

ŽILINSKÁ UNIVERZITA
V ŽILINE

Fakulta elektrotechniky a informačných technológií

FEIT Rezervačný systém

Bakalárska práca

MICHAELA MADARAŠIOVÁ

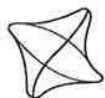
Študijný program: Multimediálne technológie

Študijný odbor: Informatika

Školiace pracovisko: Žilinská univerzita v Žiline,

Vedúci bakalárskej práce: doc. Ing. Slavomír Matúška, PhD.

Žilina 2024



Akademický rok 2023/2024

ZADANIE BAKALÁRSKEJ PRÁCE

Meno, priezvisko: **Michaela Madarašiová**
Študijný odbor: **informatika**
Študijný program: **multimediálne technológie**
Téma bakalárskej práce: **FEIT rezervačný systém**

Pokyny na vypracovanie bakalárskej práce:

1. Teoreticky spracujte možnosti návrhu dynamických web stránok.
2. Analyzujte požiadavky fakulty FEIT UNIZA pre online rezervačný systém.
3. Navrhните dizajn a definujte funkcionality navrhovaného systému.
4. Realizujte a otestujte navrhnutý systém.

Pozn. Systém bude zahŕňať responzívny dizajn, spracovanie dát s pravidlami GDPR, ukladanie a čítanie dát z databázy, automatické generovanie potvrdzovacích e-mailov, admin rozhranie a pod.

Vedúci bakalárskej práce: Matúška Slavomír, Ing. PhD., Katedra multimédií a informačno-komunikačných technológií, FEIT, ŽU v Žiline

Dátum odovzdania bakalárskej práce: 13. 05.2024

ŽILINSKÁ UNIVERZITA V ŽILINE
FAKULTA ELEKTROTECHNIKY
A INFORMAČNÝCH TECHNOLOGIÍ
KATEDRA MULTIMÉDIÍ A INFORMAČNO-
KOMUNIKAČNÝCH TECHNOLOGIÍ
Univerzitná 8215/1, 010 26 Žilina

prof. Ing. Róbert Hudec, PhD.
vedúci katedry

V Žiline dňa 31. 10. 2023

Čestné vyhlásenie

Vyhlasujem, že som zadanú bakalársku prácu vypracovala samostatne, pod odborným vedením vedúceho práce/školiťa a používala som len literatúru uvedenú v práci.

Žilina 6. mája 2024

podpis

Pod'akovanie

Týmto by som chcela poďakovať vedúcemu bakalárskej práce doc. Ing. Slavomírovi Matúškovi, PhD. za odbornú pomoc pri konzultáciách, za motiváciu a inšpiráciu pri riešení jednotlivých problémov a hlavne za ochotu a trpezlivosť.

Abstrakt

Obsahom tejto práce je teoretický rozbor technológií, ktoré sa využívajú pre vývoj moderných webových aplikácií, čo zahŕňa aj technológie využité v praktickej časti. Práca analyzuje požiadavky pre návrh nového riešenia a porovnáva ich s pôvodným riešením. Objektom práce bol návrh a implementácia online rezervačného systému s využitím MERN technológií. Táto webová aplikácia je určená pre rezerváciu špeciálnych cvičení, ktoré sú určené pre študentov stredných škôl. Tieto špeciálne cvičenia ponúka fakulta FEIT Žilinskej univerzity v Žiline. Aplikácia umožňuje spravovanie údajov, rezerváciu cvičení a schvaľovanie požiadaviek. Poskytuje jednoduché rozhranie a umožňuje efektívnejšie plánovanie. Aplikácia bola otestovaná a vďaka škálovateľnosti môže byť naďalej prispôbovaná novým požiadavkám.

Kľúčové slová: Webová aplikácia, Online rezervačný systém, MERN, Docker

Abstract

The content of this thesis consists of a theoretical analysis of the technologies used for the development of modern web applications, which includes the technologies utilized in the practical part. The thesis examines the requirements for designing the system and compares them with the original solution. The subject of the thesis was the design and implementation of an online reservation system using MERN technologies. This web application is intended for reserving special exercises designed for high school students. The special exercises are offered by the Faculty of Electrical Engineering of the University of Žilina in Žilina. The application allows data management, reservation of exercises, and approval of requests. It provides a simple interface and enables more efficient planning. The application was tested and, due to its scalability, can continue to adapt to new requirements .

Keywords: Web Application, Online Reservation System, MERN, Docker

Obsah

Úvod	1
1 Možnosti návrhu dynamických webstránok.....	2
1.1 Programovacie jazyky pre front-end vývoj	3
1.2 Programovacie jazyky pre back-end vývoj	4
1.3 Databázové systémy	5
2 Web development stack a jeho význam.....	6
2.1 JavaScript stacky a ich porovnanie.....	6
2.2 Štatistika najpoužívanějších frameworkov a technológií	8
2.3 Komparatívna analýza - porovnanie MEVN a MERN	9
3 Analýza požiadaviek fakulty FEIT	10
3.1 Súčasný stav riešenej problematiky	10
3.2 Požiadavky fakulty pre online rezervačný systém	11
3.3 Záver analýzy požiadaviek pre rezervačný systém	12
4 Systémový návrh.....	13
4.1 Návrhový vzor.....	13
4.2 Diagramy prípadov použitia	14
4.2.1 Správca cvičení	14
4.2.2 Interný učiteľ	15
4.2.3 Externý učiteľ	15
4.3 E-R model	16
5 Implementácia navrhnutého systému	18
5.1 Docker a rozdelenie aplikácie	18
5.2 Štruktúra projektu	20
5.3 API vrstva	22
5.3.1 Routing.....	22
5.3.2 Prepojenie s databázou.....	23
5.3.3 Autentifikácia a autorizácia používateľa.....	24
5.4 WEB vrstva.....	25
5.4.1 Komunikácia s back-endom	26
5.4.2 Navigácia	26
5.4.3 Vytváranie komponentov	27
5.5 Grafické rozhranie	28
5.5.1 Stránka pre prihlásenie používateľa	29
5.5.2 Domovská stránka.....	29

5.5.3	Panel správcu	29
5.5.4	Stránka spravovania používateľov.....	30
5.5.5	Stránka spravovania externých škôl	31
5.5.6	Stránka Cvičenia	31
5.5.7	Stránka Udalosti.....	32
5.5.8	Stránka Kalendár	33
5.5.9	Stránka Prihlášky	34
5.5.10	Stránka História.....	34
	Záver.....	35

Zoznam obrázkov

Obr. 1.1 – Základná architektúra webovej aplikácie.....	2
Obr. 2.1 – Prieskum StackOverflow 2023 – odpovede profesionálnych vývojárov [16].	8
Obr. 3.1 – Ukážka pôvodného riešenia.....	10
Obr. 4.1 – Návrhový vzor MVC.....	13
Obr. 4.2 – Prípady použitia pre správcu cvičení	14
Obr. 4.3 – Prípady použitia pre interného učiteľa.....	15
Obr. 4.4 – Prípady použitia pre externého učiteľa.....	15
Obr. 4.5 – E-R model	16
Obr. 5.1 – Spustenie kontajnerov v platforme Docker Desktop.....	19
Obr. 5.2 – Štruktúra priečinku <i>api</i>	20
Obr. 5.3 – Štruktúra priečinku <i>web</i>	21
Obr. 5.4 – Implementácia komunikácie s back-endom	26
Obr. 5.5 – Príklad vytvárania vnorených komponentov.....	27
Obr. 5.6 – Ukážka používateľského rozhrania	28
Obr. 5.7 – Prihlasovací formulár	29
Obr. 5.8 – Karta pre zobrazenie a úpravu údajov prihláseného používateľa.....	29
Obr. 5.9 – Grafy zobrazované v paneli správcu.....	30
Obr. 5.10 – Tabuľka spravovania používateľov	30
Obr. 5.11 – Detaily cvičenia a rezervačný formulár	31
Obr. 5.12 – Vytváranie nového cvičenia	32
Obr. 5.13 – Karta s ponukou cvičení	32
Obr. 5.14 – Karta udalosti.....	33
Obr. 5.15 – Kalendárne zobrazenie pracovného týždňa	34

Zoznam skratiek

Skratka	Anglický význam	Slovenský význam
HTML	Hypertext Markup Language	Hypertextový značkový jazyk
CSS	Cascading Style Sheets	Kaskádové štýly
URL	Uniform Resource Locator	Adresa internetovej stránky
HTTP	Hypertext Transfer Protocol	Hypertextový prenosový protokol
DOM	Document Object Model	Objektový model dokumentu
DBMS	Database Management System	Systém riadenia bázy dát
PHP	Hypertext Preprocessor	Skriptovací jazyk
SQL	Structured Query Language	Štruktúrovaný dopytovací jazyk
NoSQL	Non Structured Query Language	Neštruktúrovaný dopytovací jazyk
SPA	Single-Page Application	Jednostránková aplikácia
SEO	Search engine optimization	Optimalizácia pre vyhľadávače
JSON	JavaScript Object Notation	JavaScriptový objektový zápis
BSON	Binary JSON	Binárny JavaScriptový objektový zápis
XML	Extensible Markup Language	Rozšíriteľný značkový jazyk
JSX	JavaScript XML	JavaScriptové rozšírenie XML
MVC	Model-View-Controller	Návrhový vzor pre architektúru aplikácie
UML	Unified Modeling Language	Jednotný modelovací jazyk
API	Application Programming Interface	Rozhranie pre programovanie aplikácií
REST	Representational State Transfer	Architektúra pre distribuované hypermediálne systémy
JWT	JSON Web Tokens	JSON webové tokeny
IDE	Integrated Development Environment	Integrované vývojové prostredie

ÚVOD

Online rezervačné systémy sú nevyhnutnou súčasťou rôznych odvetví poskytujúcich služby. Tieto systémy poskytujú platformu, ktorá zefektívňuje rezervačný proces. Automatizované procesy dokážu znížiť záťaž na administratívu a v celkovom dôsledku zlepšiť produktivnosť služby. Navyše na základe cennej analýzy údajov umožňujú online rezervačné systémy predpovedať dopyt určitej služby a na základe toho upraviť interakciu alebo optimalizovať operácie.

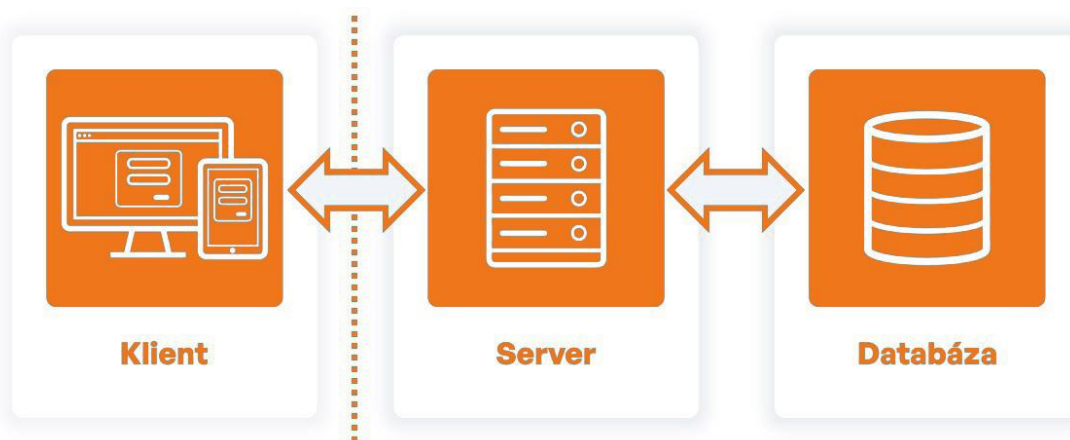
Hlavným cieľom tejto bakalárskej práce je vytvorenie online rezervačného systému, s využitím moderných technológií pre vývoj webových aplikácií. Tento systém má nahradiť pôvodné riešenie vedené na stránke Fakulty elektrotechniky a informačných technológií Žilinskej univerzity. Dôvodom výberu témy a jej spracovania je potreba optimalizácie pôvodného riešenia na základe požiadavky fakulty.

V teoretickej časti tejto práce opisujeme základné technológie, ktoré sa pre vývoj web aplikácií využívajú. Uvádzame príklady najpopulárnejších jazykov a technológií, aby sme opísali rôzne možnosti pre návrh a implementáciu webových aplikácií. Vysvetľujeme moderné kombinácie technológií a výhody ich používania. V praktickej časti sa zameriavame na návrh samotného systému s dôrazom na splnenie požiadaviek. Opisujeme postupy implementácie navrhnutého systému a funkcionality vytvorenej aplikácie. V závere prezentujeme funkcionality zhotoveného systému.

1 MOŽNOSTI NÁVRHU DYNAMICKÝCH WEBSTRÁNOK

Od prvej web stránky, ktorú vytvoril Tim Berners-Lee v roku 1990 až po dnes pozorujeme rapídny pokrok vo vývoji webstránok. V minulosti bol vývoj webu jednoduchší, pretože stránky pozostávali len zo statických HTML jednotlivých dokumentov bez obrázkov. Dnes sa už používajú v kombinácii s rôznymi programovacími jazykmi a technológiami ako napríklad Javascript a CSS, čo nám umožňuje vytvárať dynamické webstránky. HTML štandard predstavoval jedinečný nástroj pre zobrazovanie informácií. Neskôr sa objavilo CSS, čo umožňovalo vytvoriť pre tieto informácie jedinečný dizajn. Začal sa využívať jazyk Javascript pre doplnenie interaktivity. A nakoniec sme sa dostali až k možnosti vytvorenia obsahu pomocou vopred vytvorených CSS šablón ako napríklad Bootstrap a Material UI [1].

V modernom vývoji webu vytváranie webovej aplikácie zahŕňa pochopenie dvoch vzájomne prepojených častí, front-endu a back-endu. Tieto pojmy sa obvykle považujú za odlišné oblasti, ale tá najdôležitejšia časť aplikácie spočíva práve v ich prepojení, nazývaným tiež *fullstack*. Za front-end považujeme aplikáciu na strane klienta (client-side), čo predstavuje všetko čo užívateľ vidí a s čím interaguje. Back-end alebo aplikácia na strane servera je zodpovedná za správu základných funkcií aplikácie ako je spravovanie údajov, ich ukladanie a logické operácie. Táto časť funguje na vzdialenom serveri a zahŕňa aj databázu. Front-end odosiela HTTP požiadavky na back-end, pričom back-end tieto požiadavky spracuje, vykoná potrebnú logiku a odošle späť príslušnú odpoveď vo forme údajov, stavových kódov alebo chybových správ. Táto interakcia tvorí základ fungovania webových aplikácií [3]. V tejto kapitole si predstavíme konkrétne príklady technológií, programovacích jazykov, frameworkov, databáz a databázových systémov využívaných pre front-end a pre back-end.



