

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



ZADANIE BAKALÁRSKEJ PRÁCE

Študent: **Bence Bodnár**
ID študenta: 111184
Študijný program: aplikovaná informatika
Študijný odbor: informatika
Vedúca práce: doc. RNDr. Oľga Nánásiová, PhD.
Vedúci pracoviska: doc. Ing. Milan Vojvoda, PhD.
Miesto vypracovania: Ústav informatiky a matematiky (FEI)

Názov práce: **Matematický trenažér a pravdepodobnosť**

Jazyk, v ktorom sa práca vypracuje: slovenský jazyk

Špecifikácia zadania:

Cieľom práce je pomocou webovej aplikácie podporiť systematické vzdelávanie vo vybraných oblastiach matematiky, špeciálne v teórii pravdepodobnosti. Aplikácia by mala byť nielen pomôckou pri písaní testov a ich vyhodnocovaní, ale aj by mala prispieť k zvýšeniu efektivity vzdelávacieho procesu a poskytnúť študentom moderný a efektívny nástroj na zlepšenie ich matematických schopností a analytického myslenia.

Úlohy:

1. Naštudujte si základy teórie pravdepodobnosti a vlastnosti náhodnej premennej.
2. Zamerajte sa na diskrétne rozdelenia pravdepodobnosti (rovnorné rozdelenie, binomické rozdelenie, hypergeometrické rozdelenie, poissonovo rozdelenie a geometrické rozdelenie pravdepodobnosti).
3. Navrhňte webovú aplikáciu, ktorá bude pomôckou pri tvorbe testov a interaktívnych študijných materiálov.
4. Porovnajte Vaše riešenie s niektorými už existujúcimi webovými aplikáciami.

Zoznam literatúry:

1. NÁNÁSIOVÁ, Oľga; KOHNOVÁ, Silvia. Štatistika a pravdepodobnosť: Základy matematickej štatistiky a teórie pravdepodobnosti. Bratislava: Vydavateľstvo STU, 2016. 199 s. ISBN 978-80-227-4527-7.
2. VOLAUF Peter. Pravdepodobnosť a matematická štatistika. Vydavateľstvo STU, 2014. 195. ISBN 978-80-227-4144-6

Termín odovzdania bakalárskej práce: 06. 06. 2025

Dátum schválenia zadania bakalárskej práce: 17. 04. 2025

Zadanie bakalárskej práce schválil: prof. Dr. rer. nat. Martin Drozda – garant študijného programu

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

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

Obsah

Úvod	1
1 Analýza	2
1.1 Brilliant.org	2
1.2 Khan Academy	2
1.3 Vieme matiku	3
1.4 Zhodnotenie	3
2 Použité technológie a knižnice	4
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 TypeScript	6
2.1.8 MathJax	6
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	9
2.3.3 DBDiagram	10
2.4 Framework	10
2.4.1 Frontendové frameworky	10
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	12
2.5.1	Figma	13
2.5.2	Material UI	13
2.6	Gamifikácia	13
2.6.1	Gamifikácia v e-learningu	13
2.7	Server	14
2.7.1	NGINX	14
2.7.2	Docker	14
2.8	GIT	14
3	Návrh webovej aplikácie	16
3.1	Špecifikácia	17
3.2	Funkcionálne požiadavky	18
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	27
3.7.1	Gamifikácia	28
3.7.2	Generovanie úloh (TODO??)	30
3.8	Materiály	31
3.8.1	Interaktívne grafy	32
3.9	Vyhodnocovanie úloh	33
3.10	Vytváranie a úprava úloh administrátorom	34
3.11	Export testov a výsledkov	35
4	Implementácia webovej aplikácie	37
4.1	Vývojové prostredie a inštalácia závislostí	37
4.2	Frontend – Angular	37
4.2.1	Implementácia grafickej časti	41

4.2.2	Vizualizácia matematických výrazov pomocou MathJax	41
4.2.3	Zadávanie matematických výrazov s MathQuill	42
4.3	Export úloh do PDF formátu pomocou pdfmake	42
4.4	Vizualizácia dát pomocou ApexCharts	43
4.5	Backend	45
4.5.1	Autentifikácia	47
4.6	Prenos a spracovanie dát	50
4.6.1	Generovanie úloh	52
4.7	Databáza	53
4.8	Dockerizácia aplikácie ??? napicuje aj toto	53
5	Testovanie	57
	Záver	58
	Zoznam použitej literatúry	59

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.	26
Obrázok 8	Používateľské rozhranie stránky pre vytvorenie testovacej sady. .	27
Obrázok 9	Používateľské rozhranie stránky počas priebehu testu.	28
Obrázok 10	Ukážka vizuálnej spätnej väzby v gamifikovanom režime testovania.	30
Obrázok 11	Ukážka sekcie študijných materiálov s príkladom.	32
Obrázok 12	Ukážka interaktívneho grafu.	33
Obrázok 13	Ukážka podstránky vyhodnocovania testu a zobrazenie dosiahnutého percentilu.	34
Obrázok 14	Ukážka administrátorského rozhrania na úpravu úloh.	35
Obrázok 15	Ukážka vygenerovaného PDF súboru so sadou úloh na precvičovanie.	36
Obrázok 16	Výstup príkazu <code>ng version</code> v termináli.	38
Obrázok 17	Základná štruktúra Angular projektu.	40
Obrázok 18	Naša štruktúra projektu.	40
Obrázok 19	Ukážka zobrazenia matematického výrazu pomocou MathJax. .	42
Obrázok 20	Ukážka štruktúry backendových služieb v aplikácii.	46
Tabuľka 1	Vzdelávacie platformy	4

Zoznam výpisov

1	LaTeX zápis matematického výrazu	35
2	Zjednodušená štruktúra PDF dokumentu	42
3	Odstránenie LaTeX syntaxe zo zadania úlohy	43
4	Výpočet bodov Studentovho rozdelenia	44
5	Vykreslenie grafu v šablóne pomocou ApexCharts	44
6	Ukážka základnej konfigurácie Express.js servera	45
7	Konfigurácia CORS pre komunikáciu medzi FE a BE	46
8	Použitie body-parser pre spracovanie JSON požiadaviek	47
9	Funkcia na autentifikáciu používateľa cez LDAP	47
10	Spracovanie prihlasovania a uloženie používateľa do session	48
11	Implementácia AuthGuard v Angulari	49
12	Definícia rout v Angulari s ochranou pomocou AuthGuard	49
13	Volanie API na vytvorenie novej úlohy	50
14	Serverová logika na spracovanie POST požiadavky pre vytvorenie úlohy . .	51
15	Ukážka generovania úloh s binomickým rozdelením	52
16	Ukážka pripojenia na PostgreSQL databázu pomocou knižnice pg	53
17	Ukážka Dockerfile pre Angular aplikáciu	54
18	Ukážka docker-compose.yml pre multi-container systém	55

Úvod

Štúdium matematickej štatistiky a teórie pravdepodobnosti zohráva významnú úlohu pri porozumení a analýze náhodných javov a dát v rozličných oblastiach, akými sú ekonómia, vedecký výskum alebo medicína. Tieto matematické disciplíny zahŕňajú široké spektrum tém, medzi ktoré patrí analýza údajov pomocou modusu, mediánu či stredovej hodnoty, práca s náhodnými premennými, kombinatorika, alebo podmienená pravdepodobnosť. Každá z týchto oblastí poskytuje unikátne nástroje na kvantifikáciu, interpretáciu a predikciu rôznych javov a trendov.

S cieľom podporiť systematické a efektívne vzdelávanie v týchto kľúčových oblastiach sme sa rozhodli navrhnúť a implementovať trojvrstvovú webovú aplikáciu dostupnú pomocou moderných technológií. Frontendová časť aplikácie bola vyvinutá pomocou frameworku Angular v kombinácii s knižnicou Material UI, zatiaľ čo backend využíva Node.js s frameworkom Express.js. Táto aplikácia využíva interaktívne a dynamické metódy učenia, ako sú interaktívne grafy, automaticky generované testy a moderné gamifikačné prvky, čím zlepšuje angažovanosť študentov a podporuje jednoduchšie osvojenie si matematických konceptov.

Cieľom práce je nielen vytvoriť intuitívny a dostupný nástroj na učenie sa matematickej štatistiky a pravdepodobnosti, ale aj zvýšiť prístupnosť a zrozumiteľnosť týchto náročných tém pre študentov. Aplikácia má ambíciu výrazne prispieť k efektívnejšiemu vzdelávaciemu procesu a rozvíjať analytické schopnosti a matematické myslenie používateľov prostredníctvom moderných a interaktívnych výučbových metód.

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 oblastiach vedy, či už matematickej alebo počítačovej. 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 učniu prostredníctvom gamifikácie¹. Nevýhodou platformy 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]

¹Gamifikácia je využitie herných prvkov a mechaník v nehernom prostredí s cieľom zvýšiť motiváciu a participáciu 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

Analýza v súčasnosti existujúcich vzdelávacích platforiem nás priviedla k zisteniu, že na trhu chýbajú lokalizované a cenovo dostupné e-learningové riešenia pre stredoškolských a vysokoškolských študentov, ktoré by účinne kombinovali gamifikáciu, interaktivitu a prehľadné rozhranie. Existujúce platformy, ako Brilliant.org a Khan Academy, síce 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 značkový jazyk (HyperText Markup Language) (HTML) je značkový jazyk používaný na tvorbu a štruktúrovanie obsahu webových stránok. Je schopný 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) je štýlovací jazyk používaný na definovanie vzhľadu a formátovania webových stránok. Slúži na oddelenie vizuálnej prezentácie 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. Podporuje taktiež 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.[5]

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, a tak uľahčuje 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 poskytuje možnosť tvorby komplexných štýlových štruktúr.[6]

2.1.4 JavaScript

JavaScript je interpretovaný programovací jazyk, ktorý slúži na 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.[7]

2.1.5 AJAX

AJAX (Asynchronous JavaScript and XML) je technológia, cez ktorú môžu webové aplikácie 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, našeptávačoch 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 TypeScript

TypeScript je programovací jazyk vyvinutý spoločnosťou Microsoft, ktorý rozširuje možnosti JavaScriptu pridaním statického typovania a pokročilých objektovo orientovaných prvkov. Tým pomáha 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.[10]

2.1.8 MathJax

MathJax je open-source² JavaScriptový engine určený na zobrazovanie matematických notácií, ako sú LaTeX, MathML a AsciiMath, v moderných webových prehliadačoch.

