bude dopísaný v nasledujúcich krokoch práce.)

## 3.8 Materialy

### 3.8.1 Interaktívne grafy

## 3.9 Vytváranie úloh (Admin)

## 3.10 Vyhodnocovanie úloh

## 4 Implementacia webovej aplikacie

V tejto kapitole sa podrobne venujeme implementačnej časti webovej aplikácie, pričom rozdeľujeme jednotlivé aspekty systému na backend, frontend a databázovú vrstvu. Cieľom tejto časti je ukázať, ako boli teoretické návrhy z predchádzajúcej kapitoly pretavené do konkrétnej technickej realizácie.

V jednotlivých sekciách prezentujeme použité technológie a knižnice, ako aj konkrétne časti zdrojového kódu ilustrujúce dôležité funkcionality, ako napríklad generovanie úloh, vizualizácie dát pomocou grafov alebo spracovanie používateľského vstupu pri pridávaní nových otázok do databázy.

### 4.1 Vývojové prostredie a inštalácia závislostí

Počas vývoja webovej aplikácie sme využívali vývojové prostredie **Visual Studio Code**[35], ktoré poskytuje množstvo rozšírení vhodných pre vývoj v JavaScripte a TypeScripte, ako aj nástroje na efektívnu správu projektovej štruktúry, ladenie a kontrolu syntaxe.

Backendová časť aplikácie bola postavená na platforme **Node.js**, ktorá umožňuje spúšťanie JavaScriptového kódu na strane servera. Na správu knižníc a závislostí sme používali Node Package Manager (Správca balíkov pre Node.js) (NPM), ktorý je súčasťou štandardnej inštalácie Node.js. Pomocou neho boli do projektu inštalované všetky potrebné knižnice, ako napríklad `express`, `MathQuill`, `body-parser`, `cors` a ďalšie.

Závislosti boli definované v súbore `package.json`, ktorý slúži zároveň ako dokumentačný a konfiguračný súbor pre projekt. Samotná inštalácia všetkých závislostí sa vykoná príkazom:

```
npm install
```

Týmto príkazom sa automaticky stiahnu všetky knižnice uvedené v súbore `package.json` a pripraví sa na použitie v projekte.

### 4.2 Frontend – Angular

Frontendová časť aplikácie bola implementovaná pomocou frameworku **Angular**, ktorý je postavený na TypeScripte a poskytuje robustnú architektúru pre budovanie komplexných single-page aplikácií (SPA). Angular ponúka množstvo výhod ako je komponentovo orientovaný vývoj, dvojcestná väzba dát, modulárnosť a rozsiahla komunita s kvalitnou dokumentáciou.



Na to, aby sme mohli Angular používať, je potrebné mať nainštalovaný **Node.js** a s ním aj správcu balíkov NPM, ktoré sme spomínali v predchádzajúcej sekcii. Po ich nainštalovaní je možné Angular CLI nainštalovať globálne<sup>6</sup> pomocou príkazu:

```
npm install -g @angular/cli
```

Správnou inštaláciu Angularu môžeme overiť pomocou príkazu:

```
ng version
```

Výstup z tohto príkazu zobrazí verziu Angular CLI, ako aj verzie jednotlivých komponentov Angularu. Na obrázku 9 je znázornený výstup z tohto príkazu v termináli<sup>7</sup>.

The image shows a terminal window with a dark background. At the top, the Angular logo is displayed in a stylized, outlined font. Below the logo, the following information is printed: 'Angular CLI: 18.2.12', 'Node: 20.17.0', 'Package Manager: npm 10.9.0', and 'OS: win32 x64'. Then, the Angular version '18.2.13' is shown, followed by a list of installed packages: 'animations', 'common', 'compiler', 'compiler-cli', 'core', 'forms', 'platform-browser', 'platform-browser-dynamic', and 'router'. Finally, a table lists the versions of various packages. The table has two columns: 'Package' and 'Version'. The packages listed are '@angular-devkit/architect' (0.1802.12), '@angular-devkit/build-angular' (18.2.12), '@angular-devkit/core' (18.2.12), '@angular-devkit/schematics' (18.2.12), '@angular/cdk' (18.2.14), '@angular/cli' (18.2.12), '@angular/material' (18.2.14), '@schematics/angular' (18.2.12), 'rxjs' (7.8.1), 'typescript' (5.5.4), and 'zone.js' (0.14.10).