Obr. 1.1 – Základná architektúra webovej aplikácie

1.1 Programovacie jazyky pre front-end vývoj

Základné technológie pre front-end pozostávajú z HTML, CSS a Javascriptu. Tieto technológie sa využívajú takmer v každej webovej aplikácii [1, 3].

HTML predstavuje Hyper Text Markup Language (Hypertextový značkovací jazyk) a používa sa na vytvorenie základnej štruktúry webovej stránky. „Hypertextový“ dokument predstavuje dokument obsahujúci linky, ktoré umožňujú užívateľovi navigáciu do rôznych častí webovej stránky alebo odkazujú na iný obsah. „Markup Language“ je mechanizmus, ktorý slúži na to, aby mohli počítače medzi sebou komunikovať a riadiť ako sa text spracuje a zobrazí. Využívame pri tom tagy a atribúty. Tagy sú preddefinované textové označenia, ktoré prehliadačom definujú ako majú formátovať a zobrazovať obsah. Väčšinou ide o párové značky – úvodná značka `<tag>` a záverečná `</tag>`. Atribúty sú dodatočné informácie obsiahnuté v úvodnej značke [1, 4].

CSS označuje Cascading Style Sheets (Kaskádové štýly) a predstavuje voľne dostupný štandard, používaný na úpravu vzhľadu webových stránok a zabezpečenie responzívneho dizajnu. Pomocou CSS definujeme pravidlá, ktoré upravujú vzhľad prvkov v HTML dokumente. Každé pravidlo pozostáva zo selektora, ktorý určuje, ktoré prvky sa majú upravovať a bloku deklarácií, ktorý určuje konkrétne vlastnosti vzhľadu ako sú farby, typ písma, rozloženie, okraje a ďalšie. Príkladom môže byť nastavenie veľkosti písma pre prvok *p* (paragraf). Pravidlo je definované ako *p { font-size: 20px; }*, pričom *p* predstavuje selektor a deklarácia v zložených zátvorkách špecifikuje veľkosť písma. Kaskádový mechanizmus umožňuje, že štýlové pravidlá môžu byť prekrývané a dedené. Štýly môžu byť umiestnené priamo v HTML dokumente, ale zvyčajne sa nachádzajú v externých súboroch, ktoré sú prepojené s dokumentom. O vývoj tohto štandardu sa stará skupina vývojárov, ktorí postupujú na základe usmernení W3C (World Wide Web Consortium). Dnes je dostupný v troch verziách. V súčasnosti využívame verziu CSS 3 [1, 4].

JavaScript je skriptovací jazyk, ktorý vznikol v roku 1995. Umožňuje implementovať komplexné dynamické a interaktívne funkcie na webových stránkach ako napríklad aktualizáciu obsahu, interaktívne mapy, animovanú 2D/3D grafiku, atď. Završuje trojicu technológií, ktoré spolu vytvárajú základ front-endu. Okrem toho slúži primárne na spracovávanie vstupov používateľov a validáciu údajov. Dôležitou súčasťou je možnosť manipulácie s objektovým modelom dokumentu (DOM) a spravovanie asynchrónnych operácií. Dokáže komunikovať so serverom, načítať získané údaje a zmeniť obsah stránky na základe interakcie používateľa. JavaScript bol prijatý ako štandard ECMA-262 v roku 1997. [1, 4].

1.2 Programovacie jazyky pre back-end vývoj

Základmi pre back-end sú jazyky ako Java, Ruby, Python, PHP, .NET ale v moderných technológiách aj Javascript [3]. V tejto časti si okrem JavaScriptu podrobnejšie uvedieme aj ďalšie tri príklady jazykov, ktoré sa v back-ende využívajú.

Python je univerzálny a veľmi populárny programovací jazyk. Využíva sa na rôzne účely, vrátane strojového učenia, Internet of Things (IoT), spracovania dát, vývoja hier a ďalších. Je tiež veľmi vhodný pre back-end vývoj webu. Dôvodom je jeho rozsiahla zbierka štandardných knižníc. Tieto knižnice poskytujú mnoho vopred pripravených funkcií a nástrojov na back-endové riešenia. Vďaka tomu môžu vývojári rýchlejšie a efektívnejšie vytvárať spoľahlivé aplikácie. Výhodami vývoja v jazyku Python sú jednoduchá syntax, lepšia čitateľnosť kódu, dynamické písanie bez potreby deklarácie premenných, automatické spravovanie pamäte, integrácia s inými jazykmi a podpora multiplatformovej portability. Mnoho frameworkov pre back-end vývoj pracujú práve v tomto programovacom jazyku. Sú to napríklad *Django*, *Flask* a *Pyramid* [5, 6].

PHP (Hypertext Preprocessor) je veľmi rozšírený univerzálny skriptovací jazyk, ideálny pre vývoj webových aplikácií. PHP stránky sú stavané na základe HTML dokumentov, v ktorých je zakomponovaný kód, ktorý vykonáva určité operácie. Tento kód je obsiahnutý v špeciálnych zátvorkách (`<?php ? >`), ktoré označujú začiatok a ukončenie spracovania. Kód sa spúšťa na serveri a generuje HTML, ktoré sa potom odošle klientovi. Klient dostane výsledky tohto skriptu, ale nemá prístup k zdrojovému kódu. PHP je veľmi efektívny pri práci s databázami a ponúka množstvo vstavaných funkcií pre webové operácie. Najvyužívanejší PHP framework je *Laravel*, ale obľúbené sú aj *CodeIgniter* alebo *Symphony* [5, 7].

Java je štandardný objektovo-orientovaný programovací jazyk, ktorý vznikol v roku 1995. Tento jazyk sa využíva na vývoj Android aplikácií, desktop aplikácií, vývoj webu, umelú inteligenciu, cloud aplikácie a omnoho viac. Hlavnou výhodou využívania Javy je, že funguje na princípe „Write Once Run Anywhere“ (napíš raz a spusti všade). To znamená, že skompilovaný kód napísaný v Jave dokážeme spustiť na akejkoľvek platforme bez potreby rekompilácie. Konkrétne je kód najskôr skompilovaný do byte kódu, ktorý je strojovo nezávislý a potom tento byte kód beží na JVM (Java Virtual Machine) nezávisle od danej architektúry. Okrem toho Java podporuje multithreading, ktorý umožňuje súbežné vykonávanie dvoch alebo viacerých dátových procesov pre maximálne využitie CPU. Java využíva pre back-end vývoj webu frameworky ako *Spring*, *Struts* a *Grails* [5, 8].

1.3 Databázové systémy

V súvislosti s webovými stránkami a aplikáciami sa databáza využíva na ukladanie všetkých informácií a logických operácií ktoré používateľ vykonal. Pre ovládanie databázy využívame tkzv. systém riadenia bázy dát (Database Management System - DBMS). Tento softvér využíva spravovacie operácie pre pridanie, uloženie, získanie a vymazanie dát. Pomocou neho dokážu koncoví používatelia zobrazovať a interagovať s dátami, ktoré sa získajú z databázy. V tejto časti si uvedieme niekoľko príkladov databáz, ktoré sa v súčasnosti najčastejšie využívajú [9, 10].

MySQL je databáza, ktorú je možné využívať na front-ende aj na back-ende. Vznikla v roku 1995 a neskôr v roku 2009 ho odkúpilo Oracle. SQL (Structured Query Language) je akronymom pre štruktúrovaný dopytovací jazyk. Dáta sú ukladané vo forme tabuliek, kde každý riadok reprezentuje entitu a každý stĺpec predstavuje atribút. Celý proces ukladania a získavania dát prebieha v troch krokoch. Najprv klient definuje vzťahy medzi jednotlivými entitami a ukladá dáta do tabuliek. Následne klient požiada server o vykonanie dopytu na získanie konkrétnych dát. Nakoniec serverová aplikácia odpovedá na tieto požiadavky a poskytuje klientovi požadované dáta. MySQL server je rýchly, bezpečný, škálovateľný a jednoduchý [10, 11].

PostgreSQL (Post-gress-Q-L) je jeden z najpokročilejších univerzálnych objektovo-relačných databázových systémov. Riadi sa konceptom ACID properties - Atomicity, Consistency, Isolation and Durability (atomické hodnoty, konzistencia, izolácia a trvanlivosť). Podporuje mnoho pokročilých programovacích jazykov ako napríklad Java, Ruby, Perl atď. Dáta môžu byť ukladané vo forme textu, audia, obrázkov, atď. Je možné v ňom používať aj pokročilejšie moderné prístupy ako napríklad; komplexné dopyty, triggre a view funkcionality a okrem primárnych aj cudzie kľúče. Nevýhodou ale je, že transakcie sú v porovnaní s MySQL veľmi pomalé [10, 12].

MongoDB je open-source databázový systém vytvorený v roku 2007 spoločnosťou 10gen Software. Na rozdiel od predchádzajúcich príkladov patrí medzi NoSQL databázy, čo znamená „non-relational“ (nerelačná). Z toho vyplýva, že schéma takejto databázy je dynamická a ponúka úplne iný mechanizmus na ukladanie a získavanie dát. Dáta ukladá vo forme dokumentov a kolekcií. Formát v ktorom sa dáta ukladajú sa nazýva BSON (na základe štruktúry JSON). Základnou jednotkou MongoDB databázy sú dátové páry typu kľúč-hodnota. Výkonnosť MongoDB databázy je oveľa vyššia v porovnaní s relačnými databázami. Hlavné funkcionality MongoDB sú škálovateľnosť, indexing, master-slave replikácia, vysoká výkonnosť atď. Ale má aj svoje nevýhody. Napríklad využíva veľkú časť pamäte a dáta v BSON formáte sú obmedzené [10, 13].

2 WEB DEVELOPMENT STACK A JEHO VÝZNAM

Na základe uvedenej teórie, vid' kapitola 1, môžeme tvrdiť, že každá webová aplikácia je vytvorená kombináciou viacerých technológií. Táto kombinácia technológií sa taktiež nazýva *stack*, voľne preložené ako „zásobník“. V technickom význame je to kolekcia operačných systémov, nástrojov, jazykov, databáz, skriptovacích jazykov, serverov, frameworkov, APIs a podobne. Tieto súčasti pracujú spolu k dosiahnutiu spoločného cieľa, t.j. vývoj softvéru. Termín stack bol popularizovaný po vzniku LAMP stacku, ktorý je akronymom pre Linux, Apache, MySQL a PHP. S vývinom tvorby webu a objavenia interaktivity sa spopularizovali Single Page Applications (SPAs). SPA je štandard webovej aplikácie ktorý sa vyhýba načítavaniu nového obsahu pomocou opätovného načítania celej stránky. Namiesto toho sa od servera vyžadujú len časti, ktoré je potrebné aktualizovať. V porovnaní so starým spôsobom načítavania stránok je tento výsledok oveľa lepší a šetrnejší. Po spopularizovaní SPAs sa začalo objavovať viac front-endových frameworkov, pretože väčšina kódu sa vykonávala na strane klienta. Z toho dôvodu sa takéto front-end frameworky stali veľmi obľúbenými a zároveň, v tej dobe začali byť populárnejšie aj NoSQL databázy. Jedným z prvých open-source stackov, ktorý tieto technológie využíval bol MEAN stack. Tento stack sa stal vzorom pre posun k SPAs a využívaniu NoSQL databáz. Dodnes je jeden z najpopulárnejších stackov pre webové aplikácie [2].

2.1 JavaScript stacky a ich porovnanie

Medzi najpopulárnejšie web development stacky patria MEAN, MERN a MEVN v kombinácií označované aj ako ME(RVA)N stacky, s dôrazom na využité front-end technológie. Mnoho front-end frameworkov a knižníc sú stavané na JavaScripte, ale revolučným objavom bol Node.js, ktorý sa stal prvým úspešným nástrojom pre využívanie JavaScriptu na strane servera. Vďaka nemu môžeme dnes nazývať JavaScript plnohodnotným fullstack jazykom. MEAN, MERN a MEVN sú súčasne tvorené kombináciou štyroch hlavných technológií a líšia sa len v technológií použitej na strane klienta. Ďalšie tri technológie sú Node.js, Express a MongoDB. Tieto technológie pracujú spolu na strane servera a dopĺňajú systém pre tvorbu a fungovanie fullstack aplikácie [14, 15].

Node.js (uvádzané aj ako NodeJS) je asynchrónny runtime enviroment JavaScriptu, ktorý zabezpečuje určité procesy v serverovej časti aplikácie. Dôraz sa kladie na neblokujúce I/O operácie softvéru. To je výhodou Node.js, pretože nám to umožňuje vytvárať aplikácie bez hrozby uviaznutia (deadlockov). Ďalšou skvelou výhodou je, že

Node.js počúva na porte a spracováva požiadavky na základe udalostí zo slučky – to znamená, že je škálovateľný a dokáže súčasne spracovávať požiadavky od niekoľkých pripojených klientov. **Express.js** (uvádzané aj ako Express) je Node.js framework navrhnutý tak, aby zjednodušil proces vytvárania aplikácií a APIs pomocou Node.js. Umožňuje výrazne zjednodušiť určité špecifické procesy ako napríklad routing. Vo väčšine aplikácií, ktoré využívajú Node.js Express spravuje všetky HTTP požiadavky. Poslednou zložkou back-end technológií ME(RVA)N stacku je **MongoDB** (viď. kapitola 1.3). Túto databázu developeri využívajú radi pretože ponúka jednoduchší a flexibilnejší systém ako iné databázy [2, 14].

Ak porovnávame samotné stacky MEAN, MERN a MEVN, týka sa to v podstate troch hlavných technológií na front-ende [15]. Preto si v tejto časti ďalej uvedieme rozdiel medzi jednotlivými technológiami front-endu a ich porovnanie.

Angular je voľne dostupný front-end framework od spoločnosti Google, ktorý sa bežne využíva na vytváranie rozsiahlych webových aplikácií. Pôvodne bol vydaný ako pokračovanie AngularJS, ktorý sa objavil už v roku 2010. Rozdielom v týchto dvoch verziách je, že Angular pre vývoj využíva TypeScript, zatiaľ čo AngularJS využíva JavaScript. Angular poskytuje komplexný súbor nástrojov pre vývoj webových aplikácií vrátane routingu, spracovania formulárov a spracovania HTTP požiadaviek. Hlavnou výhodou využívania Angularu je dôvera vývojárov a dlhodobá udržateľnosť vďaka podpore od Google, detailná a prehľadná dokumentácia, škálovateľnosť, čitateľnosť a konzistencia kódu. Nevýhodami sú napríklad potreba používania TypeScriptu, ktorý je obľúbený iba u 30% developerov a veľkosť projektu, ktorá je značne viditeľná a nie úplne vhodná pre malé aplikácie [14, 15].