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

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.[11]

2.1.9 MathQuill

MathQuill je open-source webový editor matematických výrazov navrhnutý tak, aby bol schopný jednoducho a esteticky zadávať matematiku 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.[12]

2.1.10 ApexCharts

ApexCharts 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.[13]

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 funkcionality aplikácie, 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 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-stack³ development, kde programátori pracujú na oboch stranách aplikácie. V minulosti backend pozostával prevažne z jednoduchých serverových skriptov,

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

avšak s rozvojom webových technológií sa využívajú pokročilé frameworky umožňujúce dynamickú generáciu obsahu. Efektívny backendový kód je kľúčový pre optimalizáciu výkonu aplikácie, minimalizáciu zaťaženia servera a databázy, a zabezpečenie rýchlej odozvy pre používateľov. [14]

2.2.1 API

Rozhranie pre programovanie aplikácií (API) je rozhranie, ktoré slúži na komunikáciu jednej aplikácie s inou aplikáciou alebo systémom. Umožňuje vývojárom využívať už existujúce funkcionality bez toho, aby ich museli znova vytvárať, čím urýchľuje a simplifikuje tak 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, vďaka ktorému môžu medzi sebou aplikácie komunikovať 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 zabezpečuje 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, akým môžu prehliadače bezpečne vykonávať požiadavky na zdroje umiestnené na iných serveroch, než odkiaľ pochádza pôvodná webová stránka. V minulosti takýmto požiadavkam bránila politika rovnakého pôvodu (same-origin policy), ktorá obmedzovala skripty bežiace v prehliadači výhradne na komunikáciu so serverom tej istej domény.

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 napomáhajú prehliadaču určiť, či konkrétna požiadavka medzi rôznymi doménami môže byť bezpečne vykonaná.[16]

2.2.4 Node.js

Node.js je runtime⁴ 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 dokážu vytvoriť efektívne webové aplikácie. 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 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, ktorý umožňuje ich optimalizované 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 viacerých tabuliek. Dotazy sú základom práce s SQL a vedia 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

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

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, vďaka čomu ponúka vysokú mieru prispôsobiteľnosti pre náročné aplikácie. Je navrhnutý s dôrazom na integritu dát, rozširiteľnosť 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 vývojárom a dátovým analytikom slúži na jednoduché vytváranie diagramov databázových štruktúr pomocou nekomplikovaného textového zápisu. Jeho hlavnou výhodou je schopnosť rýchlo a efektívne navrhovať a vizualizovať databázové schémy, a tak uľahčiť 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 zabezpečuje 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,

optimalizuje a na základe tohto disponuje možnosťou tvorby 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. Pomáha vývojárom vytvárať aplikácie s bohatou funkcionalitou prostredníctvom komponentovo orientovanej architektúry, ktorá podporuje opätovné použitie kódu a optimalizuje 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 rozdeľuje aplikácie na menšie, ľahko spravovateľné časti. Angular tiež obsahuje nástroje na správu formulárov, komunikáciu so serverom a smerovanie, vďaka čomu dokážu vývojári vytvárať komplexné aplikácie s minimálnym úsilím. [22]

2.4.4 RxJs

RxJs Reactive Extensions for JavaScript (RxJs) je knižnica určená na reaktívne programovanie, ktorá zabezpečuje 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]

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 optimalizuje 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.[25][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 rýchlo 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 dve rozličné disciplíny, úzko spolupracujú s cieľom vytvoriť digitálne produkty, ktoré sú však taktiež používateľsky prívetivé.[26]

2.5.1 Figma

Figma je cloudová platforma slúžiaca na vytvorenie návrhu 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, a tento fakt spôsobuje, že je k nim lepší prístup a dajú sa jednoduchšie zdieľať. [27]

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, a tak dokážu vývojári 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. [28]

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 odmen, 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"[29] 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. Dajú sa na ňom 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 – existuje napríklad súborový, databázový, aplikačný alebo webový server. Správne nastavený server je základom spoľahlivej a bezpečnej IT infraštruktúry.[30]

2.7.1 NGINX

Nginx[31] je výkonný a spoľahlivý webový a reverzný proxy server, ktorý je navrhnutý na rýchle 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, a tak zabezpečiť 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, a vďaka tomuto dokáže spúšťať viacero kontajnerov naraz s nízkymi nárokmi na systémové prostriedky.[32]

2.8 GIT

Git je distribuovaný systém slúžiaci na správu verzií, ktorým sa dajú 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.[33]

3 Návrh webovej aplikácie

Naša webová aplikácia, koncipovaná ako dvojvrstvový systém, pozostáva z časti backendovej - vytvorenej pomocou platformy Express - a frontendovej - implementovanej pomocou frameworku Angular. Fakt, že sme obidve časti navrhli ako samostatné a na sebe nezávislé aplikácie, nám v procese vytvárania umožnil väčšiu flexibilitu nielen v oblasti samotného vývoja, ale aj testovania či nasadzovania aplikácie na server.

Ako sme už spomenuli v predchádzajúcich kapitolách, backendová časť je zodpovedná za spracovanie údajov, manipuláciu s databázou, LDAP autentifikáciu používateľov a spracovanie požiadaviek zo strany klienta. Ide o akúsi integračnú vrstvu medzi používateľským rozhraním a databázou, ktorá zabezpečuje bezpečný a efektívny prenos údajov.

Backend sme implementovali pomocou frameworku Express.js, postavenom na Node.js. Dôvodom bola nielen jeho funkcia jednoducho vytvárať REST API rozhrania, ale aj dispozícia mnohých ďalších výhod, akými sú 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 plní Express úlohu backendovej vrstvy poskytujúcej API endpointy, prostredníctvom ktorých frontend získava dynamicky obsah a dáta. Táto kombinácia zaisťuje rýchlu a plynulú interakciu používateľa s aplikáciou bez potreby neustáleho obnovovania stránky, čím sa zvyšuje používateľský komfort a celkový výkon systému.

Na efektívne riadenie oprávnení a zabezpečenie rôznych úrovní prístupu k funkcionalitám aplikácie slúži LDAP autentifikácia. Tá overuje identitu používateľov a priraduje im prístupové práva na základe ich roly, ktorými sú v našej aplikácii študent a administrátor (učiteľ).

Frontend sme navrhli ako Single Page Application (SPA) s použitím frameworku Angular. Ten sme vybrali pre jeho výbornú škálovateľnosť a komponentový prístup, ktorý umožňuje vytvárať zapuzdrené (encapsulované) komponenty. Vďaka tomuto sa kód lepšie organizuje a je viac čitateľný a lepšie sa preto pracuje na jeho údržbe a opätovnom použití.

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 centrálné spracovanie logiky a dát. Každá

Angular služba je singleton⁵ objekt a zdieľa dáta a funkcionality naprieč celou aplikáciou, vďaka čomu sa signifikantne zvyšuje modularita systému a podporuje sa znovupoužiteľnosť kódu.

Pri asynchrónnej komunikácii a práci s údajmi sme v Angulari využili mechanizmus Observables, ktorý je súčasťou knižnice RxJs. Tento mechanizmus dokáže efektívne pracovať s dátovými tokmi, ako sú napríklad HTTP požiadavky alebo používateľské vstupy, a poskytuje pokročilé možnosti týkajúce sa reaktívneho spracovania dát, zrušenia požiadaviek, spájania viacerých streamov alebo transformácie 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 diskkrétne 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 časťou ďalších kapitol tejto práce.

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

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.

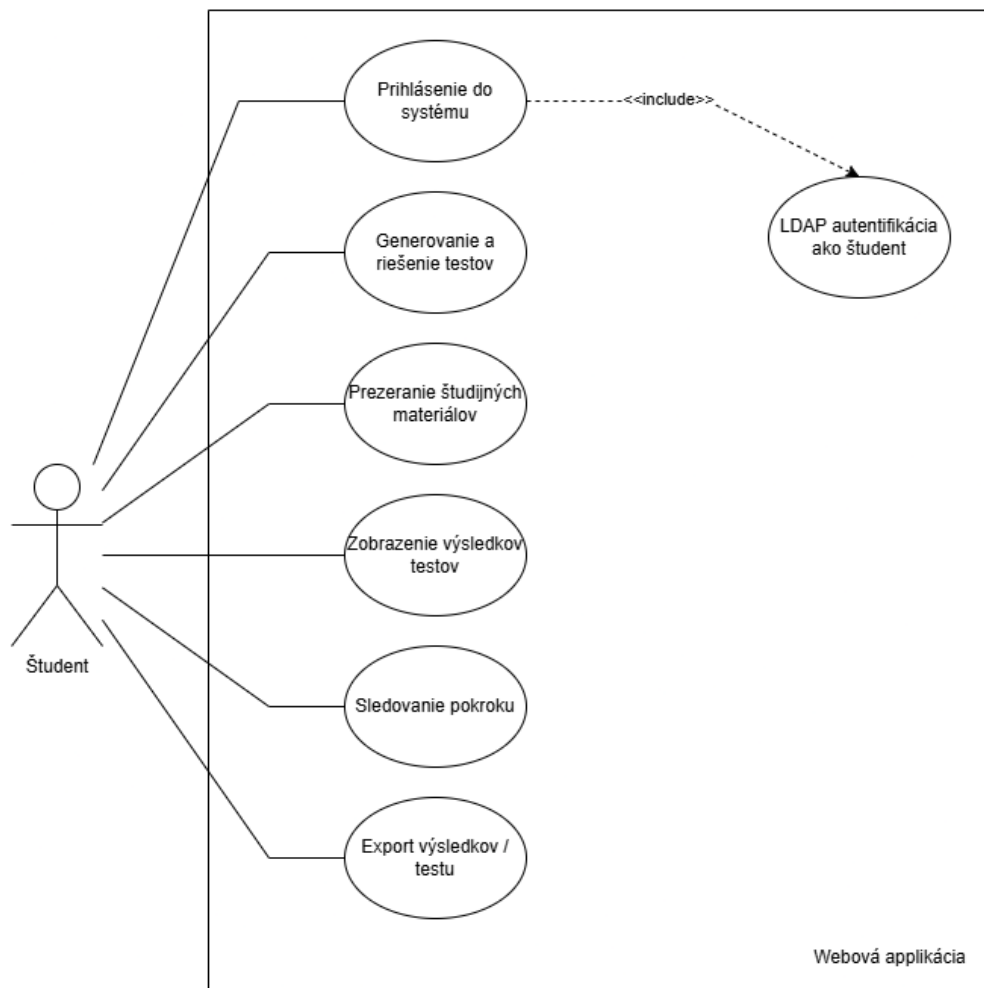
3.3 Nefunkcionálne požiadavky

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.
- Spoločiteľné 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.