Package	Version
@angular-devkit/architect	0.1802.12
@angular-devkit/build-angular	18.2.12
@angular-devkit/core	18.2.12
@angular-devkit/schematics	18.2.12
@angular/cdk	18.2.14
@angular/cli	18.2.12
@angular/material	18.2.14
@schematics/angular	18.2.12
rxjs	7.8.1
typescript	5.5.4
zone.js	0.14.10

Obr. 9: Výstup príkazu `ng version` v termináli.

Nový Angular projekt môžeme vytvoriť pomocou príkazu:

<sup>6</sup>Prepínač `-g` znamená globálnu inštaláciu, čím získame prístup ku príkazu `ng` v ktoromkoľvek priečinku systému.

<sup>7</sup>Terminál vo Visual Studio Code predstavuje integrované rozhranie pre prácu s príkazovým riadkom priamo v editore.

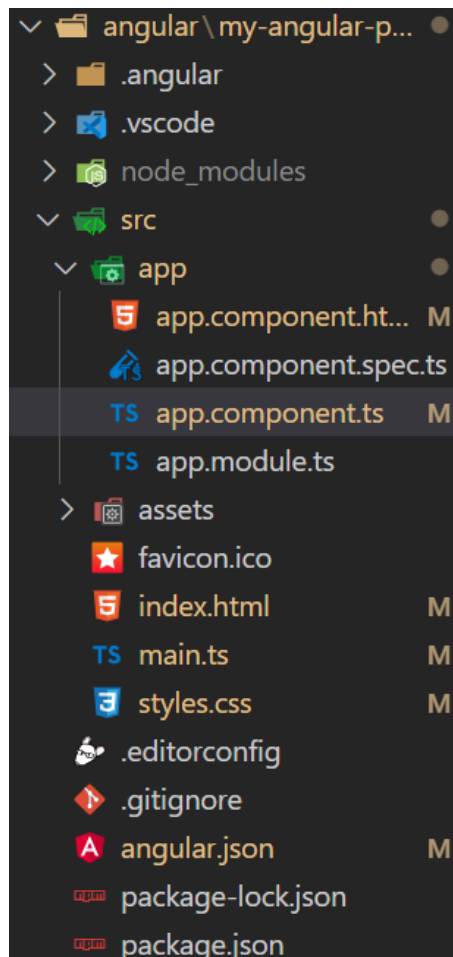
```
ng new angular-app
```

Tento príkaz vygeneruje základnú adresárovú štruktúru aplikácie, vrátane konfiguračných súborov, ako je napríklad `angular.json`, `package.json`, a priečinkov `src/`, `app/` a ďalších. Po úspešnom vytvorení projektu môžeme aplikáciu spustiť príkazom:

```
ng serve
```

Týmto príkazom sa automaticky nainštalujú všetky závislosti zo súboru `package-lock.json` a vytvorí sa priečinok `node_modules`, ktorý obsahuje všetky knižnice potrebné na beh aplikácie. Aplikácia je následne dostupná na adrese `http://localhost:4200`.

Na obrázku 10 je znázornená základná adresárová štruktúra Angular projektu po vytvorení.



Obr. 10: Základná štruktúra Angular projektu.

### 4.2.1 Graficka cast tejto PICOVINY

RxJS node-sass kvoli renderovaniu apex charts - lazy loading math jax chart.js mathquill  
jQuery pdfmake Bootstrap bootstrap animation freedsound.org

## 4.3 Backend

Node Nginx

CURL veci

**Inštalácia a použitie Express frameworku** Inštalácia Expressu je jednoduchá a vykonáva sa pomocou balíčkovacieho nástroja NPM príkazom:

```
npm install express
```

Po úspešnej inštalácii je možné vytvoriť základný server a definovať vlastné API routy. Najjednoduchšia implementácia servera môže vyzeráť nasledovne:

```
const express = require('express');  
const app = express();  
const port = 3000;  
  
app.listen(port, () => { console.log(Server beží na porte ${port}); });
```