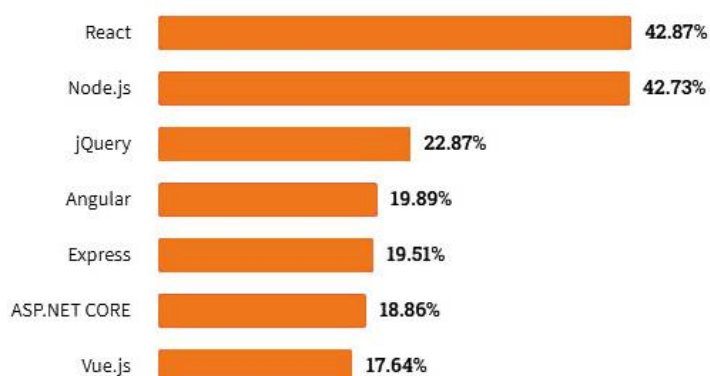
React je JavaScript knižnica, vyvinutá spoločnosťou Facebook (Meta) v roku 2013. V súčasnosti ho spolu s touto spoločnosťou spravuje aj komunita vývojárov. Jednou z najväčších výhod využívania Reactu je, že dokážeme vytvárať webové a zároveň mobilné aplikácie použitím rovnakého základu pomocou frameworku s názvom React Native. Celý vývoj užívateľského rozhrania v Reacte spočíva v práci s komponentami. Každý komponent v React aplikácii slúži ako stavebný blok. Možnosť využitia rovnakých komponentov v rámci celej aplikácie znižuje čas vytvárania a zložitosť kódu. Výkonnosť aplikácie vytvorenej v Reacte je taktiež veľkým plusom. React nezávisí od konvenčného DOM-u ale namiesto toho používa virtuálny DOM, podľa štruktúry JavaScriptu. To celkovo zvyšuje výkon a rýchlosť aplikácie. Avšak napriek tomu že React je najpopulárnejšou voľbou medzi vývojármi má aj svoje limity. Je ťažké udržať krok s neustálymi aktualizáciami a vylepšeniami, ktoré rozšírená komunita ponúka. Taktiež je potrebné používať JSX (JavaScript XML), čo je upravená verzia JavaScriptu na základe

nových funkcií ES6, čo je najnovší štandard JavaScriptu. JSX umožňuje používať HTML elementy priamo v kóde. To môže byť veľkou výhodou, ale pre niektorých to predstavuje zbytočne zložitú a menej prehľadnú syntax [2, 14, 15].

Vue je progresívny front-end framework, ktorý vyšiel v roku 2014 a bol veľmi rýchlo spopularizovaný vďaka svojej jednoduchosti. Vytvorili ho rovnakí tvorcovia, ktorí pracovali na frameworku Angular. Vue bol navrhnutý ako nástroj, založený na tých najlepších častiach Angularu a zbavení sa všetkých obmedzení, ktoré Angular obsahoval. Na rozdiel od Angularu a Reactu, nie je potrebné mať predchádzajúce znalosti o TypeScriptu alebo JavaScripte. Navyše využitie Vue.js má niekoľko výhod aj v rámci veľkosti projektu, čo má lepší vplyv na SEO webovej stránky. Vue má rozšírenú komunitu developerov, ktorí pomáhajú udržiavať a podporovať rast tejto technológie. Na druhú stranu, nevýhody zahŕňajú nedostatočný ekosystém, keďže nie všetky prehliadače a operačné systémy podporujú Vue.js aplikácie a taktiež celková nedôveryhodnosť, pretože Vue na rozdiel od Reactu a Angularu nezastrešuje žiadna veľká spoločnosť [14, 15].

2.2 Štatistika najpoužívanějších frameworkov a technológií

Z najnovšieho prieskumu od StackOverflow - Developer Survey (2023) vyplýva, že Node.js a React.js sú dve najbežnejšie webové technológie na základe odpovedí od všetkých respondentov. Profesionálni vývojári používajú oboje pomerne rovnako a začiatočníci používajú Node.js viac ako React (52% vs. 48%). jQuery a Express sú ďalšie dve obľúbené webové technológie podľa všetkých respondentov a jQuery používajú viac profesionálni vývojári ako začiatočníci (24% vs. 18%), zatiaľ čo Express využívajú viac začiatočníci ako profesionáli (25% vs. 20%). Next.js sa posunul z 11. miesta v roku 2022 na 6. v roku 2023 pravdepodobne vďaka svojej popularite u začiatočníkov [16].



Obr. 2.1 – Prieskum StackOverflow 2023 – odpovede profesionálnych vývojárov [16]

2.3 Komparatívna analýza - porovnanie MEVN a MERN

Podľa správy o vývojových zručnostiach – Developer skills report od Hacker Rank z roku 2020 30% vývojárov prešlo na Angular, zatiaľ čo 26% ostalo pri React. V správe sa tiež uvádza, že 30% programátorov sa chcelo naučiť React a 35,9% vývojárov uprednostňuje vývoj pomocou React. Z hľadiska jednoduchosti vývoja a popularity je React stále v popredí [17].

Uvedieme si niekoľko aspektov, kedy je vhodné využívať MEAN stack [14, 17]:

- Pri väčších projektoch je potrebné dodržiavať určitú architektúru, ako napríklad MVC architektúru. V tomto prípade je vhodnejšou voľbou MEAN.
- Angular podporuje HTTP požiadavky, čím zjednocuje back-end aplikácie
- MEAN stack uľahčuje integráciu knižníc tretích strán, čo je dôležité pre veľké projekty. MERN by vyžadoval konfiguráciu.

Ďalej si uvedieme niekoľko výhod MERN stacku [2, 17]:

- MERN je známy svojou jednoduchou implementáciou pre CRUD operácie.
- Vďaka Reactu vieme jednoducho implementovať interaktívne rozrania.
- Využíva fullstack JavaScript, čo eliminuje potrebu ovládať iné jazyky.
- Používa JSON formát, ktorý zjednodušuje manipuláciu s dátami.
- Node.js poskytuje vysoký výkon a efektivitu a dokáže zvládať náročné dátové procesy.
- Využíva správcu balíkov npm, ktorý ponúka prístup k množstvu balíkov a modulov
- Podporuje isomorfné (univerzálne) aplikácie, čo umožňuje spustenie rovnakého kódu na klientskom aj serverovom prostredí.

V celkovom dôsledku je MERN považovaný za efektívnejší v oblastiach jednoduchosti, škálovateľnosti, CRUD operácií a interakcie s používateľom. Je ideálny pre interaktívne aplikácie, ako aj rezervačné systémy. Vďaka Virtual DOM React poskytuje jednosmerný tok údajov a rýchle spracovanie zmien, čo zlepšuje používateľský zážitok. Na rozdiel od MERN, MEAN je lepší vo využívaní knižníc tretích strán a pri dodržiavaní MVC architektúry. Je tiež cenovo dostupnejší a je prvou voľbou pre startupy a MSP. Pre webové aplikácie v reálnom čase je MEAN určite najlepšou voľbou. Oba stacky sú však na rovnakej úrovni, pokiaľ ide o bezpečnosť. Napokon postavenie Reactu ako knižnice a nie ako frameworku poskytuje flexibilitu prispôbiť aplikácie na základe špecifických potrieb [2, 17]. Aj pre tieto dôvody budeme pri vývoji rezervačného systému využívať MERN technológie.

3 ANALÝZA POŽIADAVIEK FAKULTY FEIT

Účelom projektu FEIT Rezervačný systém je predovšetkým navrhnuť a realizovať funkčný rezervačný systém, ktorý má nahradiť pôvodné riešenie vedené na fakultnej stránke *feitcity.sk*. Pôvodné riešenie bolo vytvorené pomocou systému CMS Wordpress, avšak toto riešenie sa nezdalo byť optimálnym riešením, z hľadiska funkcionality a z hľadiska požiadaviek fakulty. Práve preto sa vedenie rozhodlo, že riešenie je potrebné zmodernizovať a optimalizovať, aby spĺňalo všetky požiadavky pre spravovanie a rezerváciu špeciálnych cvičení. Požiadavky sme sa rozhodli ujať a stala sa témou tejto bakalárskej práce.

3.1 Súčasný stav riešenej problematiky

Hlavnou nevýhodou pôvodného riešenia je, že špeciálne cvičenia uvádza ako číslovaný zoznam odkazov, pomocou ktorých sa dostaneme na túto stránku, viď obr. 3.1. Na uvedenej stránke môže užívateľ následne zobraziť základné údaje o zvolenom cvičení, poskytnúť svoje kontaktné údaje, zvoliť si vyhovujúci čas a vykonať rezerváciu. Toto riešenie je nevyhovujúce najmä z toho hľadiska, že neposkytuje širší prehľad o všetkých cvičeniach ponúkaných v rámci udalosti a taktiež neposkytuje žiadnu registráciu a prihlásenie do systému, aby mohol používateľ svoje požiadavky spravovať. Nedostatok funkcionalít pre používateľov nás donútil vytvoriť úplne nové riešenie, ktoré musí spĺňať všetky dané požiadavky pre systém v ktorom je možné poskytnúť rôzne funkcionality pre rôzne typy používateľov a lepší používateľský zážitok.

The screenshot shows a web interface for a reservation system. At the top, it displays the title '7. Biosignály ľudského tela' and details: 'Vyučujúci: Babušiak', 'Miestnosť: BD119', 'Maximálny počet študentov: 20', and 'Meranie biosignálov – EKG, EEG, srdcova frekvencia, simulátor pacienta.' A red note states 'Trvanie jedného cvičenia je 50 minút.' Below this is a calendar for April 2024, showing days from Monday to Sunday. The calendar grid has dates 1 through 30, with some dates marked 'nedostupné' (unavailable). To the right of the calendar are three main sections: 'Dátum' (Date) with a dropdown for 'Čas' (Time) and a 'Počet študentov' (Number of students) dropdown; 'Rezervácia' (Reservation) with a button 'Vyberte si dátum a čas vybraného cvičenia' (Select date and time of the chosen exercise); and 'Kontaktné informácie' (Contact information) with fields for 'Meno a priezvisko' (Name and surname), 'Názov strednej školy' (Name of the secondary school), 'Email', and 'Telefónne číslo' (Phone number). There is also a 'Doplňujúce informácie' (Additional information) field and a 'Rezervácia' button at the bottom.

Obr. 3.1 – Ukážka pôvodného riešenia

3.2 Požiadavky fakulty pre online rezervačný systém

Online rezervačný systém predstavuje softvér, ktorý nám uľahčí spravovanie rezervácií a spojenie so zákazníkmi, v našom prípade s externými učiteľmi zo stredných škôl. V zásade je to systém, ktorý má zákazníkovi umožniť zarezervovať si určitú službu, bez potreby priamej osobnej komunikácie so zamestnancom univerzity. Hlavnými výhodami využitia online rezervačného systému sú časová nenáročnosť, nepretržitá prevádzka a automatizované upozornenia. Každý rezervačný systém by mal taktiež zahŕňať možnosť prehliadania ponúk pre rezerváciu, kalendár pre prehľadné informácie na jednom mieste a automatizované emaily pre potvrdenie rezervácií a schválení [18]. Účelom tohto systému je umožniť zamestnancom univerzity jednoduché spravovanie ponúkaných cvičení a externým učiteľom prehľadnú ponuku s možnosťou rezervácie. Pomocou automatizovaných emailov budú používatelia včas informovaní o rôznych zmenách a nadchádzajúcich udalostiach.

Nazáklade nedostatkov pôvodného riešenia opísaných v predchádzajúcej časti si fakulta určila požiadavky pre nový systém, ktoré sme následne spracovali. Základným rozdielom v novom riešení je rozdelenie používateľov na rôzne typy. Toto rozdelenie umožňuje ponúknuť prispôsobené funkcionality na základe typu používateľa. Všeobecná funkcionality systému teda zahŕňa; spravovanie pridelených špeciálnych cvičení zamestnancami, prihlasovanie sa na povolené cvičenia externými učiteľmi a spravovanie všetkých cvičení a celého systému spolu s prístupom k schvaľovaniu špeciálnym užívateľom, správcom cvičení.

Okrem tejto všeobecnej funkcionality sme podľa požiadaviek fakulty definovali štyri kolekcie dát:

- A. Používatelia – Používateľov môže vytvoriť správca cvičení a každý používateľ má definovanú jednu z troch rolí. Ak má pridelenú rolu externý učiteľ, má aj príslušnosť ku škole.
- B. Externé školy – Externé školy vytvára a spravuje správca cvičení a pri vytváraní používateľa mu môže priradiť príslušnosť k danej externej škole.
- C. Cvičenia – Cvičenia vytvára správca cvičení, ktorý následne cvičenie priradí zamestnancovi alebo zamestnancom, ktorí budú toto cvičenie zastrešovať a ďalej spravovať.
- D. Udalosti – Udalosti vytvára správca cvičení. Každá udalosť bude následne obsahovať aj zoznam otvorených cvičení a prihlásenia na dané cvičenia.

Na základe známych dátových kolekcií sme podľa požiadaviek fakulty určili prípady použitia pre každý typ používateľa. Prípady použitia predstavujú najdôležitejšie funkcionality a zároveň hlavné požiadavky pre rezervačný systém. Detailnejšie si ich prejdeme v časti *Systémový návrh*, viď kapitola 4.2. Pre zhrnutie, sú to nasledovné požiadavky:

Externý učiteľ:

- Zobrazenie aktuálne otvorených cvičení
- Prihlásenie sa na otvorené cvičenie
- Zobrazenie všetkých vlastných prihlášok na cvičenia
- Zobrazenie prihlášok svojich kolegov (používatelia, ktorí majú priradenú rovnakú externú školu)
- Zobrazenie histórie, ktorých cvičení sa zúčastnili

Zamestnanec (interný učiteľ):

- Zobrazenie cvičení, ktoré im boli pridelené
- Úprava cvičení (miestnosť, popis, dĺžka, kapacita ...)
- Definovanie času a dňa, kedy sa dá na cvičenie prihlásiť
- Zobrazenie počtu prihlásených študentov na cvičenie

Správca cvičení:

- Vytváranie jednotlivých cvičení, prideľovanie cvičenia zamestnancom, ktorí budú cvičenia spravovať
- Schvaľovanie prihlásených používateľov
- Možnosť vytvoriť účet pre zamestnanca/externého učiteľa
- Vytváranie záznamov externých škôl
- Vytváranie špeciálnych udalostí
- Pridávanie cvičení pre udalosti
- Schvaľovanie prihlásení na cvičenia

3.3 Záver analýzy požiadaviek pre rezervačný systém

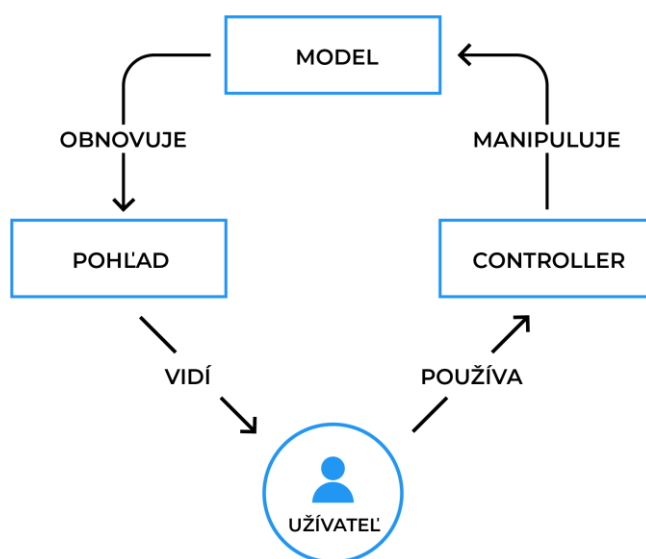
Po zhodnotení pôvodného riešenia danej problematiky sme uviedli nedostatky a možnosti ako by sa tento systém dal vylepšiť. Na základe týchto nedostatkov sme vytvorili nové požiadavky pre rezervačný systém, ktorý fakulta požaduje. Taktiež sme definovali štyri kolekcie dát, čo bude nevyhnutnou súčasťou pri návrhu databázových schém. Definovali sme všeobecnú funkcionality a požiadavky pre prípady použitia vzhľadom na rôzne typy používateľov. V ďalšej časti si tieto funkcionality bližšie uvedieme pri celkovom návrhu aplikácie.