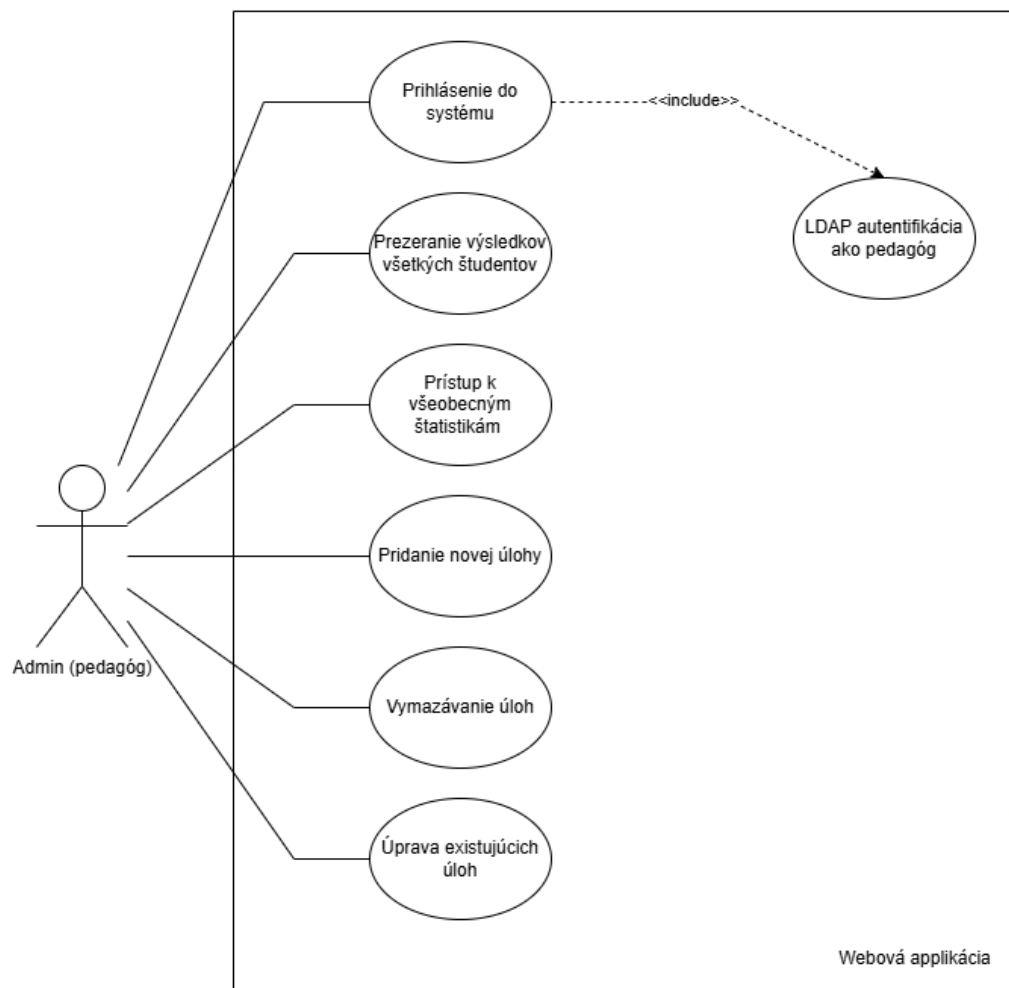
3.4 Diagramy prípadov použitia

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, vďaka ktorej vieme 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.

Okrem uvedených výhod PostgreSQL veľmi dobre spolupracuje s kontajnerizačnou platformou **Docker**. Prostredníctvom Docker kontajnerov môžeme jednoducho spustiť databázu s vopred definovanou štruktúrou a údajmi. Docker taktiež dokáže automatizovať tento proces pomocou bash skriptu, ktorý načíta inicializačné SQL skripty pri spustení kontajnera. Vďaka tomu je databáza okamžite pripravená na používanie v potrebnej konfigurácii ihneď po spustení kontajnera.

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.



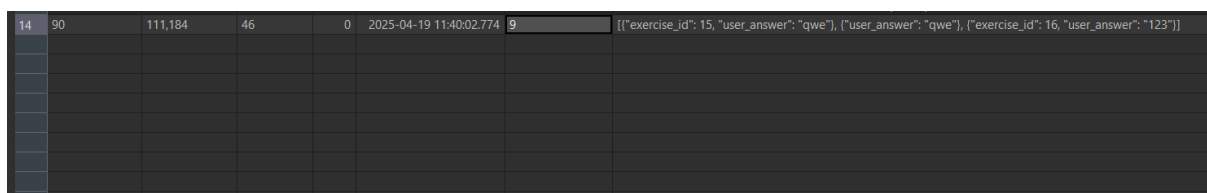
Obr. 3: Logický ER diagram (Entity-Relationship Diagram) databázového modelu.

Databáza sa skladá z nasledovných tabuliek:

- *users* – uchováva informácie o prihlásených používateľoch. Keďže autentifikáciu realizujeme 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.

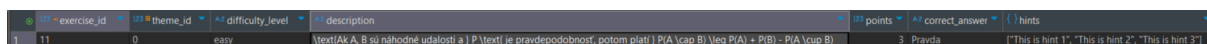


14	90	111,184	46	0	2025-04-19 11:40:02.774	9	[{"exercise_id": 15, "user_answer": "qwe"}, {"exercise_id": 16, "user_answer": "123"}]

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

Každé zadanie úlohy (*description*) je uložené ako reťazec vo formáte LaTeX, ktorého prednosťou je 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	$\text{Ak } A, B \text{ sú náhodné udalosti a } P \text{ text} \text{ je pravdepodobnosť, potom platí } P(A \cup B) \leq P(A) + P(B) - P(A \cap B)$	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

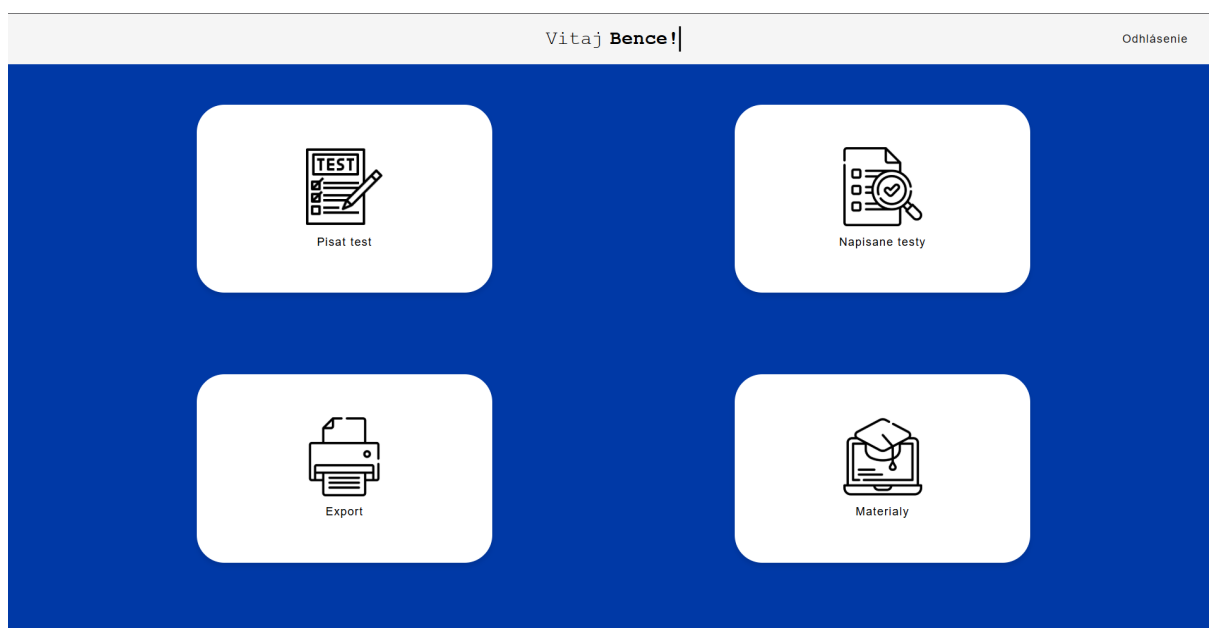
Používateľské rozhranie aplikácie bolo navrhnuté s dôrazom na prehľadnosť, jednoduchosť a intuitívnu navigáciu, čím sme chceli docieľiť zníženie kognitívnej záťaže používateľa a zvýšiť celkovú používateľskú skúsenosť (UX). Cieľom návrhu bolo zabezpečiť, aby sa

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 poskytol 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 použiteľnosť používateľského rozhrania.



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 bola používateľská skúsenosť čo najlepšia (UX).

The image shows a user interface for creating a test set. It is enclosed in a blue border. The interface consists of several sections: a top section for 'Celkový počet úloh' (Total number of tasks) with a text input field showing '0'; a section for 'Výber tém' (Topic selection) with five checkboxes: 'Uncategorized', 'Pravdepodobnosť a rozdelenia', 'Štatistika a analýza dát', 'Kombinatorika a teória množín', and 'Aplikácie a reálne scenáre'; a section for 'Typ úloh' (Task type) with three sliders for 'Ľahké' (Easy), 'Stredné' (Medium), and 'Ťažké' (Hard), each with a text input field showing '0'; and a bottom section with a 'časový limit' (time limit) dropdown menu set to '5 minút', a 'Gamifikácia' (Gamification) toggle switch, and a blue 'Generovať' (Generate) button.

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

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

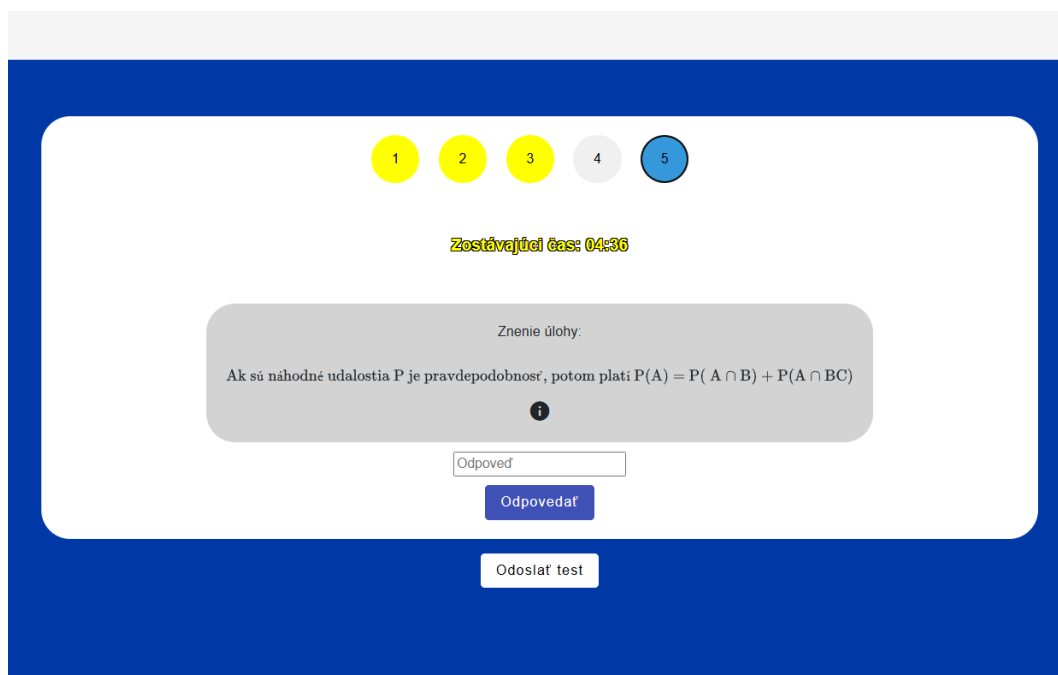
- **Výber tém** – používateľ si môže označiť matematické témy, z ktorých chce byť testovaný. Ide najmä o témy ako pravdepodobnosť a rozdelenia, štatistika a analýza dát, kombinatorika a teória množín, a tiež aplikácie a reálne scenáre.
- **Typ a obtiažnosť úloh** – umožňuje nastaviť počet ľahkých, stredne 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 9). 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.