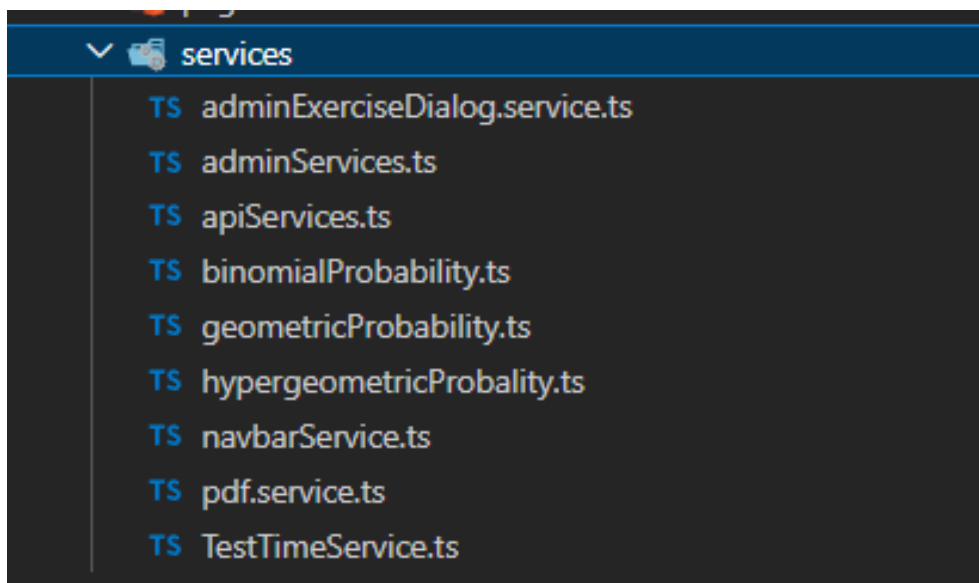
Výpis 1: Ukážka základnej konfigurácie Express.js servera

Tento server môžeme spustiť cez terminál pomocou príkazu:

```
node server.js
```

kde `server.js` je názov súboru obsahujúceho uvedený kód. Ak je server úspešne spustený, v konzole sa vypíše správa „Server beží na porte 3000“. V projekte je logika rozdelená do tzv. **servisných tried** (services), ktoré abstrahujú jednotlivé domény funkcionality – napríklad práca s testami, používateľmi alebo štatistikami. Tieto triedy slúžia ako rozhranie medzi routou a databázovou logikou. Na obrázku 11 je znázornený výrez zo štruktúry backendu so zameraním na jednotlivé služby.

Tento kód vytvorí základný server, ktorý je pripravený na ďalšie definovanie rout a pridávanie funkcionality. V projekte je logika rozdelená do tzv. **servisných tried** (services), ktoré abstrahujú jednotlivé domény funkcionality. Tieto triedy slúžia ako rozhranie medzi routou a databázovou logikou. Na obrázku 11 je znázornený výrez zo štruktúry backendu so zameraním na jednotlivé služby.



Obr. 11: Ukážka štruktúry backendových služieb v aplikácii.

Na to, aby frontend mohol komunikovať s backendom, je potrebné povoliť prístup medzi rôznymi doménami (napr. frontend beží na `http://localhost:4200` a backend na `http://localhost:3000`). Na tento účel sme použili knižnicu **CORS** (*Cross-Origin Resource Sharing*), ktorá bola nakonfigurovaná nasledovne:

```
const cors = require('cors');
app.use(cors({
  origin: 'http://localhost:4200',
  credentials: true
}));
```

Výpis 2: Konfigurácia CORS pre komunikáciu medzi FE a BE

Bez tejto konfigurácie by prehliadač blokoval požiadavky medzi rôznymi portmi alebo doménami, čo by znemožnilo komunikáciu medzi frontendom a backendom.