4 SYSTÉMOVÝ NÁVRH

Pri návrhu sme sa v rámci aplikácie na strane klienta zameriavali na uplatnenie všetkých požiadaviek fakulty pre užívateľov, aby sme dosiahli optimálny užívateľský zážitok. To zahŕňalo aj vytvorenie požadovaných stránok v hlavnom menu vzhľadom na typ užívateľa. Na strane servera sme sa zameriavali na vytvorenie správneho databázového modelu pre ukladanie požadovaných dát do databázy. V tejto časti sa budeme venovať návrhu aplikácie, vrátane návrhového vzoru, prípadov použitia a konceptuálneho modelu databázy.

4.1 Návrhový vzor

Model-View-Controller (MVC) je návrhový vzor, ktorý rozdeľuje aplikáciu na tri rozličné komponenty: Model, View (pohľad) a Controller (kontrolér). Táto štruktúra nám umožňuje spravovať komplexnú aplikáciu v izolovaných častiach, pričom každá časť zodpovedá určitým úlohám. Model spravuje dáta a logiku aplikácie, vrátane interakcie s databázou. Komponent View zodpovedá za interpretáciu používateľského rozhrania na základe dát sprostredkovaných modelom pomocou kontroléra. Kontrolér je niečo ako sprostredkovateľ, ktorý celý tento proces riadi. Spracováva prichádzajúce požiadavky, logiku aplikácie v spojení s modelom a posiela odpovede s údajmi do pohľadov. Využitie tohto návrhového vzoru je výhodou, pretože rozdelenie na jednotlivé časti uľahčuje vývojový proces a umožňuje nám pracovať na rôznych častiach aplikácie samostatne. MERN tento návrhový vzor priamo nevyužíva, ale je možné ho integrovať nasledovným spôsobom: React môže slúžiť ako pohľad, Node.js a Express.js fungujú ako kontrolér a MongoDB predstavuje model [19].



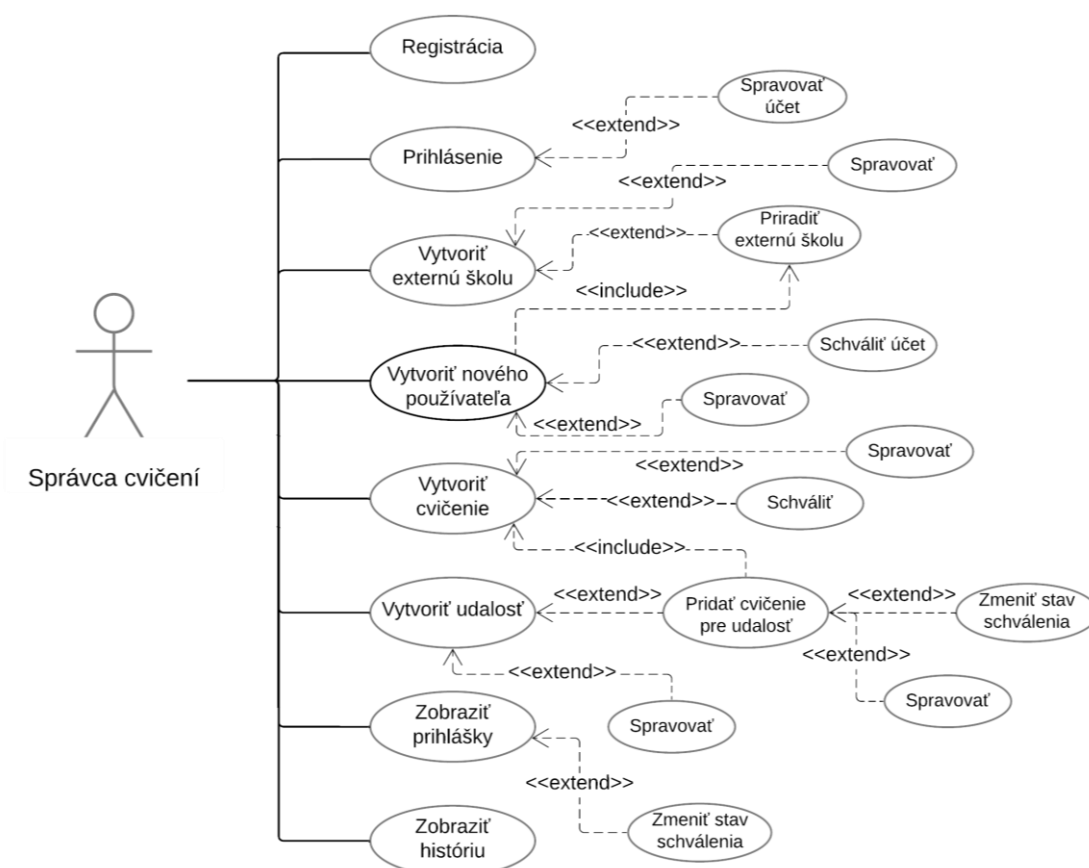
Obr. 4.1 – Návrhový vzor MVC

4.2 Diagramy prípadov použitia

Diagram prípadu použitia UML je vizuálna reprezentácia vzťahov medzi hercami (používatelia) a systémom. V oblasti softvérového vývoja sa používa ako primárna forma zobrazenia požiadaviek pre navrhovaný softvér. Diagram nám pomáha navrhnuť systém z pohľadu koncového používateľa a znázorniť, ako sa bude náš systém využívať [20]. Z hľadiska požiadaviek na systém máme tri typy koncových používateľov: správca cvičení, interný učiteľ (zamestnanec) a externý učiteľ. Pomocou diagramov si následne predstavíme prípady použitia pre každého z týchto používateľov.

4.2.1 Správca cvičení

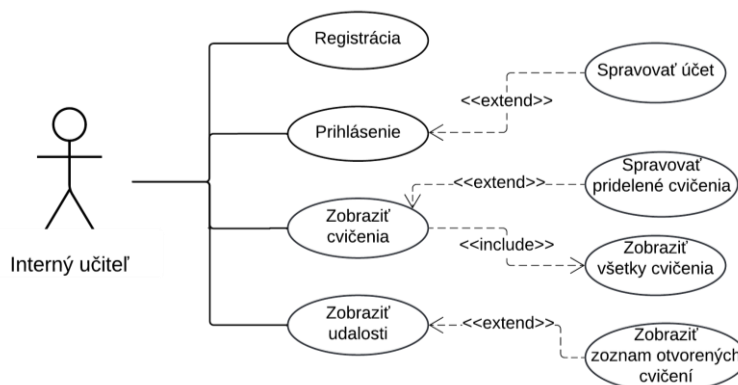
Na základe znázorneného diagramu si môžeme všimnúť, že používateľ s rolou *správca cvičení* má najviac prípadov použitia a dokáže ovplyvňovať celý systém. Má vytvorený účet rovnako ako ostatní používatelia a svoje prihlasovacie údaje môže v aplikácii spravovať. Po prihlásení má tento používateľ prístup ku všetkým stránkam a zobrazeniu dát. Jeho úlohou je ale spravovanie všetkých používateľov a externých škôl, vytváranie cvičení a otvorených cvičení v rámci udalosti a taktiež schvaľovanie vytvorených cvičení a prihlášok.



Obr. 4.2 – Prípady použitia pre správcu cvičení

4.2.2 Interný učiteľ

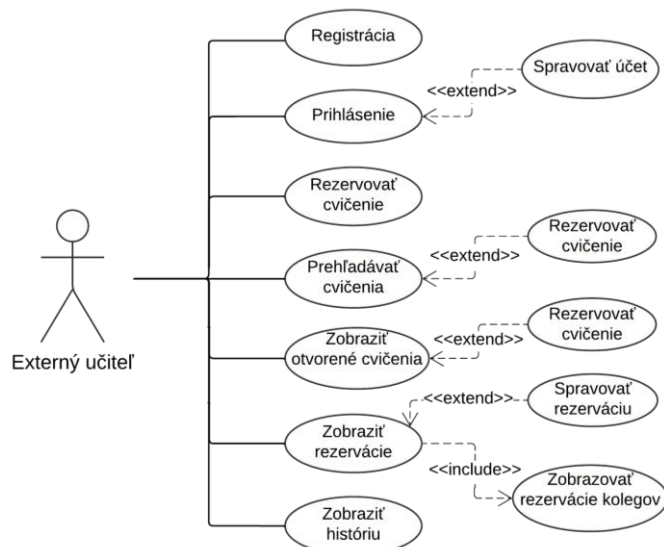
Interný učiteľ má na rozdiel od správcu obmedzený počet prípadov použitia. Jeho používateľská skúsenosť spočíva hlavne v spravovaní údajov cvičení, ktoré mu správca prideliť. To znamená, že tento používateľ bude využívať systém hlavne v čase pred samotným začiatkom udalosti. Keď sa všetky cvičenia schvália a udalosť bude v priebehu, už nebude môcť údaje ovplyvniť, avšak stále môže používať systém upozornení a sledovať nadchádzajúce udalosti.



Obr. 4.3 – Prípady použitia pre interného učiteľa

4.2.3 Externý učiteľ

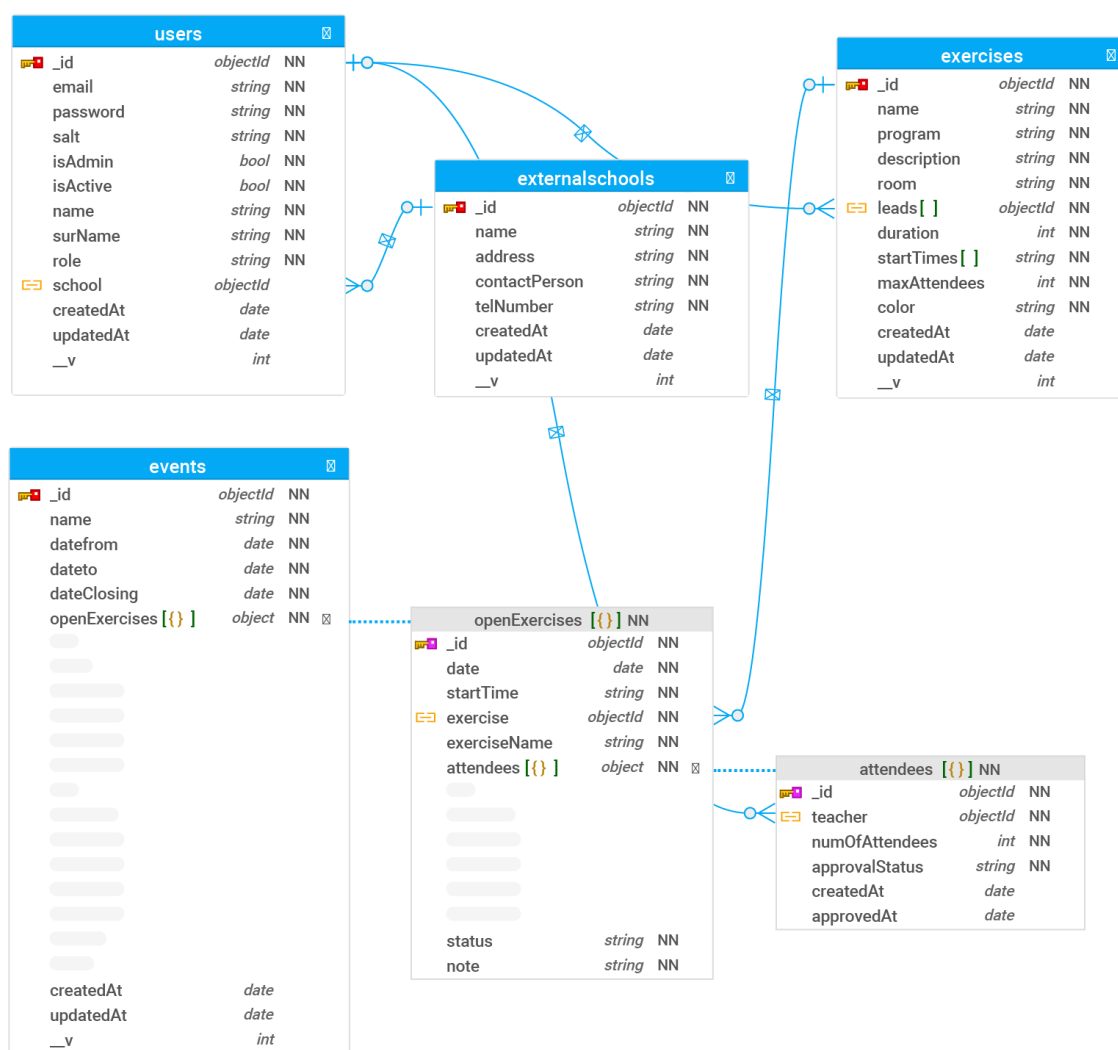
Ako hlavný koncový používateľ je považovaný externý učiteľ. Tento typ používateľa vykonáva rezervácie v našom systéme. Možnosť rezervácie sa ponúka pri prehľadávaní detailov cvičení, pri zobrazení otvorených cvičení alebo ihneď v hlavnom menu. Používateľ si potom zoznam svojich rezervácií vie zobraziť a pokiaľ nie je rezervácia schválená alebo zamietnutá, má možnosť úpravy alebo zrušenia rezervácie.