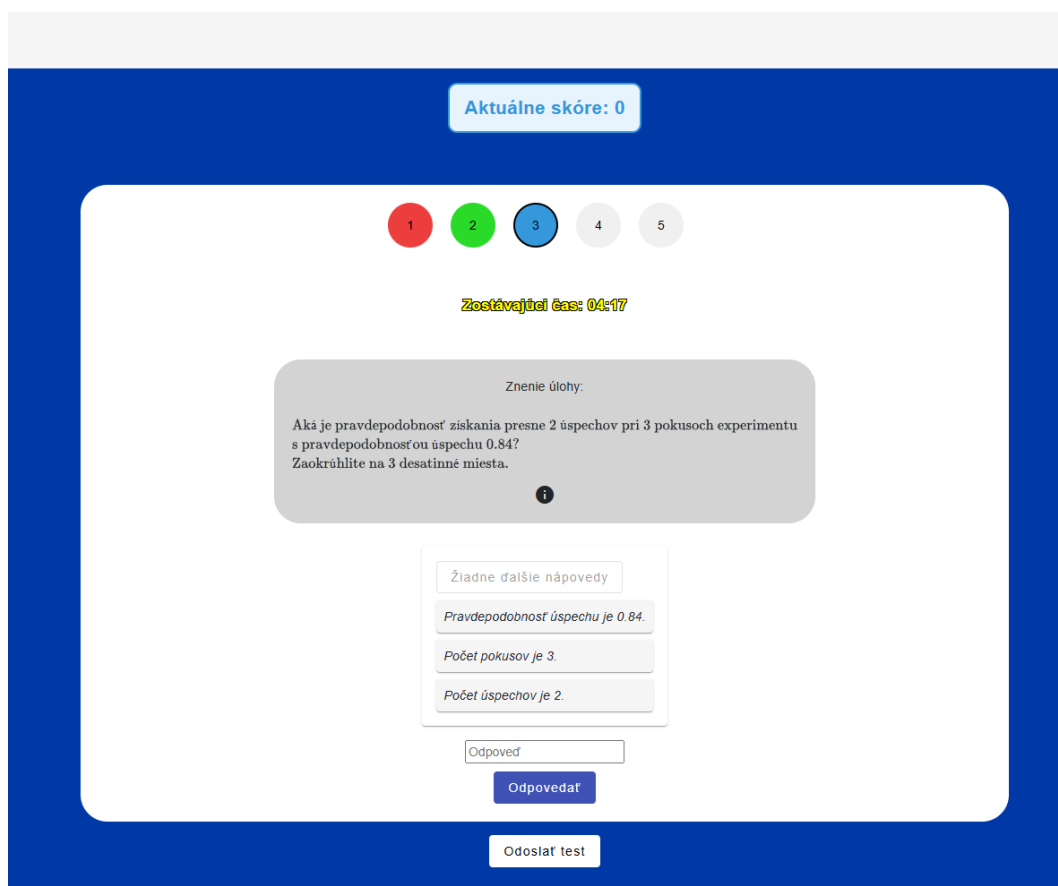
Obr. 9: Používateľské rozhranie stránky počas priebehu testu.

3.7.1 Gamifikácia

S cieľom zvýšiť atraktivnosť vzdelávacieho procesu a posilniť motiváciu študentov bola do aplikácie implementovaná **gamifikácia**, teda využitie herných prvkov v nehernej kontexte. Gamifikačné mechanizmy majú potenciál výrazne zlepšiť angažovanosť študentov, podporiť ich súťaživosť a vytvoriť pozitívne motivujúce prostredie na systematické precvičovanie vedomostí.

V rámci navrhnutej aplikácie sme zaviedli nasledujúce gamifikačné prvky:

- **Percentilové hodnotenie** – namiesto klasického zoradenia používateľov podľa počtu bodov aplikácia využíva hodnotenie na základe **percentilu**. Tento ukazovateľ vyjadruje, aké percento ostatných študentov dosiahlo horší alebo rovnaký výsledok. Zobrazenie percentilu poskytuje študentovi okamžitú a objektívnu informáciu o jeho relatívnej výkonnosti v rámci všetkých účastníkov testovania a zároveň ho motivuje k ďalšiemu zlepšovaniu. Ukážku zobrazovania percentilového hodnotenia možno vidieť na obrázku 13.
- **Bodový systém s bonusmi** – používateľ získava štandardné body za každú správne zodpovedanú otázku. Okrem toho sú pridelené aj bonusové body za rýchlosť odpovede, čím sa študenti podporujú v efektívnom a rýchlom riešení úloh.
- **Vizuálna a zvuková spätná väzba** – aplikácia okamžite reaguje na správne alebo nesprávne odpovede študenta. Vizuálne je táto spätná väzba reprezentovaná farebným označením stavu jednotlivých otázok, pričom je doplnená aj vhodnými zvukovými efektmi. Týmto spôsobom dostáva používateľ jasnú a jednoznačnú informáciu o svojej aktuálnej úspešnosti. Čo je zobrazené na obrázku 10.



Obr. 10: Ukážka vizuálnej spätnej väzby v gamifikovanom režime testovania.

- **Nápovedy s penalizáciou** – v prípade, že používateľ potrebuje pomôcť s riešením úloh, má možnosť využiť nápovedu. Tá ho môže nasmerovať k správnejmu riešeniu, avšak za každú použitú nápovedu sú používateľovi odpočítané body. Týmto spôsobom podporujeme študentov k samostatnosti a zároveň pridávame do vzdelávania súťažný rozmer.

Na stránke so štatistikami má študent následne možnosť vidieť, v akom percentile výkonnosti sa nachádza, čo mu poskytuje objektívnu informáciu o jeho relatívnej úspešnosti voči ostatným používateľom aplikácie. Tieto prvky spolu vytvárajú efektívne a interaktívne prostredie, ktoré podporuje kontinuálne vzdelávanie a rozvoj matematických schopností študentov.

3.7.2 Generovanie úloh (TODO??)

Pri návrhu aplikácie sme sa rozhodli implementovať mechanizmus **automatického generovania úloh**, ktorého cieľom bolo zvýšiť variabilitu testov a zabezpečiť, aby používatelia

neboli limitovaní iba na statické úlohy uložené v databáze. Takto generované úlohy pridávajú do testovania prvok náhodnosti a prekvapenia, čím dochádza k efektívnejšiemu prevereniu skutočných vedomostí a schopností študenta. Generátor úloh je navrhnutý tak, aby dynamicky vytváral nové zadania založené na vybraných pravdepodobnostných rozdeleniach. V súčasnej verzii aplikácie podporujeme generovanie úloh z nasledujúcich rozdelení:

- **Binomické rozdelenie**
- **Geometrické rozdelenie**
- **Hypergeometrické rozdelenie**
- **Poissonovo rozdelenie**
- **Rovnomerné rozdelenie**

Každá vygenerovaná úloha je unikátna, pričom vstupné parametre ako počet pokusov, počet úspechov alebo pravdepodobnosť úspechu sú generované náhodne v definovanom rozsahu.

3.8 Materiály

Ďalšou súčasťou aplikácie je sekcia študijných materiálov, ktorá slúži študentom ako podpora počas štúdia matematickej štatistiky a pravdepodobnosti. Táto časť aplikácie obsahuje systematicky usporiadané študijné poznámky, ktoré pokrývajú učebné osnovy predmetu Matematická štatistika. Obsah je kategorizovaný podľa jednotlivých tematických oblastí, a tak sa dá v študovanom obsahu jednoducho orientovať.

Každá téma obsahuje teoretické poznatky spolu s jasne spracovanými vzorovými príkladmi. Súčasťou týchto príkladov je aj detailný postup výpočtu, ktorý študentom slúži ako metodická pomôcka pri riešení podobných úloh.

Diskrétné rozdelenia

Alternatívne, Binomické, Poissonovo, Geometrické, Rovnomerné a Hypergeometrické rozdelenie

Alternatívne rozdelenie:

- Definícia:** Pre dve možnosti (napr. úspech alebo neúspech) s pravdepodobnosťou p pre úspech: $P(A) = p$, $P(X = x_1) = p$, $P(X = x_2) = 1 - p$.
- Očakávaná hodnota:** $E(X) = x_1p + x_2(1 - p) = p$.
- Rozptyl:** $D(X) = (x_1 - x_2)^2p(1 - p) = p(1 - p)$.

Binomické rozdelenie:

- Pre nezávislé pokusy s pravdepodobnosťou úspechu p : $P(X = k) = \binom{n}{k}p^k(1 - p)^{n-k}$.
- Očakávaná hodnota:** $E(X) = np$.
- Rozptyl:** $D(X) = np(1 - p)$.

Poissonovo rozdelenie:

- Poissonova veta:** Pre binomické rozdelenie, kde $n \rightarrow -\infty$ a $p \rightarrow 0$, platí: $\lim_{n \rightarrow \infty} P(X_n = k) = \frac{\lambda^k e^{-\lambda}}{k!}$, pre $k = 0, 1, 2, \dots$
- Distribučná funkcia:** $P(X \leq n) = \sum_{k=0}^n \frac{\lambda^k e^{-\lambda}}{k!}$.
- Očakávaná hodnota a rozptyl:** $E(X) = D(X) = \lambda$.

Geometrické rozdelenie:

- Definícia:** Ak X je počet neúspešných pokusov pred úspechom: $P(X = k) = (1 - p)^k p$.
- Očakávaná hodnota:** $E(X) = \frac{1-p}{p}$.
- Rozptyl:** $D(X) = \frac{1-p}{p^2}$.