Na správne spracovanie dát typu `application/json`, ktoré odosiela Angular prostredníctvom POST alebo PUT požiadaviek, sme použili middleware **body-parser**. Tento middleware zabezpečí, že JSON objekt v tele požiadavky bude automaticky spracovaný a dostupný cez `req.body`. Je to obzvlášť dôležité pri práci s databázou PostgreSQL, kde používame dátový typ JSONB.

```
const bodyParser = require('body-parser');
app.use(bodyParser.json());
```

Výpis 3: Použitie body-parser pre spracovanie JSON požiadaviek

Týmto nastavením zabezpečíme, že každý JSON payload zo strany klienta bude backendom správne spracovaný a následne uložený do databázy vo formáte JSONB.

### 4.3.1 Autentifikácia

Prihlasovanie používateľov v aplikácii je riešené pomocou **LDAP autentifikácie**, pričom aplikácia komunikuje s akademickým LDAP serverom `ldap.stuba.sk`. Na overenie prihlasovacích údajov využívame knižnicu `ldap-authentication`, ktorá zabezpečuje jednoduché pripojenie a autentifikáciu voči vzdialenému LDAP serveru. Knižnica má ako závislosť `ldapjs`, ktorá realizuje nízkoúrovňovú komunikáciu.

Samotná autentifikácia je realizovaná pomocou nasledovnej asynchrónnej funkcie:

```
async function ldapAuth(username, password) {
  try {
    const options = {
      ldapOpts: { url: 'ldap://ldap.stuba.sk' },
      userDn: 'uid=${username},ou=People,dc=stuba,dc=sk',
      userPassword: password,
      userSearchBase: 'ou=People,dc=stuba,dc=sk',
      usernameAttribute: 'uid',
      username: username,
    };

    return await authenticate(options);
  } catch (error) {
    throw new Error('LDAP bind failed: ' + error.message);
  }
}
```

Výpis 4: Funkcia na autentifikáciu používateľa cez LDAP

Na strane backendu používame knižnicu `express-session`, ktorá nám umožňuje uchovávať informáciu o prihlásenom používateľovi v rámci session. Týmto spôsobom vieme zabezpečiť, že používateľ ostane prihlásený aj pri opakovanej návšteve stránky počas aktívnej relácie.

Po úspešnej autentifikácii sa vytvorí zjednodušený objekt používateľa, ktorý sa uloží do databázy (ak ešte neexistuje), a zároveň sa uloží do session nasledovne:

```
app.post('/login', async (req, res) => {
  const { username, password } = req.body;
  try {
```

```

const user = await ldapAuth(username, password);
const processedUser = {
  userId: user.uisId,
  employeeType: user.employeeType,
  givenName: user.givenName,
  lastName: user.sn,
  email: user.mailLocalAddress[1],
};

// Insert into DB if new
try {
  const existing = await db.findUserById(processedUser.userId);
  if (!existing) {
    await db.insertUser(processedUser);
  }
} catch (dbErr) {
  console.error('DB error:', dbErr.message);
}

// Save user to session
req.session.user = processedUser;
res.status(200).json(processedUser);
} catch (err) {
  res.status(401).json({ error: 'Authentication failed: ' + err.message });
}
});

```

Výpis 5: Spracovanie prihlasovania a uloženie používateľa do session

**Ochrana súkromných a administrátorských rout na strane frontendu** Na strane klienta (v Angulari) využívame vlastný súbor `auth.guard.ts`, ktorý zabezpečuje, že k určitým stránkam aplikácie má prístup iba autentifikovaný používateľ. Tento guard je súčasťou smerovania (routing) a overuje, či existuje používateľ v úložisku `sessionStorage`. Ak nie, používateľ je automaticky presmerovaný na prihlasovaciu stránku.