Obr. 4.4 – Prípady použitia pre externého učiteľa

4.3 E-R model

Entitno-relačný model je konceptuálny model, ktorý slúži ako vizuálny nástroj na navrhovanie a pochopenie štruktúry databáz. Využíva stavebné prvky ako entity predstavujúce objekty reálneho sveta, atribúty predstavujúce ich vlastnosti a vzťahy ako väzby medzi entitami [21]. Databáza je základným prvkom našej a zároveň každej webovej aplikácie. Na základe požiadaviek bolo potrebné vytvoriť správny model, ktorý by sa dal uplatniť aj v aplikácií vytvorenej pomocou MERN technológií. Vytvorenie E-R modelu je možné, ale nie je priamo podporované MongoDB databázou, keďže dáta v MongoDB databáze sa ukladajú vo forme kolekcií a dokumentov (viď. kapitola 1.3). MongoDB taktiež umožňuje vytváranie hierarchických štruktúr, ktoré môžu obsahovať viacero úrovní vnútri jedného dokumentu. Vzťahy medzi entitami sú reprezentované ako referencie cez ObjectId na atribút z inej entity. Vzťahy môžu byť typu jedna k jednej (1:1), jedna k mnohým (1:N) alebo mnohí k mnohým (N:M) [21]. Na základe znázorneného modelu môžeme povedať, že všetky vzťahy naprieč entitami sú typu 1:N.



Obr. 4.5 – E-R model

Na základe požiadaviek v časti 3.2 sme navrhli štyri kolekcie dát. Tieto kolekcie sú súčasťou nášho modelu a zahŕňajú nasledujúce entity:

- A. Entita Používatelia (users) obsahuje základné atribúty každého užívateľa v našom systéme. Pre účely zabezpečenia zahŕňa atribúty ako *email*, *password* a *salt* a taktiež označenia statusu *isAdmin* a *isActive*. Každý používateľ musí mať zadané meno, priezvisko a jeho rolu v systéme. Ak je daný používateľ externý učiteľ, má priradenú externú školu, v opačnom prípade bude mať atribút *school* hodnotu *null*. Používatelia s touto hodnotou sa automaticky radia ako zamestnanci, ale vzťahuje sa to aj na správcu cvičení. Vzťahy medzi entitou Používatelia a entitami Udalosti a Cvičenia sú taktiež závislé od role používateľa. Používatelia s rolou *zamestnanec* môžu byť asociovaní s viacerými cvičeniami vo vzťahu 1:N. Externí učitelia, ktorí vykonávajú rezerváciu sú taktiež vo vzťahu 1:N s podmnožinou *attendees* v rámci entity Udalosti.
- B. Entita Externé školy (externalSchools) predstavuje externé školy priradené externým učiteľom. Každá škola obsahuje atribúty ako názov školy, adresa, kontaktná osoba a telefónne číslo. Vzťah medzi externými školami a používateľmi môžeme opísať ako vzťah jedna k mnohým (1:N), keďže každá škola môže byť pridelená viacerým učiteľom ale každý učiteľ môže mať pridelenú len jednu školu.
- C. Entita Cvičenia (exercises) obsahuje vlastnosti, ktoré špecifikujú detaily každého cvičenia. Atribút *leads* je pole identifikačných objektov vo vzťahu k entite Používatelia. Každý z priradených používateľov v tomto poli zastrešuje dané cvičenie a má možnosť ho ďalej spravovať. Môže spravovať ďalšie dôležité atribúty ako dĺžka cvičenia, časy začiatku cvičenia a maximálna kapacita. Vo vzťahu k entite Udalosti sa tieto vlastnosti potom využijú v rámci vytvárania otvorených cvičení.
- D. Entita Udalosti (events) je jedinou entitou obsahujúcou vnorené dokumenty v našom databázovom systéme. Udalosť je vytvorená s definovanými vlastnosťami ako názov, dátum začiatku a konca udalosti a dátum uzávierky podávania prihlášok. Každá udalosť bude najskôr obsahovať prázdne pole pre budúce otvorené cvičenia a každé otvorené cvičenie zahŕňa prázdne pole pre účastníkov. Vnorený dokument otvorené cvičenia (*openExercises*) teda obsahuje konkrétne cvičenia odkazujúce na entitu Cvičenia. Okrem toho tento vnorený dokument špecifikuje účastníkov (*attendees*), čo predstavuje ďalšie pole objektov obsahujúcich informácie o účastníkoch vrátane prihláseného učiteľa (odkazujúce *ObjectId* na entitu Používatelia), zadaného počtu študentov a stav schválenia od univerzity.

5 IMPLEMENTÁCIA NAVRHNUTÉHO SYSTÉMU

V tejto časti si postupne uvedieme postupy implementácie rezervačného systému, a technológie, ktoré na to boli použité. Postupy zahŕňajú integráciu technológií uvedených v teoretickej časti, vid'. kapitola 2, ďalej využitie preddefinovanej šablóny, vybraných knižníc a nástrojov. Podľa návrhového vzoru opísaného v časti 4.1 je systém štruktúrovaný spôsobom, ktorý nám uľahčil vývojový proces. Vývojovým prostredím pre našu aplikáciu je populárne a univerzálne IDE Visual Studio Code (VS Code). Pre urýchlenie procesu implementácie sme využili open-source šablónu Mantis. Táto šablóna ponúkala hotovú projektovú štruktúru, predkonfigurované knižnice a komponenty pre našu aplikáciu. Je zhotovená pomocou React knižnice Material-UI (MUI), ktorú sme ďalej využívali pri vytváraní vlastných komponentov, čo nám zaručilo kvalitné rozhranie, implementovanú responzivitu a jednotný vizuál pre celú aplikáciu. Využitie šablóny nám pomohlo naštartovať vývojový proces a zamerať sa na prispôbovanie aplikácie podľa našich špecifických požiadaviek. Zdrojový kód šablóny je dostupný na Githube [25].

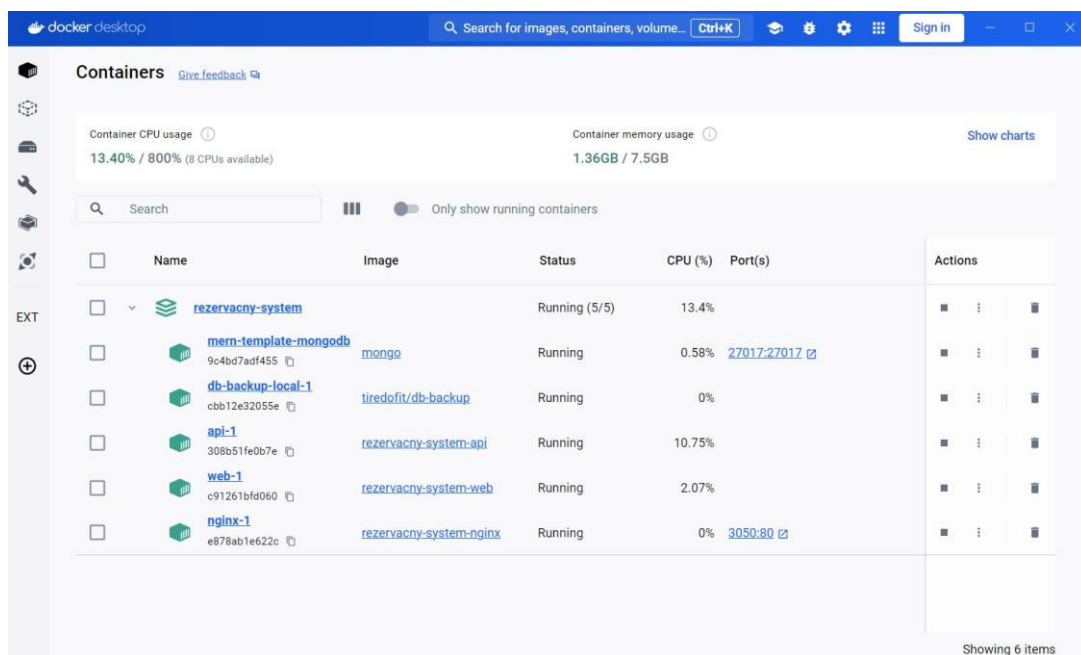
Aby sme zabezpečili, že aplikácia bude prenosná a dá sa ľahko nasadiť v rôznych prostrediach, použili sme Docker na kontajnerizáciu rôznych komponentov. Po stiahnutí šablóny, inštalácií a implementácií kontajnerizácie sme projekt nasadili na GitLab. Integrácia GitLabu a VS Code prostredia zabezpečila celkové zefektívnenie vývojového procesu. Umožnilo nám to využívať GitLab ako vzdialené úložisko a nástroj na zachovanie histórie vývoja. Pri významných zmenách v kóde sme vytvorili nové verzie a odovzdali zmeny do úložiska. Týmto postupom sme sa zameriavali na vytvorenie robustnej a škálovateľnej aplikácie, spĺňajúcej špecifikované funkčné požiadavky uvedené v časti 4.2.

5.1 Docker a rozdelenie aplikácie

Docker je široko využívaná platforma pre vývoj, prenos a spúšťanie aplikácií v nezávislých častiach, tkzv. kontajneroch. Kontajnerizácia umožňuje zapúzdrenie aplikácií spolu s ich závislosťami a konfiguráciami, čím sa zabezpečí prenositeľnosť a jednotnosť aplikácie pri spúšťaní v rôznych prostrediach. To umožňuje schopnosť behu aplikácie nezávisle od operačného systému a prostredia v ktorom bola aplikácia vytvorená. Táto vlastnosť bola pre nás obzvlášť dôležitá, pretože sme aplikáciu spúšťali na rôznych operačných systémoch (Linux Ubuntu a Windows 10). Veľkou výhodou je taktiež, že ak dôjde k zapríčineniu chyby a výpadku jednej služby, ostatné služby zostávajú neovplyvnené [22].

V časti 4.1 sme si uviedli dôvod rozdelenia aplikácie na jednotlivé časti a fungovanie týchto častí ako samostatných služieb. Každá časť aplikácie bude teda spúšťaná ako samostatný kontajner. Pre našu aplikáciu sú to nasledovné kontajnery:

- **nginx** - Nginx pôsobí ako reverzný proxy server a webový server. Prijíma prichádzajúce HTTP požiadavky a odosiela ich k príslušným back-endovým službám na základe URL.
- **web** - Web predstavuje front-endovú časť našej aplikácie. Tento kontajner slúži na spracovanie prichádzajúcich požiadaviek pre front-end, vrátane poskytovania HTML, CSS a JavaScript súborov, ako aj riadenie klientskeho smerovania.
- **api** – Tento kontajner prevádzkuje back-end aplikácie. Beží na ňom Node.js server a spracováva prichádzajúce požiadavky z front-endu. Kontajner vykonáva hlavnú logiku aplikácie a poskytuje endpointy pre získavanie, manipuláciu a ďalšie operácie cez API, pričom interaguje s databázou.
- **mern-template-mongodb** – Kontajner prevádzkuje server MongoDB, ktorý spracováva operácie na ukladanie, získavanie a správu dát. Poskytuje škálovateľné riešenie pre back-endové API.
- **db-backup-local** – Tento kontajner sme vytvorili pre účely zálohovania dát databázy. Periodicky zálohuje dáta uložené v MongoDB kontajneri, aby zabezpečil integritu dát a možnosť obnovy v prípade straty dát. Je nakonfigurovaný na pravidelné zálohovanie databázy.

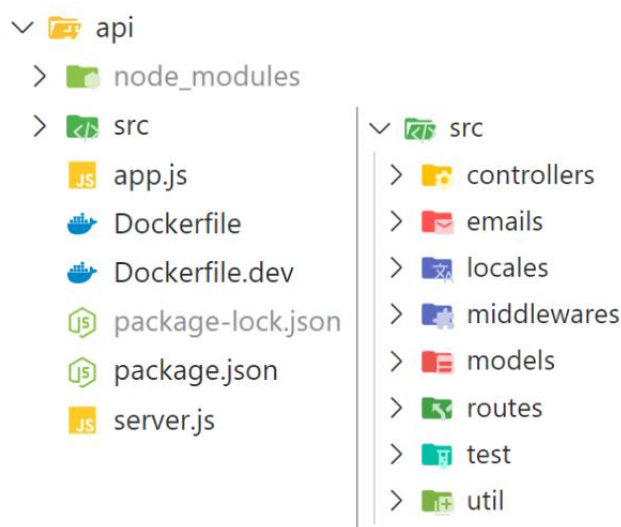


Obr. 5.1 – Spustenie kontajnerov v platforme Docker Desktop

5.2 Štruktúra projektu

Na základe rozdelenia kontajnerov a z teoretických poznatkov o základnej architektúre aplikácie z prvej kapitoly rozdeľujeme aplikáciu na jednotlivé bloky. Okrem zložiek definovaných Docker službou (*nginx*, *backup*, *volumes*) sme si pre účely rozdelenia služieb front-endu a back-endu vytvorili dve hlavné zložky s názvami *api* a *web*. Cieľom je, aby tieto dve hlavné časti fungovali ako samostatné aplikácie, pričom môžu byť vyvíjané, testované a nasadené nezávisle.

Api je jeden z hlavných priečinkov aplikácie a súčasne jedným z kontajnerov, bližšie uvedených v predchádzajúcej časti. Predstavuje kľúčový komponent systému, slúžiaci na zabezpečenie efektívnej komunikácie medzi klientom a databázou. Obsahuje implementáciu celkovej logiky systému a funguje ako centrálny uzol. Priečinok zahŕňa *node_modules*, obsahujúci všetky závislosti a knižnice potrebné pre back-end a taktiež hlavný zdrojový priečinok *src*.