Rovnomerné rozdelenie:

- Definícia:** Pre rovnomerne distribuovanú náhodnú premennú $X \in x_1, x_2, \dots, x_n$, s pravdepodobnosťou: $P(X = x_i) = \frac{1}{n}$, pre všetky i .
- Očakávaná hodnota:** $E(X) = \frac{1}{n} \sum_{i=1}^n x_i$.
- Rozptyl:** $D(X) = \frac{1}{n} \sum_{i=1}^n x_i^2 - E(X)^2$.

Hypergeometrické rozdelenie:

- Definícia:** Ak máme N prvkov a k úspešných prvkov, vyberáme n náhodných prvkov bez vrátenia:

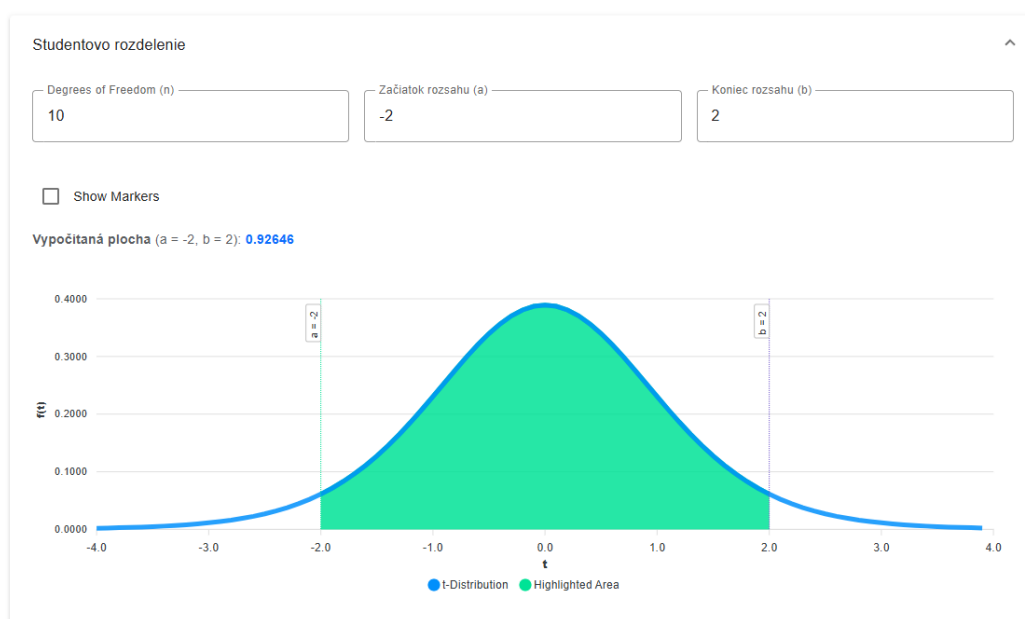
$$P(X = i) = \frac{\binom{k}{i} \binom{N-k}{n-i}}{\binom{N}{n}}, \quad \text{pre } i = \max(0, k + n - N) \dots \min(k, n).$$
- Príklad:** Pravdepodobnosť, že medzi 5 náhodne vybranými študentmi budú 3 absolventi: $P(X = 3) = \frac{\binom{20}{3} \binom{80}{2}}{\binom{100}{5}} = 0.3854$.

Viac môžete dočítať tu: [Diskrétné rozdelenia](#)

Obr. 11: Ukážka sekcie študijných materiálov s príkladom.

3.8.1 Interaktívne grafy

Na zvýšenie interaktivity a pochopenia zložitých matematických konceptov sú v rámci materiálov integrované aj interaktívne grafy. Tieto grafy vizuálne demonštrujú dôležité pravdepodobnostné princípy, ako napríklad priebehy distribučných funkcií alebo hustoty pravdepodobnosti rôznych rozdelení. Grafické znázornenie výrazne uľahčuje študentom pochopenie abstraktných matematických myšlienok a prehľbuje ich schopnosť aplikovať získané vedomosti v praxi.



Obr. 12: Ukážka interaktívneho grafu.

3.9 Vyhodnocovanie úloh

Po dokončení testu má používateľ k dispozícii detailný rozbor svojho riešenia. Na tejto podstránke je možné spätne prehliadať absolvované testy, pričom ku každému testu je zobrazovaný zoznam jednotlivých úloh s vyznačením správnosti odpovede.

Každú úlohu je možné rozkliknúť a prezrieť si zadanie spolu s odpoveďou, čo podporuje proces spätného učenia a upevňovania vedomostí.

Okrem samotného zoznamu úloh je súčasťou zobrazenia aj základné štatistické zhrnutie testu, ako napríklad počet správnych odpovedí, celkový počet získaných bodov a dosiahnutý percentil.

Percentil poskytuje používateľovi okamžitú informáciu o jeho relatívnom postavení medzi všetkými účastníkmi testovania, a to na základe získaného skóre.

Zoradiť podľa Test ID	Zoradiť podľa Bodov	Zoradiť podľa Dátumu
Test ID: 46	Získané body: 0 / 9	Dátum: 4/19/25, 11:40 AM
Test ID: 45	Získané body: 0 / 12	Dátum: 4/19/25, 10:43 AM
Test ID: 44	Získané body: 0 / 12	Dátum: 4/19/25, 10:17 AM
Test ID: 43	Získané body: 0 / 12	Dátum: 4/18/25, 3:08 PM
Test ID: 38	Získané body: 3 / 15	Dátum: 4/7/25, 11:49 AM
Test ID: 35	Získané body: 8 / 15	Dátum: 3/27/25, 11:15 PM
Test ID: 33	Získané body: 0 / 9	Dátum: 3/27/25, 11:15 PM
Test ID: 31	Získané body: 0 / 12	Dátum: 3/27/25, 11:02 PM
Test ID: 28	Získané body: 0 / 9	Dátum: 3/27/25, 10:04 PM
Test ID: 26	Získané body: 0 / 9	Dátum: 3/27/25, 10:03 PM

Items per page: 10 1 - 10 of 14

Nižšie nájdete podrobné štatistiky a informácie o všetkých odovzdaných testoch vrátane témy každej úlohy.

Tvoj celkový výkon je v 27.3% percentile.

Obr. 13: Ukážka podstránky vyhodnocovania testu a zobrazenie dosiahnutého percentilu.

3.10 Vytváranie a úprava úloh administrátorom

V rámci administrátorskej časti aplikácie bola implementovaná funkcionality umožňujúca správcovi systému vytvárať nové úlohy a upravovať existujúce zadania. Cieľom tejto funkcionality je umožniť dynamické rozširovanie databázy úloh bez potreby manuálnych zásahov do databázy na úrovni servera.

Pri návrhu rozhrania na vkladanie úloh bolo dôležité zabezpečiť, aby bol proces tvorby matematických otázok intuitívny a používateľsky prívetivý, najmä pri práci s matematickými zápsmi a vzorcami. Tento cieľ bol dosiahnutý integráciou knižnice **MathQuill**, ktorá umožňuje dynamické a interaktívne zadávanie matematických výrazov.

MathQuill poskytuje rozhranie, v ktorom môže administrátor zadávať matematické vzorce v LaTeX formáte. V reálnom čase dochádza k ich prekladu do vizuálne čitateľnej podoby, čím sa zabezpečuje okamžitá spätná väzba a minimalizácia syntaktických chýb. Matematické výrazy sú zobrazované v editovateľnom textovom poli, ktoré verne imituje klasickú matematickú notáciu, čo výrazne zvyšuje používateľský komfort a presnosť zadávania úloh.

Táto funkcionality nielen zjednodušuje proces tvorby úloh, ale zároveň zaručuje, že všetky matematické zápisy budú správne interpretované pri zobrazovaní úloh študentom v testoch a materiáloch.

Upraviť úlohu

Znenie úlohy

Ak A, B sú náhodné udalosti a P je pravdepodobnosť, potom platí $P(A \cap B) \leq P(A) + P(B) - P(A \cup B)$

Správna odpoveď*

Pravda

Body*

3

Obtiažnosť*

Easy

Nápoved' (jeden na riadok)*

This is hint 1
This is hint 2
This is hint 3

Close Confirm

Obr. 14: Ukážka administrátorského rozhrania na úpravu úloh.

Na obrázku 14 je možné vidieť instantný preklad pomocou knižnice MathQuill pri editovaní úlohy. Používateľ zadáva výraz v LaTeX formáte, ktorý sa okamžite zobrazí vo vykreslenej matematickej forme.

Použitý zápis:

```
\text{Ak } A, B \text{ sú náhodné udalosti a } P \text{ je pravdepodobnosť, potom} \\ \text{platí } P(A \cap B) \leq P(A) + P(B) - P(A \cup B)
```

Výpis 1: LaTeX zápis matematického výrazu

3.11 Export testov a výsledkov

V rámci rozšírenia funkcionality aplikácie bola implementovaná sekcia **Export**, ktorá reflektuje potreby a preferencie študentov preferujúcich tradičné metódy učenia, ako je štúdium pomocou papiera a pera aj v digitálnej ére.

Stránka Export obsahuje dve hlavné funkcionality:

- **Generovanie testu do PDF formátu** – používateľ má možnosť vygenerovať si novú sadu úloh podobne ako v režime preskúšania. Úlohy sú obohatené o nápovedy a následne spracované do PDF dokumentu, ktorý si študent môže vytlačiť a riešiť v

offline prostredí. Táto funkcionálnosť je určená najmä pre tých, ktorí uprednostňujú fyzický výpočet úloh a písanie riešení na papier.

- **Export vypracovaných výsledkov** – študent si môže stiahnuť aj prehľad svojich absolvovaných testov vrátane odpovedí, výsledkov a štatistík. Tento export umožňuje jednoduchú archiváciu pokroku a spätnú analýzu riešených úloh.

Obe PDF súbory sú generované dynamicky na základe dát získaných z aplikácie, pričom na tvorbu dokumentov je využívaná knižnica **pdfmake**. Štruktúra vygenerovaných súborov bola navrhnutá tak, aby bola prehľadná, čitateľná a používateľsky prívetivá.

Na obrázku 15 je ukážka časti vygenerovaného PDF dokumentu obsahujúceho zadania úloh.