Okrem toho tento guard podporuje aj kontrolu rolí používateľov. Pomocou atribútu `expectedEmployeeType` v definícii trasy vieme zabezpečiť, že niektoré stránky (napr. administrátorské) budú dostupné len používateľom s určitou rolou (napr. `admin`).

```

canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean {
  const user = this.apiService.getUserFromStorage();

```

```

if (!user) {
  this.router.navigate(['/']);
  return false;
}

const expectedEmployeeType = route.data['expectedEmployeeType'];
if (expectedEmployeeType && user.employeeType !== expectedEmployeeType) {
  this.router.navigate(['/menu']);
  return false;
}

return true;
}

```

Výpis 6: Implementácia AuthGuard v Angulari

Definícia ciest sa nachádza v súbore `app-routing.module.ts`, kde sa pomocou `canActivate` zabezpečuje ochrana jednotlivých komponentov. Ako môžeme vidieť na príklade nižšie, každá trasa, ktorá vyžaduje prihlásenie, je chránená pomocou `AuthGuard`. Pre admin rozhrania je navyše špecifikovaný parameter `expectedEmployeeType`, čo zabezpečuje aj kontrolu oprávnení.

```

const routes: Routes = [
  { path: '', component: LoginModalComponent },
  { path: 'menu', component: MenuPageComponent, canActivate: [AuthGuard] },
  { path: 'test', component: TestCreationComponent, canActivate: [AuthGuard] },
  { path: 'stats', component: StatsPageComponent, canActivate: [AuthGuard] },
  { path: 'done', component: TestDoneComponent, canActivate: [AuthGuard] },
  { path: 'mats', component: MaterialsPageComponent, canActivate: [AuthGuard] },
  { path: 'test-writing', component: TestWritingComponent, canActivate: [AuthGuard] },
  },
  { path: 'export', component: ExportPageComponent, canActivate: [AuthGuard] },
  { path: 'admin', component: AdminPageComponent, canActivate: [AuthGuard], data: {
    expectedEmployeeType: 'admin' } },
  { path: 'admin/statistics', component: AdminStatisticsComponent, canActivate: [
    AuthGuard], data: { expectedEmployeeType: 'admin' } },
  { path: '**', redirectTo: '' },
];

```

Výpis 7: Definícia rout v Angulari s ochranou pomocou AuthGuard

Týmto spôsobom zabezpečujeme na klientovi, že sa na súkromné alebo administrátor-ské stránky dostane iba oprávnený používateľ s platným session údajom, čím zvyšujeme

bezpečnosť aplikácie.

### **4.3.2 Posielanie dat ????**

tu napíšeme o ApiService a o AdminService, a taktiež o autogenerovaných veciach ktoré budú ale prepísané na helper servicky

## **4.4 DB**

prepájanie PostgreSQL built in mechanica pre data persistence and backups PostgreSQL allows you to specify a data directory where database files are stored, which you can mount as a volume or bind mount in Docker.\*\*\*\*

## **4.5 Docker**

single vs multiple container approach



## 5 Testovanie

## 6 Zaver

# Záver

Conclusion is going to be where?

Here.

# Zoznam použitej literatúry

1. *Brilliant* [online]. Brilliant. [cit. 2024-12-08]. Dostupné z : <https://www.brilliant.org/>.
2. *Khan Academy* [online]. Khan Academy. [cit. 2024-12-08]. Dostupné z : <https://www.khanacademy.org/>.
3. *Viem matiku* [online]. Vieme to. [cit. 2024-12-08]. Dostupné z : <https://www.viemeto.org/zhrnutie-projektu>.
4. DVOŘÁK, Jan. *Čo to je jazyk HTML a jeho základy*. 2022. Tech. spr. Webglobe. Dostupné tiež z: <https://www.webglobe.sk/blog/html-zaklady>.
5. *CSS*. Coderama. Dostupné tiež z: <https://coderama.com/slovník/css>.
6. BITTNER, Honza. *Úvod do CSS preprocesora Sass*. ITnetwork. Dostupné tiež z: <https://www.itnetwork.sk/html-css/webove-portfolio/tutorial-moderne-webove-portfolio-sass>.
7. *JavaScript*. ITnetwork. Dostupné tiež z: <https://www.itnetwork.sk/javascript>.
8. ŠTRÁFELDA, Jan. *AJAX*. Dostupné tiež z: <https://www.strafelda.cz/ajax>.
9. ŠTRÁFELDA, Jan. *jQuery*. Dostupné tiež z: <https://www.strafelda.cz/jquery>.
10. KVAPIL, Jiří. *Úvod do TypeScriptu*. 2022. Tech. spr. Webglobe. Dostupné tiež z: <https://www.itnetwork.sk/javascript/typescript/uvod-do-typescriptu>.
11. *MathJax*. MathJax. Dostupné tiež z: <https://docs.mathjax.org/en/latest/basic/mathjax.html>.
12. *MathQuill*. MathQuill. Dostupné tiež z: <https://docs.mathquill.com/en/latest/>.
13. SAGE, Rogan. *ApexCharts: How to Build and Manage Customer-Facing Dashboards*. Embeddable, 2025. Dostupné tiež z: <https://embeddable.com/blog/build-dashboards-with-apexcharts>.
14. *Backend*. Coderama. Dostupné tiež z: <https://coderama.com/slovník/backend>.
15. *Čo je API (Application Programming Interface)*. Smarty Academy. Dostupné tiež z: <https://smartyacademy.sk/co-je-api-application-programming-interface/>.
16. HOSSAIN, Monsur. *CORS in Action: Creating and consuming cross-origin apis*. Simon a Schuster, 2014.