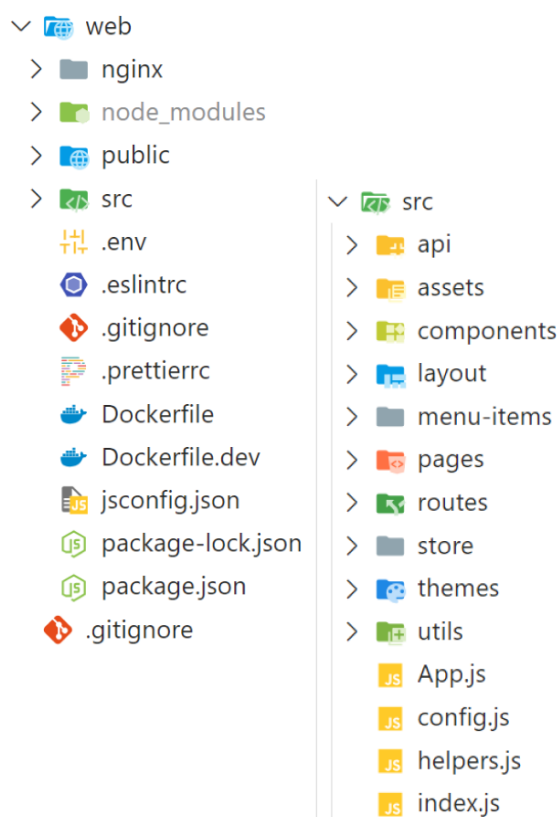


Obr. 5.2 – Štruktúra priečinku *api*

Na základe štruktúry zdrojový priečinok *src* obsahuje nasledujúce hlavné priečinky:

- **Controllers** – obsahuje súbory, ktoré definujú logiku spravovania požiadaviek pre príslušnú množinu dát
- **Emails** – obsahuje komponenty pre odosielanie e-mailov
- **Locales** – preklady textov a chybových hlásení v jazyku na základe lokalizácie
- **Middlewares** – spracovanie autentifikácie používateľov a chybových hlásení
- **Models** – definície mongoose schém pre dátové modely
- **Routes** – logicky rozdelené súbory, ktoré obsahujú koncové body
- **Util** – obsahuje univerzálne funkcie vrátane formátovania odpovedí

Aplikácia na strane klienta taktiež funguje ako samostatná aplikácia pod priečinkom *web*. Štruktúra tejto časti pozostáva zo zachovanej štruktúry projektu, ktorú obsahovala použitá šablóna *Mantis*. V zdrojovom priečinku (*src*) pod priečinkom *pages*, obsahuje jednotlivé pohľady (*views*) pre každú podstránku aplikácie. Okrem toho obsahuje základný *layout* šablóny spolu s upravenými súbormi pre hlavné menu a ďalšie súbory, ktoré využívame naprieč front-endom aplikácie.



Obr. 5.3 – Štruktúra priečinku *web*

Na základe uvedenej štruktúry pre *web* si opíšeme priečinky zdrojového priečinku *src*:

- **Api** – implementácia *axios*, ktorý riadi smerovanie požiadaviek
- **Assets** – obsahuje rôzne obrázky, štýly a skripty potrebné pre jednotlivé stránky
- **Components** – univerzálne komponenty, ktoré využívame v rozhraní
- **Layout** - základné rozloženie rozhrania, zachované zo šablóny *Mantis*
- **Menu-Items** – definovanie položiek pre hlavné menu aplikácie
- **Pages** – obsahuje jednotlivé pohľady (stránky) aplikácie, ktoré sú zostavené z viacerých komponentov
- **Routes** – definovanie smerovania ciest aplikácie pre jednotlivé podstránky
- **Themes** – súbory pre konfiguráciu tém prevzaté zo šablóny *Mantis*
- **Utils** - ďalšie užitočné nástroje a funkcie, napríklad pre zachovanie sily hesla

5.3 API vrstva

V tejto časti sa zameriame na back-end aplikácie, ktorý obsahuje logiku spracovania požiadaviek a komunikácie s databázou, vrátane autentifikácie a autorizácie používateľov. V teoretickej časti sme spomínali možnosť využitia jazyku JavaScript aj na back-ende. To platí aj pre knižnice, ktoré sme na tejto vrstve využili. Pre inštaláciu nových knižníc sme použili Node Package Manager (npm). Medzi najdôležitejšie použité knižnice na back-ende patria:

- **Express:** Framework pre Node.js, ktorý umožňuje jednoduchú tvorbu serverových aplikácií. Tento framework sme uviedli v časti 2.1.
- **Mongoose:** Nástroj pre modelovanie objektov v databáze MongoDB, ktorý sme využili na definíciu schém, viď. podkapitolu 5.3.2.
- **Dotenv:** Slúži na načítanie premenných prostredia zo súboru .env do objektu process.env.
- **Nodemon:** Nástroj na automatickú detekciu zmien a reštartovanie aplikácie.
- **Jsonwebtoken:** Knižnica pre prácu s JSON Web Tokens, ktorú sme využili na autentifikáciu a autorizáciu používateľov.
- **Node-cron:** Je Node.js knižnica, ktorá umožňuje vytvárať opakujúce sa úlohy v časových intervaloch. Túto vlastnosť sme využili pre automatizované emaily.

5.3.1 Routing

Routing je kľúčovým prvkom back-endu aplikácie. Určuje ako sa spracovávajú a posielajú požiadavky z front-endu do príslušných kontrolérov. Pre tento účel naša aplikácia používa REST API. REST API (Representational State Transfer API) predstavuje rozhranie založené na architektonickom štýle REST. API rozhrania umožňujú externým aplikáciám komunikovať s databázou bez toho, aby museli pristupovať k jej štruktúre. Koncové body aplikácie (endpoints) sú štruktúrované tak, aby pokrývali potrebné CRUD operácie (vytvorenie, načítanie, aktualizáciu a vymazanie záznamu) pre jednotlivé entity v databáze. Dáta sa v REST API vracajú vo formáte JSON alebo XML, čo je z pohľadu spracovania vyhovujúce. Každý koncový bod URL adresou a špecifikovaním HTTP metódy, ktorá určuje typ požiadavky a jej spracovanie [23, 24]. Tieto HTTP metódy zahŕňajú:

- GET: načítanie záznamov alebo údajov z databázy
- POST: vytvorenie nového záznamu v databáze
- PUT: úprava existujúceho záznamu alebo vytvorenie záznamu na základe ID
- DELETE: vymazanie záznamu špecifikovaného pomocou ID

Na základe uvedených metód sme v súboroch zložky *controllers* definovali metódy, slúžiace na spracovanie požiadaviek. Pomocou týchto požiadaviek vykonávame dopyty na koncové body, ktoré definujeme v zložke *routes* s rozdelením podľa príslušných kolekcí dát. Pri spracovaní požiadavky sa potom vyvoláva príslušný kontrolér, ktorý danú akciu vykoná a pripraví odpoveď. Pre testovanie koncových bodov sa využívajú špecializované platformy. My sme využívali nástroj *Postman*, ktorý nám umožňoval odosielať požiadavky s rôznymi parametrami a overiť odpoveď servera. Tým sme zabezpečili správnosť implementácie routingu a metód kontrolérov. Ošetrovanie chýb prebieha priamo v kontroléroch pri implementácii metód. V prípade chyby aplikácia poskytne správu o stavovom kóde s prispôbenou odpoveďou o nesprávnej požiadavke alebo o chybe v spracovaní.

5.3.2 Prepojenie s databázou

Pretože tvoríme rezervačný systém, táto aplikácia vyžaduje úložisko údajov pre správu a poskytovanie informácií svojim používateľom. Pre tento účel využívame knižnicu *mongoose*, ktorá poskytuje API pre prácu s MongoDB databázou. Pomocou *mongoose* taktiež definujeme schémy, modely, dotazy a operácie nad nimi. Samotné fungovanie uchovávanía údajov MongoDB databázy je uvedené v časti 2.3.

Pre prácu s databázou sme najskôr definovali schémy, predstavujúce štruktúru údajov podľa navrhnutého databázového modelu v časti 4.3. Definované schémy obsahujú informácie o poliach, ich typoch ale aj obmedzeniach, ktoré zamedzujú vytváranie nesprávnych údajov. Následne sme na základe definovaných schém vytvorili modely (*models*). Modely predstavujú rozhranie pre interakciu s konkrétnymi entitami v databáze, vrátane vytvárania, aktualizovania, načítania a odstraňovania záznamov.

Keďže naša aplikácia obsahuje štyri rôzne *mongoose* schémy, ako príklad uvádzame vytvorenie schémy pre jednoduchší model *ExternalSchool*:

```
const mongoose = require("mongoose");

const ExternalSchoolSchema = new mongoose.Schema(
  {
    name: { type: String, required: true },
    address: { type: String, required: true },
    contactPerson: { type: String },
    telNumber: { type: String },
  },
  { timestamps: true }
);

module.exports = mongoose.model("ExternalSchool", ExternalSchoolSchema);
```

Obr. 5.1 – Príklad definície mongoose schémy a modelu

5.3.3 Autentifikácia a autorizácia používateľa

Autentifikáciu a autorizáciu používateľov aplikácie sme zabezpečili pomocou JSON Web Tokens (JWT). Keď sa používateľ prihlási do systému, server mu prideli JWT token, ktorý obsahuje informácie o jeho identite a oprávneniach. Tento token sa odovzdáva v hlavičke požiadavky pri každej následnej požiadavke od používateľa na chránené koncové body. Vytvorili sme middleware, ktorý zabezpečuje kontrolu platnosti a oprávnenia JWT tokenu pre každú prichádzajúcu požiadavku.

```
const verifyToken = async (req, res, next) => {
  if (req.url.startsWith("/public")) {
    return next();
  }
  const token = req.headers["x-access-token"];
  if (!token) {
    return res.status(403).send();
  }
  try {
    const decoded = jwt.verify(token, process.env.TOKEN_KEY);
    req.user = decoded;
    if (req.url.startsWith("/admin")) {
      if (!decoded.isAdmin) {
        return res.status(403).send();
      }
    }
    next();
  } catch (err) {
    throw Error(req.t("messages.invalid_token"), 401);
  }
};
```

Obr. 5.1 – Implementácia autentifikácie v súbore auth.js

Ak požiadavka smeruje na verejný koncový bod, middleware ju jednoducho preskočí, pretože nevyžaduje autentifikáciu. V inom prípade middleware overí platnosť tokenu obsiahnutého v hlavičke požiadavky pomocou šifrovacieho kľúča uloženého v premenných prostredia. Ak je token platný, dekoduje sa a informácie o používateľovi sa pripoja k objektu požiadavky. Ak je požiadavka smerovaná na koncové body určené len pre správcu, middleware overuje, či má používateľ administrátorské oprávnenie. Ak používateľ nemá dostatočné oprávnenie, požiadavka je zamietnutá s príslušným stavovým kódom. V prípade, že token je neplatný alebo neexistujúci, alebo ak požiadavka nespadá pod oprávnenie používateľa, middleware vracia chybový stavový kód alebo chybovú správu. Týmto spôsobom zabezpečujeme, aby len oprávnení používatelia mohli vykonávať akcie s prístupom k záznamom.

5.4 WEB vrstva

Front-end našej webovej aplikácie bol implementovaný pomocou využitia rôznych UI knižníc, modulov a preddefinovaných komponentov, ktoré sme si nainštalovali. Nové komponenty sme pridávali v rámci spomínanej šablóny, ktorá nám poskytla už hotový layout spolu s implementovaným *drawer* komponentom a hlavným menu. Postupne sme pridávali ďalšie podstránky a definovali nové cesty.

Medzi hlavné knižnice a moduly, ktoré sme nainštalovali a použili, patria:

- **react-redux:** Knižnica, ktorá poskytuje podporu pre využitie stavu aplikácie pomocou Redux v kombinácii s Reactom. Poskytuje prepojenie stavu (state) s komponentami.
- **react-router-dom:** Knižnica na správu routovania v rámci aplikácie. Poskytuje funkcie pre navigáciu medzi stránkami a spracovávanie URL ciest.
- **axios:** Knižnica pre správu HTTP požiadaviek v JavaScripte, ktorú sme použili na komunikáciu s back-endom. Táto knižnica poskytuje jednoduchý spôsob prístupu k API pomocou asynchrónnych požiadaviek, čo umožňuje dynamické spracovanie dát. Je univerzálna a môže bežať v prehliadači aj v Node.js s rovnakým kódom.
- **@mui/material:** MUI je knižnica komponentov pre React založená na princípoch Material Design. Poskytuje bohatú sadu komponentov pre tvorbu moderných používateľských rozhraní.
- **mui-datatables:** Komponent pre tvorbu dátových tabuliek v aplikáciách vytvorených pomocou Reactu a MUI. Túto knižnicu sme použili pre spravovacie tabuľky, ktoré pracujú s veľkým množstvom záznamov.
- **react-big-calendar:** Knižnica ktorá ponúka komponent kalendára. Použili sme ju na zobrazenie cvičení v dennom, týždennom alebo mesačnom pohľade.
- **react-color:** Knižnica, ktorá poskytuje rôzne rozhrania pre výber farby. V aplikácií používame komponent CirclePicker.
- **moment:** Knižnica na prácu s dátumami a časom. Poskytuje funkcie na manipuláciu s časovými údajmi a formátovanie dátumov. Využili sme ju pri vytváraní dátumov a porovnaní s aktuálnym časom.
- **dayjs:** Alternatíva k moment.js, ktorá poskytuje podobné funkcie pre prácu s dátumami a časom.
- **react-apexcharts:** Moderná knižnica interaktívnych grafov pomocou ktorej sme vytvorili interaktívne grafy v paneli správcu.

5.4.1 Komunikácia s back-endom

V aplikácii je komunikácia s back-endom definovaná v súbore `api.js` v zložke `api`. Využili sme spomínanú knižnicu `axios`. Kód vyzerá takto:

```
import axios from "axios";
const instance = axios.create({
  baseURL: "/api"
});
instance.interceptors.request.use(
  async (config) => {
    const token = localStorage.getItem("appToken");
    if (token) {
      config.headers["x-access-token"] = `${token}`;
    }
    return config;
  },
  (err) => {
    return Promise.reject(err);
  }
);
export default instance;
```

Obr. 5.4 – Implementácia komunikácie s back-endom

Tento kód vytvára inštanciu `Axios` s predvoleným základným URL (`baseURL`) nastaveným na `"/api"`, čo znamená, že všetky požiadavky vykonané prostredníctvom tejto inštancie budú smerované na back-end. Na manipuláciu s požiadavkami a odpoveďami sa používajú interceptory. Request interceptor zachytáva požiadavky pred tým než sú odoslané. V tomto prípade sa pokúša získať token uložený v `localStorage` pod názvom `"appToken"`. Ak token existuje, pridá ho do hlavičky požiadavky pod kľúčom `"x-access-token"`. Potom vráti konfiguráciu požiadavky s pridaným tokenom, čím zabezpečuje autorizáciu požiadaviek na serveri. Ak dôjde k chybe, interceptory chybu zachytia a predajú ju ďalšiemu spracovaniu. Týmto spôsobom zabezpečujeme jednotnú komunikáciu s back-endom.

5.4.2 Navigácia

Pri vytváraní nových podstránok sme pre každú stránku v aplikácii museli definovať cestu v súboroch v priečinku `routes` pomocou uvedenej knižnice `react-router-dom`. Tieto cesty sú zodpovedné za riadenie navigácie v rámci aplikácie a určujú, aký komponent sa má zobrazíť na základe požadovanej cesty. Na spracovanie definovaných ciest a navigovania používa knižnica efekt (hook) `useRoutes`. Každá cesta má definovanú URL a komponent, ktorý sa má zobrazíť, keď je cesta aktívna. Pre uvedenie príkladu; jedna cesta môže byť definovaná ako `/manageUsers` a komponent, ktorý sa na tejto adrese zobrazí bude `<ManageUsersScreen />`.