Generované úlohy – Strana 1 z 1		
Generované úlohy		
Zadanie	Nápovedy	Spravná odpoveď
Ak A, B sú náhodné udalosti a P je pravdepodobnosť, potom platí $P(A \cup B) = P(A) + P(B) - P(A \cap B)$	This is hint 1, This is hint 2, This is hint 3	Pravda
Vieme že prístupový kód sa skladá, z piatich rôznych znakov z množiny { 1, 2, 3, 4, 5, 6}. Aká je pravdepodobnosť, že na prvý krát uhádneme prístupový kód? Výsledok vyjadrite na 4 desatinné miesta.	This is hint 1, This is hint 2, This is hint 3	0,0013888
-	Pravdepodobnosť úspechu je 0.94., Počet pokusov je 5., Počet úspechov je 4.	0.234
Náhodná premenná $X \sim \text{Bi}(n; p)$ a vieme, že $E(X) = 9$ a $D(X) = 6,3$. Vypocitajte disperziu n.p. $Y = 2X - 1$.	This is hint 1, This is hint 2, This is hint 3	25,2
Ak sú náhodné udalosti A, B, C je pravdepodobnosť, potom platí $P(A \cup B \cup C) = P(A) + P(B) + P(C) - P(A \cap B) - P(A \cap C) - P(B \cap C) + P(A \cap B \cap C)$	This is hint 1, This is hint 2, This is hint 3	Pravda
V krabici s 20-timi fixkami sa nachádzajú 3 vadné. Vyberieme S. Náhodná premenná X predstavuje počet vadných fixiek. Ako sa rozdelenie pravdepodobnosti náhodnej premennej X? a niečo navyše	This is hint 1, This is hint 2, This is hint 3	Hypergeometrické

Obr. 15: Ukážka vygenerovaného PDF súboru so sadou úloh na precvičovanie.

4 Implementácia webovej aplikácie

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**[34], 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 globálne⁶ 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 16 je znázornený výstup z tohto príkazu v termináli⁷.



```
Angular CLI
Angular CLI: 18.2.12
Node: 20.17.0
Package Manager: npm 10.9.0
OS: win32 x64

Angular: 18.2.13
... animations, common, compiler, compiler-cli, core, forms
... platform-browser, platform-browser-dynamic, router

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. 16: Výstup príkazu `ng version` v termináli.

Nový Angular projekt môžeme vytvoriť pomocou príkazu:

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

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

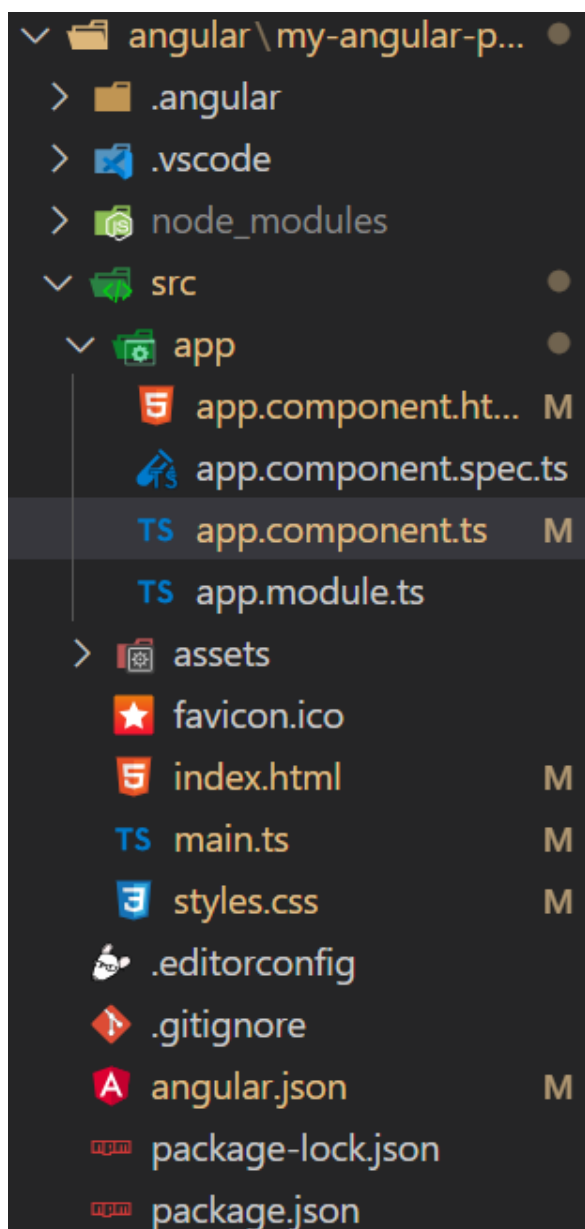
`ng new angular-app`

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:

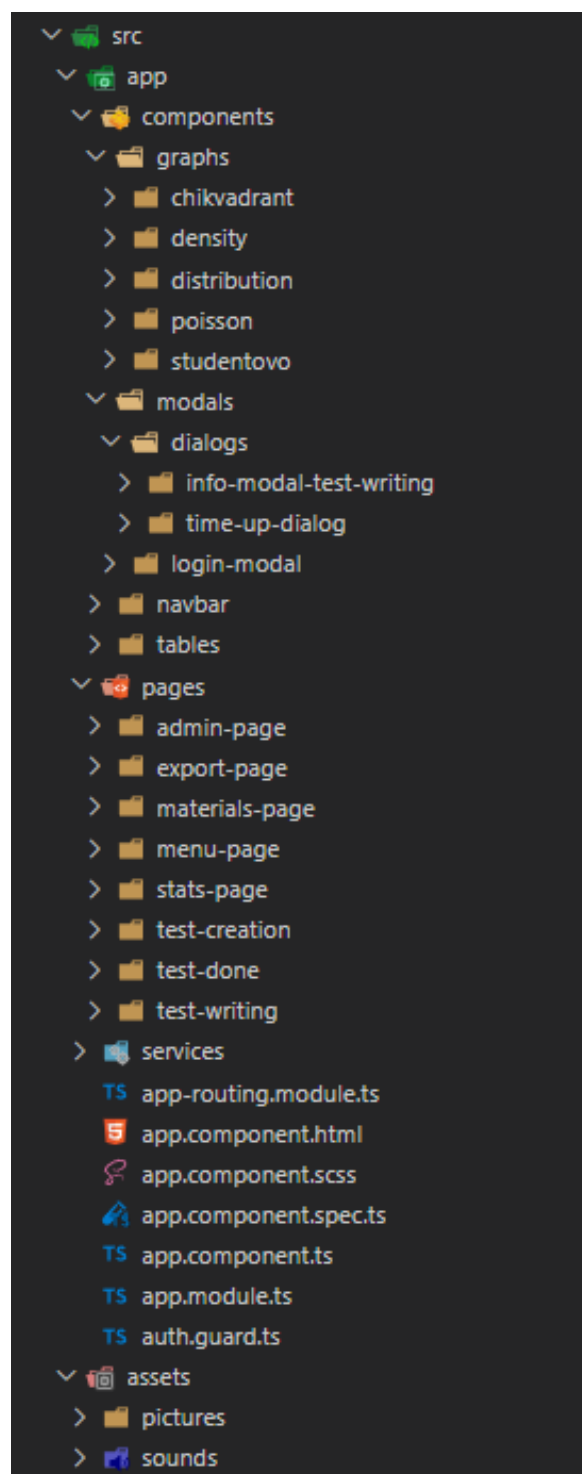
`ng serve`

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 17 je znázornená základná adresárová štruktúra Angular projektu po vytvorení, pričom na obrázku 18 je zobrazená štruktúra nášho projektu.



Obr. 17: Základná štruktúra Angular projektu.



Obr. 18: Naša štruktúra projektu.

4.2.1 Implementácia grafickej časti

Pri implementácii grafického rozhrania webovej aplikácie sme kládli dôraz na jednotný vzhľad, prehľadnosť a používateľskú prívetivosť. Aby sme zabezpečili konzistentnosť vzhľadu naprieč celou aplikáciou, rozhodli sme sa použiť knižnicu **Angular Material**. Táto knižnica poskytuje súbor preddefinovaných UI komponentov a interaktívnych prvkov v súlade s dizajnovým jazykom Material Design, čím výrazne zjednodušuje vývoj a zároveň zachováva estetickú konzistenciu.

Integrácia knižnice Angular Material do aplikácie prebiehala jednoduchým príkazom:

```
ng add @angular/material
```

Ako doplnkovú technológiu sme použili **node-sass**, čo je nástroj umožňujúci kompiláciu súborov SCSS do štandardných CSS súborov. Tento nástroj podporuje pokročilé CSS techniky, ako napríklad vnorenie pravidiel, používanie premenných a mixinov⁸, čím je zabezpečené efektívnejšie spravovanie štýlov celej aplikácie.

Zvukové prvky, ktoré dopĺňajú herné mechanizmy gamifikácie, boli integrované prostredníctvom platformy **Freesound.org**[35], ktorá poskytuje verejne dostupné zvukové efekty.

4.2.2 Vizualizácia matematických výrazov pomocou MathJax

Pre efektívne a presné zobrazovanie matematických zápisov, vzorcov a rovníc sme v aplikácii použili JavaScriptovú knižnicu **MathJax**. Táto knižnica umožňuje jednoduché vykreslenie matematiky napísanej v LaTeX formáte priamo v HTML elementoch pomocou špeciálnej property:

```
<p [mathjax]="$$content$$"></p>
```

Takto je možné dynamicky zobrazovať matematický obsah bez potreby ďalších manuálnych zásahov do DOM stromu, pričom MathJax automaticky preloží LaTeX zápis do vizuálne čitateľnej matematickej podoby priamo v prehliadači používateľa.

⁸Mixiny sú opakovateľné skupiny štýlov v SCSS, ktoré zjednodušujú kód.

Poissonovo rozdelenie:

- **Poissonova veta:** Pre binomické rozdelenie, kde $n \rightarrow \infty$ a $p \rightarrow 0$, platí:
$$\lim_{n \rightarrow \infty} P(X_n = k) = \frac{\lambda^k e^{-\lambda}}{k!}, \quad \text{pre } k = 0, 1, 2, \dots$$
- **Distribučná funkcia:** $P(X \leq n) = \sum_{k=0}^n \frac{\lambda^k e^{-\lambda}}{k!}$.
- **Očakávaná hodnota a rozptyl:** $E(X) = D(X) = \lambda$.

Obr. 19: Ukážka zobrazenia matematického výrazu pomocou MathJax.

4.2.3 Zadávanie matematických výrazov s MathQuill

Administrátorská sekcia aplikácie vyžadovala intuitívny a používateľsky jednoduchý spôsob zadávania matematických výrazov a vzorcov. Pre túto potrebu sme zvolili knižnicu **MathQuill**, ktorá umožňuje používateľom (administrátorom) zapisovať matematiku v reálnom čase pomocou LaTeX syntaxe, pričom dochádza k okamžitému vizuálnemu prekladu tejto syntaxe do štandardného matematického zápisu.

Knižnica MathQuill je závislá na knižnici **jQuery**, ktorá zabezpečuje asynchrónnu manipuláciu s HTML a CSS prvkami stránky, ako aj jednoduchú prácu s animáciami a DOM elementmi. Spojením týchto technológií sme vytvorili plynulé a responzívne prostredie pri práci s matematickými výrazmi.

4.3 Export úloh do PDF formátu pomocou pdfmake

Jednou z funkcií exportnej stránky je generovanie PDF dokumentu, ktorý obsahuje súhrn úloh, ich správnych odpovedí a prípadných návodov. Na tento účel sme využili knižnicu **pdfmake**, ktorú sme do projektu integrovali pomocou príkazu:

```
npm install pdfmake
```

Vytvorenie PDF dokumentu prebieha zostavením tzv. `documentDefinition` objektu, ktorý definuje rozloženie, obsah a štýl dokumentu. Dynamicky generované úlohy sú následne vložené do tabuľky, kde každú riadok reprezentuje jednu úlohu.

Ukážka definície hlavičky a štýlu tabuľky:

```
const documentDefinition = {
  pageSize: 'A4',
  header: (currentPage, pageCount) => ({
    text: Generované úlohy - Strana ${currentPage} z ${pageCount},
    alignment: 'center',
    style: 'header' }),
  content: [ {
```

```

    table: { headerRows: 1,
      widths: ['auto', 'auto', ''],
      body: [ ['Zadanie', 'Nápovedy', 'Správna odpoved'],
        ...this.exercises.map(ex => [ ... ]) ]
    }
  } ],
  styles: { / ... */ } };

```

Výpis 2: Zjednodušená štruktúra PDF dokumentu

Keďže zadania úloh obsahujú matematické výrazy vo formáte LaTeX, pred ich vložením do PDF je potrebné ich vyčistiť od LaTeX príkazov. Túto úlohu rieši pomocná funkcia:

```

function removeLatexCommands(text) {
  text = text.replace(/\\text{([~]}*)}/g, '$1');
  return text.replace(/[a-zA-Z]+/g, '').trim();
}

```

Výpis 3: Odstránenie LaTeX syntaxe zo zadania úlohy

Výsledný PDF dokument je vytvorený pomocou volania služby:

```
pdfMake.createPdf(documentDefinition).download('...');
```

Na obrázku 15 je uvedený príklad takto vygenerovaného výstupu.

4.4 Vizualizácia dát pomocou ApexCharts

Na vizualizáciu dát a grafických zobrazení v rámci aplikácie sme zvolili knižnicu **ApexCharts**, ktorá poskytuje moderné, responzívne a ľahko implementovateľné grafy pre Angular aplikácie. Integrácia tejto knižnice bola realizovaná jednoduchým príkazom v termináli:

```
npm install apexcharts ng-apexcharts
```

Po úspešnej inštalácii balíkov je možné vytvoriť požadované grafy priamo v komponente aplikácie. Ako konkrétny príklad uvádzame vizualizáciu hustoty pravdepodobnosti **Studentovho rozdelenia**, ktorého implementáciu máme zahrnutú aj vo vizuálnej časti tejto práce (viď obrázok č. 12).

Výpočet jednotlivých bodov pre vykreslenie Studentovho rozdelenia realizujeme nasledovnou metódou:

```

calculateTDistribution(degreesOfFreedom: number):
{ x: number; y: number }[] {
  const gamma = (z: number): number => {
    if (z === 1) return 1; if (z === 0.5) return Math.sqrt(Math.PI);
    return (z - 1) * gamma(z - 1);
  };

  const tPDF = (t: number, df: number): number => {
    const numerator = gamma((df + 1) / 2);
    const denominator = Math.sqrt(df * Math.PI) * gamma(df / 2);
    return (numerator / denominator) * Math.pow(1 + (t * t) / df, -(df + 1) / 2);
  };

  const tData = []; const range = 4; const step = 0.1;

  for (let t = -range; t <= range; t += step) {
    const x = parseFloat(t.toFixed(1));
    const y = parseFloat(tPDF(t, degreesOfFreedom).toFixed(4));
    tData.push({ x, y });
  }

  return tData; }

```

Výpis 4: Výpočet bodov Studentovho rozdelenia

Takto pripravené dáta sú následne vizualizované pomocou komponentu `apx-chart`, ktorý implementujeme priamo do HTML šablóny Angular komponentu nasledovne:

```

<apx-chart
  [series]="tChartOptions.series || []"
  [chart]="tChartOptions.chart || { type: 'line' }"
  [xaxis]="tChartOptions.xaxis || {}"
  [yaxis]="tChartOptions.yaxis || {}"
  [markers]="tChartOptions.markers || {}"
  [annotations]="tChartOptions.annotations || {}">
</apx-chart>

```

Výpis 5: Vykreslenie grafu v šablóne pomocou ApexCharts

Týmto prístupom sme zabezpečili efektívnu a jednoduchú implementáciu grafických

vizualizácií pravdepodobnostných rozdelení v aplikácii, čo používateľom umožňuje intuitívnejšie pochopenie zložitých matematických konceptov.

Na optimalizáciu výkonu pri práci s grafmi sme implementovali oneskorené vykresľovanie (lazy-loading) pomocou Angular direktívy `*ngIf`. Komponenty s grafmi sa inicializujú až v momente, keď používateľ otvorí príslušný obsah (napr. rozbaľovací panel `mat-expansion-panel`). Tento prístup efektívne znižuje čas načítania stránky a šetrí systémové prostriedky, pretože grafy sa nevytvárajú skôr, ako je to potrebné.

4.5 Backend

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 6: Ukážka základnej konfigurácie Express.js servera

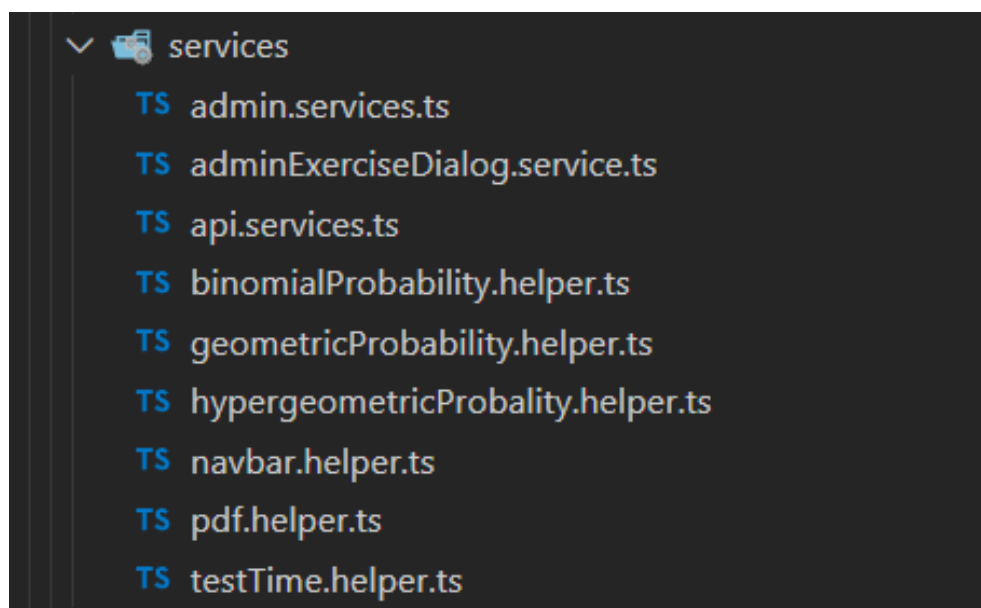
Tento server môžeme spustiť cez terminál pomocou príkazu:

```
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 20 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.



Obr. 20: 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 7: 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 sa postará o to, 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 8: 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.

4.5.1 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 9: 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.uid,
      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 10: 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 zaisťuje, 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 11: 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 12: Definícia rout v Angulari s ochranou pomocou AuthGuard

Týmto spôsobom má klient garantované, že sa na súkromné alebo administrátorské stránky dostane iba oprávnený používateľ s platným session údajom, čím zvyšujeme bezpečnosť aplikácie.

4.6 Prenos a spracovanie dát

Prenos dát medzi backendom a frontendovou časťou aplikácie zaistujú už skôr spomenuté **servisné triedy**, ktoré využívajú Angularovú službu `HttpClient` na komunikáciu s REST API. Dáta získané z databázy zvyčajne nevyžadujú ďalšie filtrovanie ani úpravy na strane klienta, pretože sú už v požadovanom formáte. Preto volania API endpointov môžeme realizovať priamo a jednoducho pomocou servisných tried, ako napríklad ukazuje nasledujúca metóda na pridanie novej úlohy:

```
createExercise(exercise: any): Observable<any> {  
  return this.http.post<any>(`${this.baseUrl}/admin/exercises`, exercise);  
}
```

Výpis 13: Volanie API na vytvorenie novej úlohy

Zaujímavosťou v rámci spracovania dát pred uložením do databázy je potreba špeciálnej úpravy textu zadania úloh. Dôvodom je obmedzenie knižnice MathJax verzie 3.1, ktorú využívame na vykresľovanie matematických výrazov. Táto verzia nepodporuje automatické zalamovanie textu ani viacriadkové zobrazenie LaTeX výrazov, preto zadania úloh pred uložením do databázy upravujeme pomocou špeciálne implementovaných helper-funkcií.

Príklad finálnej podoby zadania po úprave je nasledujúci:

```
\begin{aligned}  
& \text{V jednej krabici je 10 modrých a 5 červených balónov.} \\  
& \text{V druhej krabici je osem bielych a 12 modrých balónov.} \\  
& \text{Náhodne si zvolíme jednu krabicu a vyberieme z neho 1 balón.} \\  
& \text{Aká je pravdepodobnosť, že z nej vytiahnutý balón bude červený?}  
\end{aligned}
```

Tento formát umožňuje MathJaxu správne zobrazenie zadania na viacerých riadkoch v aplikácii.

Po odoslaní požiadavky na vytvorenie novej úlohy smerom k serveru sa táto požiadavka spracováva v definovanej API route na strane backendu (v súbore *server.js*). Tu nasleduje overenie požadovaných parametrov, spracovanie voliteľných údajov a vloženie údajov do databázy PostgreSQL pomocou parametrizovaného SQL dotazu.

```
app.post('/admin/exercises', async (req, res) => {
  try {
    const { theme_id, difficulty_level, description, points, correct_answer, hints } =
      req.body;
    if (!theme_id || !difficulty_level || !description || points === undefined || !
      correct_answer) {
      return res.status(400).json({ error: 'Missing required fields' });
    }

    const hintsValue = hints ? hints : JSON.stringify([]);

    const insertQuery = `
      INSERT INTO exercises (theme_id, difficulty_level, description, points,
        correct_answer, hints)
      VALUES ($1, $2, $3, $4, $5, $6)
      RETURNING *;
    `;
    const values = [theme_id, difficulty_level, description, points, correct_answer,
      hintsValue];

    const result = await db.query(insertQuery, values);
    res.status(201).json({ exercise: result.rows[0] });
  } catch (error) {
    console.error('Error creating new exercise:', error);
    res.status(500).json({ error: 'Internal Server Error' });
  }
});
```

Výpis 14: Serverová logika na spracovanie POST požiadavky pre vytvorenie úlohy

Ako je vidieť v ukážke 14, požiadavka na vytvorenie novej úlohy je spracovaná tak, že vstupné dáta sa najprv validujú, následne sa vytvorí SQL dotaz s využitím placeholderov (1,2, ...), čo prispieva k bezpečnosti voči SQL injection. Pole *hints* je ošetrené ako voliteľné a ak nie je prítomné, do databázy sa vloží prázdne pole v JSON formáte.