17. ING. MIROSLAV GOMBÁR PhD., Mgr. Andrea Hricová. Databázový systém. *Ústav Digitálnych kompetencií*. 2007, roč. 1. Dostupné tiež z: [https://www.unipo.sk/public/media/15344/databazove\\_systemy.pdf](https://www.unipo.sk/public/media/15344/databazove_systemy.pdf).
18. *What is SQL (Structured Query Language)?* Amazon. Dostupné tiež z: <https://aws.amazon.com/what-is/sql/>.
19. WORSLEY, John a DRAKE, Joshua D. *Practical PostgreSQL*. " O'Reilly Media, Inc.", 2002.
20. *Framework*. Coderama. Dostupné tiež z: <https://coderama.com/slovník/framework>.
21. PODMOLÍK, Leopold. *Frontend framework*. visionslab. Dostupné tiež z: <https://visionslabs.io/sk/aky-frontend-framework-zvolit/>.
22. MÁČA, Jindřich. *Úvod do Angular frameworku*. itnetwork. Tech. spr. Dostupné tiež z: <https://www.itnetwork.sk/javascript/angular/zaklady/uvod-do-angular-frameworku>.
23. G., Sanskriti. *Learn RxJs: Introduction*. Mind Browser. Dostupné tiež z: <https://www.mindbrowser.com/learn-rxjs/>.
24. KALUŽA, Marin, KALANJ, Marijana a VUKELIĆ, Bernard. A comparison of back-end frameworks for web application development. *Zbornik veleučilišta u rijeci*. 2019, roč. 7, č. 1, s. 317–332.
25. SYED, Basarat. *Beginning Node.js*. Apress, 2014.
26. HAHN, Evan. *Express in Action: Writing, building, and testing Node.js applications*. Simon a Schuster, 2016.
27. MIKOVÁ, Tereza. *Aký je rozdiel medzi UX a UI, a prečo potrebujete na super web oboje*. 2024. Tech. spr. Webglobe. Dostupné tiež z: <https://www.webglobe.sk/blog/rozdiel-medzi-ux-a-ui>.
28. *Figma*. Coderama. Dostupné tiež z: <https://coderama.com/slovník/figma>.
29. *Angular Material*. Material-UI. Dostupné tiež z: <https://material.angular.io/>.
30. DOUŠKOVÁ, Alena, KARIKOVÁ, Soňa a BLAŽÍČEK, Tomáš. Gamifikácia–špecifikum multimediálneho vzdelávania slovenských učiteľ'ov v zahraničí. *Vzdělávání dospělých 2021*. 2022, s. 110.
31. *Server*. Coderama. Dostupné tiež z: <https://coderama.com/slovník/server>.

32. *Nginx*. Webglobe. Dostupné tiež z: <https://www.webglobe.sk/poradna/co-je-nginx>.
33. *Docker*. Webglobe. Dostupné tiež z: <https://www.webglobe.sk/poradna/co-je-docker>.
34. *Git*. .msgprogramator. Dostupné tiež z: <https://msgprogramator.sk/git-github/>.
35. *Visual Studio Code documentation*. Microsoft. Dostupné tiež z: <https://code.visualstudio.com/docs/getstarted/getting-started>.