5.4.3 Vytváranie komponentov

Hlavným aspektom vytvárania React aplikácie je vytváranie komponentov, ktoré predstavujú samostatné jednotky z ktorých sa skladá používateľské rozhranie. V našej aplikácii sme používali hlavne vnorené komponenty. Tento princíp vytvárania komponentov nám umožnil využívať univerzálne komponenty a taktiež rozložiť si rozhranie na menšie časti a implementovať jednotlivé časti postupne. Okrem toho prispieva k lepšej čitateľnosti a údržbe kódu.

Pomocou vnorených komponentov sme riešili napríklad vytváranie dialógov. Stránky obsahujú tlačidlá a rôzne ikony, pričom po akcií kliknutia sa zobrazí dialógové okno obsahujúce formulár. V tomto prípade rozhranie stránky predstavuje komponent najvyššej vrstvy a dialógové okno, ktoré sa zobrazí po akcií kliknutia je vnoreným komponentom. Samotný formulár predstavuje ďalší vnorený komponent umiestnený v komponente dialógu. Tento postup sme opakovali pri všetkých spravovacích akciách.

Pri prenose dát medzi komponentmi sme využívali *props* (skratka pre properties). *Props* sú argumenty, ktoré sa odovzdávajú z jedného komponentu do iného. Na overenie typov a štruktúry týchto *props* sme používali *PropTypes*. Táto knižnica umožňuje definovať očakávané typy a štruktúru *props* pre komponenty. Pomáha nám to predchádzať chybám a zabezpečuje spoľahlivosť rozhrania.

Nasledujúci príklad znázorňuje vytváranie hlavného (ParentComponent) a vnoreného (ChildComponent) komponentu, vrátane prenášania vlastností (props):

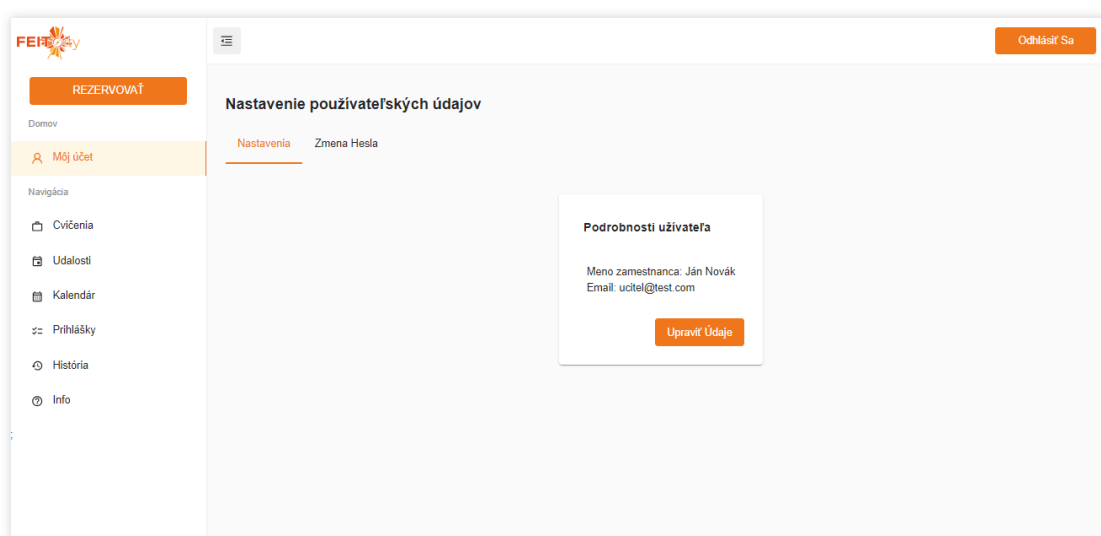
```
function ParentComponent() {  
  //predpokladáme získanie dát z databázy  
  return (  
    <div>  
      <ChildComponent data={data} />  
    </div>  
  );  
}  
  
export default ParentComponent;
```

```
const ChildComponent = ({ data }) => {  
  return (  
    <div>  
      {data}  
    </div>  
  );  
}  
  
ChildComponent.propTypes = {  
  data: PropTypes.string,  
};  
  
export default ChildComponent;
```

Obr. 5.5 – Príklad vytvárania vnorených komponentov

5.5 Grafické rozhranie

Rozhranie pre používateľov sme navrhli na základe prípadov použitia, vid' kapitola 4.2. Dbali sme na dodržaní potrebných požiadaviek fakulty a zabezpečení osobitého prístupu pre každého používateľa. Zahrnuli sme aj Info stránku, ktorá je vytvorená predbežne pre budúce požiadavky fakulty. Rozloženie a dizajn rozhrania pozostávalo zo zachovaných prvkov použitej šablóny. Pre lepší dojem sme upravili farebnosť konfiguráciou predvolenej témy. Ako základnú farbu sme použili odtieň oranžovej, ktorý zodpovedá oficiálnej rozlišovacej farbe fakulty. Okrem toho sme na mieste predvoleného loga použili oficiálne logo Feitcity. Jednotlivé pohľady aplikácie sú dostupné aj v prílohe.



Obr. 5.6 – Ukážka používateľského rozhrania

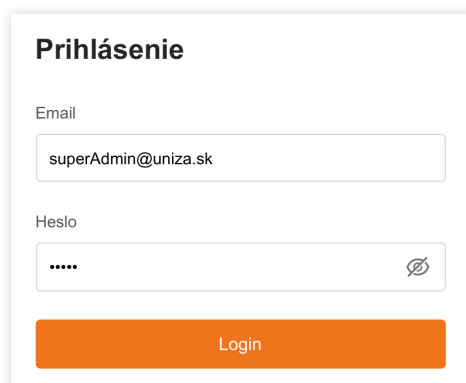
Rozhranie pozostáva z bočného panelu, kde sa nachádzajú položky menu, z vrchnej lišty a zo samotného obsahu. Pomocou menu umožňujeme jednoduchú navigáciu do všetkých podstránok aplikácie. Na uvedenom obrázku je znázornené grafické rozhranie používateľa s rolou externý učiteľ, ktorý má sprístupnenú aj možnosť rezervácie. Zobrazená je domovská stránka obsahujúca možnosť úpravy používateľských údajov.

V tejto časti si uvedieme funkcie a prípady použitia pre jednotlivé podstránky, ktoré sme implementovali. Každý typ používateľa má sprístupnený iný zoznam položiek. Na základe typu používateľa sú zoznamy položiek určené nasledovne:

1. Správca cvičení: Panel správcu, Domovská stránka, Používatelia, Externé školy, Cvičenia, Udalosti, Kalendár, Prihlášky, História, Info
2. Interný učiteľ: Domovská stránka, Cvičenia, Udalosti, Kalendár, Info
3. Externý učiteľ: Rezervačný formulár, Domovská stránka, Cvičenia, Udalosti, Kalendár, Prihlášky, História, Info

5.5.1 Stránka pre prihlásenie používateľa

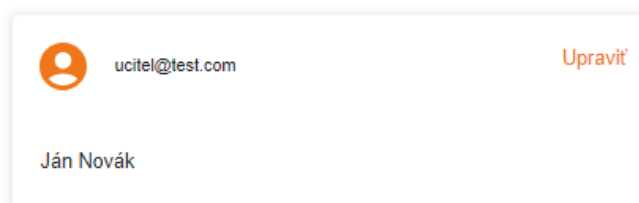
Po presmerovaní na hlavnú stránku sa najskôr pre neprihlásených používateľov zobrazí prihlasovacie okno. Štýl prihlasovacieho formulára vychádza z použitej šablóny Mantis. Používatelia sa prihlasujú pomocou emailu a vytvoreného hesla. Prihlasovacie údaje im poskytne správca systému.



Obr. 5.7 – Prihlasovací formulár

5.5.2 Domovská stránka

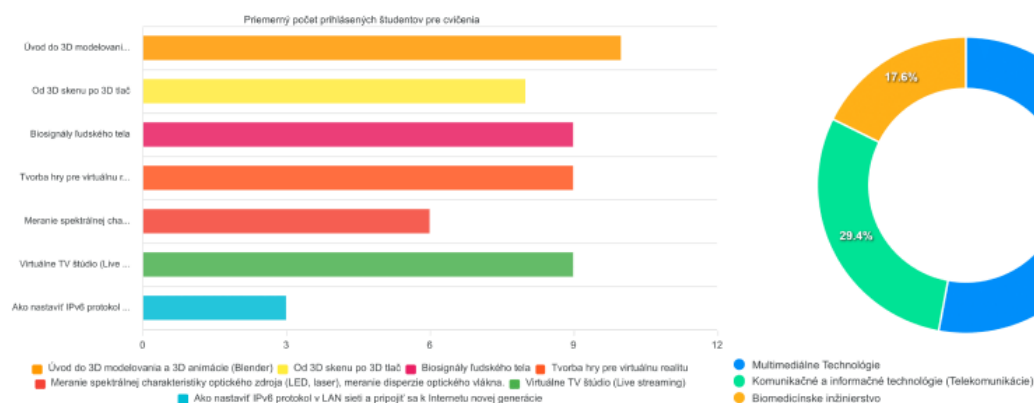
Domovská stránka sa zobrazuje rovnako pre všetkých používateľov. Na tejto stránke majú používatelia po kliknutí na tlačidlo upraviť možnosť úpravy svojich osobných údajov, vrátane zmeny hesla, ktoré im prideliť správca.



Obr. 5.8 – Karta pre zobrazenie a úpravu údajov prihláseného používateľa

5.5.3 Panel správcu

Panel správcu je dôležitou súčasťou aplikácie, ktorá zahŕňa spravovanie systému a dát. Tento panel poskytuje základný prehľad o stave poskytovanej služby a štatistiky niektorých dát. V našom prípade panel správcu zobrazuje aktuálny počet registrovaných používateľov, celkový počet otvorených cvičení, aktuálny počet rezervácií schválených správcou a celkový počet rezervácií. Okrem toho zobrazuje tabuľku o najaktuálnejších rezerváciách so základnými údajmi. Pomocou rôznych grafov má správca prehľad o štatistikách, ako počet otvorených cvičení na každý odbor v percentách, alebo porovnanie priemerného zaplnenia kapacity pre každé cvičenie.



Obr. 5.9 – Grafy zobrazované v paneli správcu

5.5.4 Stránka spravovania používateľov

Prístup k spravovaniu používateľov má len používateľ s rolou správca. Obsah stránky je tabuľka vytvorená pomocou spomínanej knižnice *mui-datatables*. Táto spravovacia tabuľka umožňuje zmeniť požadovaný počet riadkov na stránku a taktiež vyhľadávanie, filtrovanie a zoradovanie záznamov. Dostupné sú aj ďalšie možnosti ako tlač a stiahnutie záznamov. Okrem toho sa správcovi zobrazuje tlačidlo *Pridať používateľa*, pomocou ktorého vytvára nový účet pre používateľa a príslušné dáta sa potom zobrazia v tabuľke. Po vytvorení záznamu máobrazené ďalšie možnosti v rámci stĺpcu *Akcie*. Kliknutím na ikonu *upraviť používateľa* môže upraviť vytvorený záznam a kliknutím na ikonu *vymazať* sa zobrazí upozornenie o vymazaní všetkých záznamov spojených s používateľom. Správca môže túto správu potvrdiť, čím sa vymaže záznam alebo zrušiť a vrátiť sa späť. Týmto spôsobom fungujú aj ďalšie spravovacie tabuľky na nasledujúcich stránkach, ktoré si uvedieme.

Zoznam používateľov						
Pridať Používateľa						
🔍						🔍 📄 🖨️ 📱 ☰
Email	Meno zamestnanca	Pracovisko	Typ používateľa	Účet aktívny ↑	Administrátorský účet	Akcie
ucitel@test.com	Ján Novák	SOŠ Dubnica nV	externý učiteľ	☑️	☐	✎ 🗑️
juraj.bienik@uniza.sk	Juraj Bienik	UNIZA	interný učiteľ	☑️	☐	✎ 🗑️
jozef.dubovan@uniza.sk	Jozef Dubovan	UNIZA	interný učiteľ	☑️	☐	✎ 🗑️
roberta.hlavata@uniza.sk	Róberta Hlavatá	UNIZA	interný učiteľ	☑️	☐	✎ 🗑️
patrik.kamencay@uniza.sk	Patrik Kamencay	UNIZA	interný učiteľ	☑️	☐	✎ 🗑️
Riadky na stránku: 5 1-5 z 17 < >						

Obr. 5.10 – Tabuľka spravovania používateľov

5.5.5 Stránka spravovania externých škôl

K uvedenej stránke má taktiež prístup len správca cvičení. Zobrazuje sa ako spravovacia tabuľka, pričom účelom je vytváranie základných dát o externých školách. Zadajú sa údaje ako názov školy, adresa, kontaktná osoba a jej telefónne číslo. Vytvorené dáta potom správca priraduje používateľom v správe používateľov. Rovnako ako v predchádzajúcom prípade, do možností spravovania spadá vytváranie, úprava a vymazanie záznamov.

5.5.6 Stránka Cvičenia

Hlavnou súčasťou systému je definovanie a spravovanie údajov o cvičeniach. Dáta súčinne spravujú interní učitelia, ktorým správca cvičení vytvorené cvičenie prideliť. Prispôbia si údaje ako popis cvičenia, miestnosť, vyhovujúce začínajúce časy pre cvičenie a dokonca môžu pridať farbu, ktorá sa potom bude zobrazovať pri každom výskyte daného cvičenia v kalendári. Správcovia aj interní učitelia môžu zobrazovať stránku Cvičenia v dvoch pohľadoch, a to *tabuľka cvičení* a *karty cvičení*. Tieto dva pohľady môžu kedykoľvek prepínať pomocou tlačidla, ktoré mení stav zobrazenia. Správca cvičení vidí všetky cvičenia, pričom len správca cvičení dokáže vytvárať nové cvičenia. Interný učiteľ naopak v tabuľke vidí len cvičenia, ktoré mu správca pridelil a kliknutím na tlačidlo *spravovať* dokáže upravovať daný záznam. Ak sa jedná o externého učiteľa, tento používateľ nemá k tabuľke spravovania cvičení žiadny prístup. Nemá prístupné tlačidlo na zmenu stavu zobrazovania, preto sa mu na tejto stránke môžu zobraziť len karty cvičení. Karty cvičení sú zoskupené podľa zadaných odborov pre každé cvičenie. Po kliknutí na jedno z cvičení sa používateľom zobrazí dialógové okno s bližšími informáciami o danom cvičení. Externí učitelia majú možnosť v tomto okne kliknúť na tlačidlo *Rezervovať*, pričom sa ponúkne rezervačný formulár.

Detaily cvičenia

Vyberte cvičenie
Virtuálne TV štúdio (Live streaming)

Cvičenie: Virtuálne TV štúdio (Live streaming)

Vyučujúci: Uhrina, Bienik

Študenti sa na cvičení naučia pracovať s virtuálnym štúdiom TriCaster (obrazová a zvuková réžia, kľúčovanie, virtuálne TV štúdio, live strih).