Týmto spôsobom sa zabezpečuje, že všetky nové úlohy sú správne štruktúrované a bezchybne vložené do databázy. V prípade úspechu sa späť na klienta odosiela JSON odpoveď s detailmi vytvorenej úlohy.

4.6.1 Generovanie úloh

Popri úlohách získavaných priamo z databázy aplikácia implementuje aj dynamické generovanie úloh, vďaka čomu je obsah testov viac variabilný a nepredvídateľný. Nižšie uvedená implementácia demonštruje generovanie úloh s použitím binomického rozdelenia:

```
export function binomialProbabilityRandom(): binomialExercise[] {
  const exercises: binomialExercise[] = [];

  for (let i = 0; i < 1; i++) {
    const n = Math.floor(Math.random() * 10) + 3; const k = Math.floor(Math.random() *
      (n + 1)) + 2; const p = Math.round(Math.random() * 100) / 100;
    const probability = calculateProbability(n, k, p);
    const description = generateDescription(n, k, p);
    const correct_answer = calculateAnswer(n, k, p);
    const hints = [
      'Pravdepodobnosť úspechu je ${p}.',
      'Počet pokusov je ${n}.',
      'Počet úspechov je ${k}.'
    ];

    exercises.push({ n, k, p, probability, description, correct_answer, points: 3, hints
      });
  }
  return exercises;
}
```

Výpis 15: Ukážka generovania úloh s binomickým rozdelením

Hlavné časti implementácie sú nasledovné:

- **Generovanie náhodných parametrov** — Hodnoty n , k a p sú náhodne generované.
- **Výpočet pravdepodobnosti** — Vypočítaný na základe matematickej definície binomického rozdelenia.
- **Generovanie textu zadania** — Zadanie je generované v LaTeX formáte vhodnom pre MathJax.

Úlohy, ktoré sú dynamicky generované, sú pri volaní API automaticky kombinované s úlohami získanými z databázy. Týmto spôsobom vzniká hybridná sada úloh pozostávajúca zo staticky definovaných aj náhodne generovaných položiek, ktorá sa následne zobrazí používateľovi ako jeden komplexný test.

4.7 Databáza

Po úspešnej inštalácii databázového systému PostgreSQL sa môžeme uistiť o správnosti nastavenia spustením interaktívneho príkazu `psql` v príkazovom riadku. Tento príkaz nám otvorí terminál PostgreSQL, v ktorom môžeme zadávať SQL dotazy priamo do databázy.

Prepojenie databázy s našou webovou aplikáciou zabezpečujeme prostredníctvom JavaScriptového súboru `db.js`. V tomto súbore definujeme pripojenie na databázu pomocou knižnice `pg`, ktorá poskytuje jednoduchú integráciu PostgreSQL do Node.js aplikácie.

Príklad konfigurácie pripojenia je nasledujúci:

```
const router = express.Router();
const { Pool } = require('pg');

// Konfigurácia pripojenia na PostgreSQL
const pool = new Pool({
  user: 'postgres',
  host: 'localhost', // adresa databázového servera
  database: 'postgres', // názov databázy
  password: 'postgres123', // heslo používateľa
  port: 5432, // štandardný port PostgreSQL
});

module.exports = {
  pool, // exportujeme pripojenie pre ďalšie použitie
  query: (text, params) => pool.query(text, params),
  findUserById,
  insertUser,
};
```

Výpis 16: Ukážka pripojenia na PostgreSQL databázu pomocou knižnice `pg`

V uvedenom príklade nastavujeme používateľské meno, heslo, adresu hostiteľa a číslo portu, cez ktorý sa aplikácia pripája k databáze. Objekt `pool` nám umožňuje efektívne spravovať viacero súčasných pripojení do databázy.

Tieto exportované moduly následne využívame v hlavnom súbore `server.js`, kde vykonávame SQL dotazy a získavame dáta potrebné pre beh aplikácie.

4.8 Dockerizácia aplikácie ??? napíše aj toto

Všetky predchádzajúce kroky inštalácie samostatných závislostí (Node.js, Angular CLI, PostgreSQL) sú pri dockerizovanej aplikácii zbytočné, pretože Docker abstrahuje a izoluje

pracovné prostredie každej zložky aplikácie. Vďaka tomu sa nemusíme zaoberať ručnou správou verzií či navzájom prepojených knižníc.

Naša architektúra je rozdelená do viacerých kontajnerov:

- **Frontend (Angular)** – beží v samostatnom Nginx kontajneri.
- **Backend (Node.js/Express)** – beží v samostatnom Node.js kontajneri.
- **Databáza (PostgreSQL)** – využíva oficiálny PostgreSQL obraz.

Takýto prístup (multi-container system) zabezpečuje lepšiu modularitu, škálovateľnosť a prehľadnosť infraštruktúry.

Nižšie je ukážka Dockerfile pre Angular aplikáciu:

```
FROM node:20.11.1 as builder

# Set the working directory
WORKDIR /app

# Copy package files and install dependencies
COPY package*.json ./
RUN npm install --force

# Copy the application code and build
COPY . .
RUN npm install -g @angular/cli && ng build --configuration production

# Use a smaller image for serving the Angular app
FROM nginx:alpine

# Copy the built Angular app to the nginx public directory
COPY --from=builder /app/dist/browser /usr/share/nginx/html

# Expose port 80
EXPOSE 80

# Default command
CMD ["nginx", "-g", "daemon off;"]
```

Výpis 17: Ukážka Dockerfile pre Angular aplikáciu

V tomto Dockerfile sa v prvej fáze (builder) vytvára produkčná build Angular aplikácie a v druhej fáze sa tento build kopíruje do Nginx kontajnera, ktorý ho staticky servuje.

Relevantná časť `docker-compose.yml`, ktorá definuje jednotlivé služby a ich závislosti:

```
services:
  angular-frontend:
    container_name: frontend
    build:
      context: .
      dockerfile: Dockerfile.angular
    ports:
      - "80:80"
    depends_on:
      - node-backend

  node-backend:
    container_name: backend
    build:
      context: .
      dockerfile: Dockerfile.node
    ports:
      - "3000:3000"
    depends_on:
      - postgres-db

  postgres-db:
    image: postgres:latest
    container_name: postgres-db
    restart: always
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres123
    ports:
      - "5432:5432"
    volumes:
      - ./data:/var/lib/postgresql/data
      - ./scripts/db:/docker-entrypoint-initdb.d //spustenie skriptu na inicializáciu databázy
```

Výpis 18: Ukážka `docker-compose.yml` pre multi-container systém

Celý proces zostavenia, inštalácie závislostí a spustenia všetkých kontajnerov zabezpečíme jediným príkazom:

```
docker-compose up --build
```

Tento príkaz vytvorí potrebné Docker obrazy a spustí všetky definované služby v správnom poradí.

5 Testovanie

Dôležitou súčasťou vývoja webovej aplikácie bolo používateľské testovanie, ktoré malo za cieľ získať spätnú väzbu o používateľskom rozhraní a celkovom používateľskom zážitku (UX). Počas tohto testovania bola aplikácia poskytnutá väčšiemu počtu používateľov, ktorí sa následne vyjadrili k jej funkčnosti, prehľadnosti a intuitívnosti jednotlivých prvkov.

Získaná spätná väzba bola prevažne pozitívna a potvrdila, že aplikácia dobre napĺňa svoj cieľ poskytovať intuitívne a interaktívne prostredie pre výučbu matematickej štatistiky a pravdepodobnosti. Objavili sa však aj pripomienky a návrhy, ktoré sa týkali určitých detailov používateľského prostredia a jeho ovládania. Tieto pripomienky sme dôsledne zohľadnili a implementovali sme viaceré úpravy a zlepšenia, vďaka ktorým sa zvýšila celková prehľadnosť, zrozumiteľnosť a jednoduchosť ovládania aplikácie.

Práve vďaka tejto spätnej väzbe sa nám podarilo ešte viac zlepšiť používateľskú skúsenosť a celkovú kvalitu výsledného produktu.

Záver

Cieľom tejto bakalárskej práce bolo navrhnúť a implementovať trojvrstvovú webovú aplikáciu, ktorá podporuje systematické vzdelávanie v oblasti matematickej štatistiky a pravdepodobnosti. Pri realizácii práce sme úspešne splnili všetky stanovené požiadavky, pričom sme vychádzali z odporúčanej odbornej literatúry a zároveň využili moderné technológie a vývojové postupy.

Výsledkom je plne funkčná a dostupná webová aplikácia, ktorá kombinuje interaktívne výučbové materiály, automaticky generované testy, vizualizáciu výsledkov a prvky gamifikácie. Táto kombinácia umožňuje študentom efektívnejšie si osvojovať teoretické poznatky a zároveň si ich overovať v praktických úlohách.

Súčasťou vývoja bolo aj testovanie aplikácie reálnymi používateľmi, vďaka ktorému sme získali cennú spätnú väzbu. Na základe tejto spätnej väzby sme upravili viaceré prvky používateľského rozhrania a zjednodušili niektoré interakcie, čo viedlo k zlepšeniu používateľskej skúsenosti a intuitívnosti práce s aplikáciou.

Aplikácia tak úspešne napĺňa svoj účel ako moderný e-learningový nástroj a predstavuje plnohodnotné riešenie v kontexte podpory výučby pravdepodobnosti a štatistiky.

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. *Vieme 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.
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. MÁCA, 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. HAHN, Evan. *Express in Action: Writing, building, and testing Node.js applications*. Simon a Schuster, 2016.
26. 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>.
27. *Figma*. Coderama. Dostupné tiež z: <https://coderama.com/slovník/figma>.
28. *Angular Material*. Material-UI. Dostupné tiež z: <https://material.angular.io/>.
29. DOUŠKOVÁ, Alena, KARIKOVÁ, Soňa a BLAŽÍČEK, Tomáš. Gamifikácia–špecifikum multimediálneho vzdelávania slovenských učiteľ'ov v zahraničí. *Vzdělávání dospělých 2021*. 2022, s. 110.
30. *Server*. Coderama. Dostupné tiež z: <https://coderama.com/slovník/server>.
31. *Nginx*. Webglobe. Dostupné tiež z: <https://www.webglobe.sk/poradna/co-je-nginx>.

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