Trvanie cvičenia je 50 minút.

Zvoľte dátum a čas

Zadajte počet študentov

Obsadenosť: 0/10

Zrušiť REZERVovať Zrušiť Rezervovať

Obr. 5.11 – Detaily cvičenia a rezervačný formulár

Vytvoriť nové cvičenie

Názov cvičenia *

Zvoľte študijný program ▼

Číslo učebne *

Priradte vedúcich cvičení ▼

Doba trvania (min) 50 min Čas začiatku ▼ Maximálny počet účastníkov 10

Popis

Priradte farbu cvičenia:

☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐

Zrušiť Pridať Cvičenie

Obr. 5.12 – Vytváranie nového cvičenia

Študijný program - Multimediálne Technológie

Virtuálne TV Štúdio (Live Streaming)

Od 3D Skenu Po 3D Tlač

Tvorba Hry Pre Virtuálnu Realitu

Úvod Do 3D Modelovania A 3D Animácie (Blender)

Obr. 5.13 – Karta s ponukou cvičení

5.5.7 Stránka Udalosti

Pri návrhu databázového modelu v časti 4.3 sme si uviedli, že kolekcia dát Udalosti je jedinou kolekciou, ktorá obsahuje viacnásobne vnorené dokumenty. Udalosť obsahuje otvorené cvičenia a každé otvorené cvičenie obsahuje na začiatku prázdne pole pre budúcich účastníkov. Štruktúra samotnej udalosti je jednoduchá. Z tohto dôvodu nepotrebujeme udalosti zobrazovať v spravovacej tabuľke, ale len ako jednoduchú kartu.

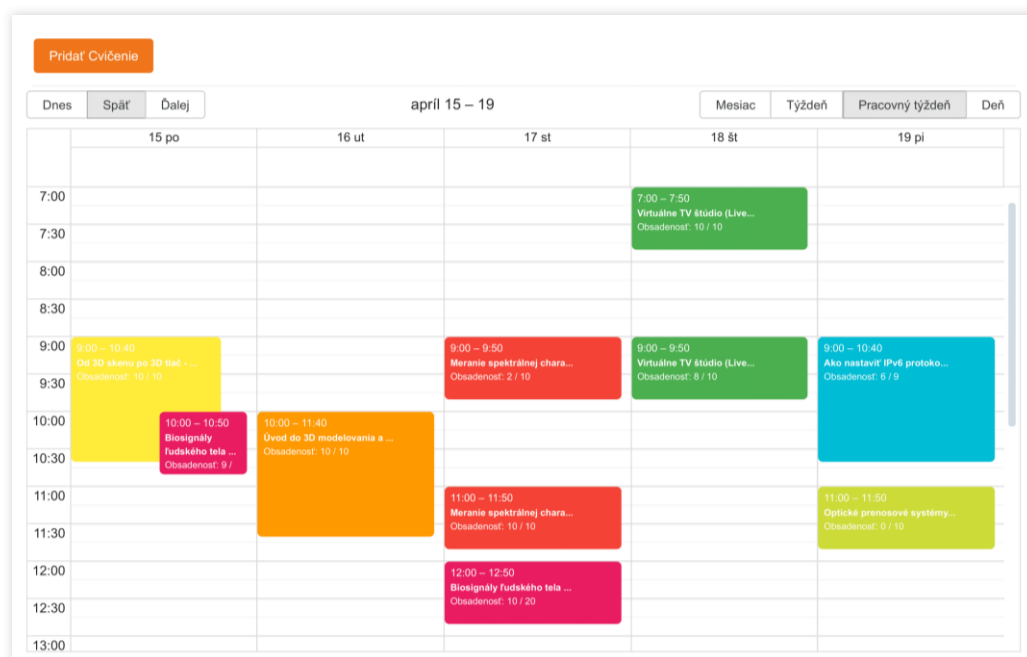


Obr. 5.14 – Karta udalosti

Súčasťou karty udalosti je aj tlačidlo *otvorené cvičenia*, ktoré nás presmeruje na inú adresu. Na tejto adrese vidíme všetky otvorené cvičenia pre danú udalosť, zobrazené v spravovacej tabuľke rovnako ako v predchádzajúcich prípadoch. Tu sa zobrazujú všetky podrobné údaje o konaní, kapacite a obsadenosti každého cvičenia. Správca cvičení má na tejto stránke otvorené cvičenia vytvárať, pričom dátum cvičenia môže byť len v rozmedzí začiatku a konca udalosti, ktorej je toto cvičenie priradené. Zadanie dátumu iného ako toto rozmedzie je zamedzené. Taktiež musí pri definovaní začiatku vyberať len z časov, ktoré definoval interný učiteľ, ktorému je toto cvičenie pridelené. Interní učitelia môžu túto stránku zobrazovať, ale nemajú možnosť spravovať dáta. Externí učitelia môžu takisto stránku len zobrazovať ale ponúka sa im aj možnosť rezervácie pre každý záznam v tabuľke.

5.5.8 Stránka Kalendár

Medzi najdôležitejšie komponenty, ktoré bolo potrebné implementovať patrila najmä kalendár, ktorý by nám umožnil zobrazovanie dátumov a časov konania cvičení v prehľadom formáte. K tomu nám pomohla inštalácia a využitie *react-big-calendar*. Tento komponent obsahoval komplexné funkcie a rôzne pohľady pre kalendárne zobrazenie udalostí v čase. Poskytoval mnoho možností prispôsobenia a spomedzi iného aj lokalizáciu a zmenu jazyka. Stránka sa zobrazuje všetkým používateľom, nezávisle od typu používateľa, avšak obsahuje niekoľko rozdielov v interakciách. Správcovia cvičení majú možnosť pridávať nové otvorené cvičenia, rovnako ako v podstránke otvorených cvičení v rámci stránky Udalosti. Po vytvorení sa cvičenia priamo zobrazia v kalendári. Pri vykonaní akcie kliknutia na jedno zo zobrazených cvičení v kalendári sa zobrazí dialóg s bližšími informáciami o danom cvičení, rovnako ako na stránke Cvičenia. Externí učitelia majú aj na tejto stránke možnosť rezervácie.



Obr. 5.15 – Kalendárne zobrazenie pracovného týždňa

5.5.9 Stránka Prihlášky

Táto stránka je určená primárne pre Externých učiteľov, aby mohli zobrazovať a prehľadávať svoje vykonané rezervácie. Prístup ku všetkým záznamom má správca cvičení, ktorého úlohou v rámci tejto stránky je schvaľovať rezervácie pred začiatkom danej udalosti. Interní učitelia nemajú k tomuto zobrazeniu žiadny prístup. Pokiaľ konkrétna rezervácia nie je schválená alebo zamietnutá správcom, učiteľ, ktorý danú rezerváciu vykonal má stále možnosť túto rezerváciu upraviť alebo zrušiť. Pri úprave môže zmeniť počet účastníkov, ktorý na začiatku zadal. Unikátnou funkcionalitou pre túto stránku je, že prihlásený učiteľ nezobrazuje len vlastné rezervácie, ale v prípade existujúcich záznamov od používateľov, ktorí majú priradenú rovnakú externú školu vidí aj rezervácie svojich kolegov.

5.5.10 Stránka História

V neposlednom rade, vychádzajúc z požiadaviek bolo potrebné implementovať zobrazenie záznamov, ktoré by externým učiteľom poskytovali prehľad o cvičeniach, ktoré už absolvovali a v danom čase už nie sú aktuálne. Túto požiadavku spíňa stránka História, ktorá poskytuje záznamy o rezerváciách z minulosti. V tomto prípade každý učiteľ môže zobrazovať len vlastné záznamy. Správca cvičení má taktiež prístup k histórii, pričom tieto záznamy zahŕňajú aj meno a kontakt na daného externého učiteľa. Táto stránka uzatvára implementáciu navrhnutých funkčných požiadaviek pre systém.

ZÁVER

Cieľom tejto bakalárskej práce bolo navrhnuť a implementovať online rezervačný systém, určený pre rezerváciu a spravovanie špeciálnych cvičení, ktoré ponúka fakulta FEIT Žilinskej univerzity v Žiline. V rámci teoretickej časti sme sa zaoberali predstavením rôznych možností návrhu webových aplikácií a obhájením technológií, ktoré sme si z týchto možností zvolili. V rámci praktickej časti sme vypracovali analýzu požiadaviek pre online rezervačný systém, vytvorenie návrhu nového optimalizovaného systému a postupnú implementáciu návrhu.

Rezervačný systém poskytuje personalizované rozhrania a funkcionality vzhľadom na typ používateľa. Osobe, ktorá bude systém spravovať poskytuje možnosti manipulácie so záznamami údajov pomocou špeciálnych tabuliek určených pre manipuláciu s veľkým množstvom dát. Pre vyučujúcich, ktorí budú dané špeciálne cvičenia zastrešovať, ponúka možnosť prispôbenia pridelených cvičení a určenia vhodných časov podľa vlastných možností a predstáv. Externým učiteľom poskytuje hlavne možnosť rezervácie ponúkaných cvičení a jednoduché zobrazenia pre prehľad o všetkých potrebných informáciách a stave rezervácií.

Aplikácia bola úspešne realizovaná na základe poskytnutých požiadaviek. V prípade schválenia zodpovednými pracovníkmi fakulty môže byť nasadená na server a uvedená do používania na fakultnej stránke.

Zoznam použitej literatúry

- [1] **S. S. N. Challapalli, P. Kaushik, S. Suman, B. D. Shivahare, V. Bibhu and A. D. Gupta**, "*Web Development and performance comparison of Web Development Technologies in Node.js and Python*,"in 2021 International Conference on Technological Advancements and Innovations (ICTAI), 2021, s. 303-307, doi: 10.1109/ICTAI53825.2021.9673464.
- [2] **V. Subramanian**, *Pro MERN Stack: Full Stack Web App Development with Mongo, Express, React, and Node*, Apress, 2017, s. 1-15. ISBN: 978-1-4842-4390-9
- [3] **M. Schwarzmüller**, "*Front-end vs Back-end*", [online]. Dostupné na: <<https://academind.com/tutorials/front-end-vs-back-end>>. [Prístup: 10. 10. 2023].
- [4] **MDN**, "*What is JavaScript?*", [online]. Dostupné na: <https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript> [Prístup: 12. 10 2023].
- [5] **GeeksforGeeks**, "*Top 7 Programming Languages for Back-end Web Development*", [online]. Dostupné na: <<https://www.geeksforgeeks.org/top-7-programming-languages-for-back-end-web-development/>> [Prístup: 21. 10. 2023].
- [6] **Python**, "*Applications for Python*", [online]. Dostupné na: <<https://www.python.org/about/apps/#web-and-internet-development>> [Prístup: 21. 10. 2023].
- [7] **PHP**, "*PHP: Hypertext Preprocessor*", [online]. Dostupné na: <<https://www.php.net/manual/en/intro-what-is.php>> [Prístup: 21. 10. 2023].
- [8] **Java**, "*What is Java?*", [online]. Dostupné na: <https://www.java.com/en/download/help/what-is_java.html> [Prístup: 21. 10. 2023].
- [9] **GeeksforGeeks**, "*What is Database?*", [online]. Dostupné na: <<https://www.geeksforgeeks.org/what-is-database/>> [Prístup: 21. 10. 2023].
- [10] **GeeksforGeeks**, "*Most Popular Databases*", [online]. Dostupné na: <<https://www.geeksforgeeks.org/most-popular-databases/>> [Prístup: 21. 10. 2023].
- [11] **MySQL**, "*What is mySQL?*", [online]. Dostupné na: <<https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html>> [Prístup: 21. 10. 2023].
- [12] **PostgreSQL**, "*About*", [online]. Dostupné na: <<https://www.postgresql.org/about/>> [Prístup: 21. 10. 2023].

- [13] **MongoDB**, "*Why use MongoDB*", [online]. Dostupné na: <https://www.mongodb.com/it-it/resources/products/fundamentals/why-use-mongodb> [Prístup: 21. 10. 2023].
- [14] **K. Kean, MakeUseOf** (30.9.2023), "*Full Stack JavaScript: Exploring MERN, MEAN, and MEVN*", [online]. Dostupné na: <https://www.makeuseof.com/mern-mean-mevn-full-stack-javascript/> [Prístup: 25. 10. 2023].
- [15] **M. Joshi, BrowserStack** (11.5.2023), "*Angular vs React vs Vue: Core Differences*", [online]. Dostupné na: <https://www.browserstack.com/guide/angular-vs-react-vs-vue> [Prístup: 25. 10. 2023].
- [16] **Stack Overflow** (2023), "*Stack Overflow Developer Survey*", [online]. <https://survey.stackoverflow.co/2023/#most-popular-technologies-language> [Prístup: 25. 10. 2023].
- [17] **OrioneSolutions, DataScienceCentral** (31.5.2021), "*MERN vs MEAN: Which stack to use in 2021*", [online]. Dostupné na: <https://www.datasciencecentral.com/mern-vs-mean-which-stack-to-use-in-2021/> [Prístup: 25. 10. 2023].
- [18] **Omnify Blog** (4.4.2020), "*One reservation system*", [online]. Dostupné na: <https://www.getomnify.com/blog/one-reservation-system> [Prístup: 19. 4. 2024].
- [19] **Codecademy**, "*MVC*", [online]. Dostupné na: <https://www.codecademy.com/article/mvc> [Prístup: 20. 4. 2024].
- [20] **Visual Paradigm**, "*What is UML?*", [online]. Dostupné na: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/> [Prístup: 20. 4. 2024].
- [21] **GeeksforGeeks**, "*Introduction of ER Model*", [online]. Dostupné na: <https://www.geeksforgeeks.org/introduction-of-er-model/> [Prístup: 20.4. 2024].
- [22] **Docker Documentation**, "*Overview*", [online]. Dostupné na: <https://docs.docker.com/get-started/overview/> [Prístup: 30. 4. 2024].
- [23] **IBM**, "*REST APIs*", [online]. Dostupné na: <https://www.ibm.com/topics/rest-apis> [Prístup: 31. 4. 2023].
- [24] **I. V. Abba, FreeCodeCamp** (28. 1. 2022), "*What is REST: REST API Definition for Beginners*", [online]. Dostupné na: <https://www.freecodecamp.org/news/what-is-rest-rest-api-definition-for-beginners/> [Prístup: 31. 4. 2024].
- [25] **GitHub**, zdrojový kód šablóny Mantis, [online]. Dostupné na: <https://github.com/codedthemes/mantis-free-react-admin-template>

PRÍLOHY

Zoznam príloh

Príloha A | Obsah USB

PRÍLOHA A

Priložené USB obsahuje:

- Prácu v elektronickej podobe (formát PDF)
- Zdrojový kód aplikácie vrátane inštrukcií pre inštaláciu