Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Overenie funkcionality a refaktorizácia systému virtuálnych výučbových prostredí

Bakalárska práca

2025 **Dmytro Kolosovskyi**

Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Overenie funkcionality a refaktorizácia systému virtuálnych výučbových prostredí

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra počítačov a informatiky (KPI)
Školiteľ:	Ing. Štefan Korečko, PhD.

Košice 2025 **Dmytro Kolosovskyi**

Abstrakt v SJ

Táto bakalárska práca opisuje vylepšenie a prepracovanie existujúcej vzdelávacej platformy pomocou technológie virtuálnej reality. Pomocou testovania a prepracovania softvérovej architektúry sa výrazne zlepšila škálovateľnosť, spoľahlivosť a udržiavateľnosť systému. Medzi hlavné dosiahnuté prínosy patrí zavedenie komplexného testovacieho prostredia, ktoré znížilo počet chýb a zlepšilo ďalší vývoj. Ďalším dôležitým zlepšením bolo zavedenie bezpečného autorizačného procesu vrátane obnovy hesla a prihlásenia pomocou Google OAuth2. Celkovo tieto zmeny prinášajú platforme moderný základ, na ktorom možno budovať ďalšie inovatívne riešenia, ktoré zabezpečujú vysokú dostupnosť a spoľahlivosť a poskytujú používateľom bezproblémový a bezpečný prístup k vzdelávacím zážitkom vo virtuálnej realite.

Kľúčové slová v SJ

Petryho siete, testovanie softvéru, spoľahlivosť systému, refaktorovanie, vzdelávacia aplikácia, virtuálna realita, škálovateľnosť, autorizácia

Abstrakt v AJ

This bachelor thesis describes the enhancement and redesign of an existing learning platform with virtual reality technology. With the help of testing and redesigning of the software architecture, the scalability, reliability and maintainability of the system have been significantly improved. The main benefits achieved include the introduction of comprehensive testing environment, which has reduced the number of bugs and improved further development. Another important improvement was the introduction of a secure authorization process, including password recovery and login using Google OAuth2. Overall, these changes bring the platform a modern foundation on which additional innovative solutions can be built, ensuring high availability and reliability and providing users with seamless, interactive, and secure access to educational experiences in virtual reality.

Kľúčové slová v AJ

Petry nets, software testing, system reliability, refactoring, educational application, virtual reality, scalability, authorization

Bibliografická citácia

KOLOSOVSKYI, Dmytro. *Overenie funkcionality a refaktorizácia systému virtuálnych výučbových prostredí*. Košice: Technická univerzita v Košiciach, Fakulta elektrotechniky a informatiky, 2025. 64s. Vedúci práce: Ing. Štefan Korečko, PhD.

77194

TECHNICKÁ UNIVERZITA V KOŠICIACH

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Katedra počítačov a informatiky

ZADANIE BAKALÁRSKEJ PRÁCE

Študijný odbor: **Informatika**
Študijný program: **Informatika**

Názov práce:

Overenie funkcionality a refaktorizácia systému virtuálnych výučbových prostredí

Functional Review and Refactoring of Virtual Educational Environment System

Študent: **Dmytro Kolosovskyi**
Školiteľ: **Ing. Štefan Korečko, PhD.**
Školiace pracovisko: **Katedra počítačov a informatiky**
Konzultant práce:
Pracovisko konzultanta:

Pokyny na vypracovanie bakalárskej práce:

1. Oboznámiť sa so systémom virtuálnych výučbových prostredí, vytvorenom na domovskom pracovisku.
2. Analyzovať existujúce riešenia pre vzdelávacie platformy a súvisiace architektonické koncepty.
3. Navrhnúť testovací proces pre systém virtuálnych výučbových prostredí, implementovať podporné prostriedky a proces realizovať.
4. Na základe analýzy a výsledkov testovacieho procesu navrhnúť a implementovať vybrané úpravy a rozšírenia systému virtuálnych výučbových prostredí.
5. Návrhové a implementačné práce koordinovať s ďalšími riešiteľmi prác na systéme virtuálnych výučbových prostredí.
6. Vypracovať dokumentáciu podľa pokynov vedúceho práce.

Jazyk, v ktorom sa práca vypracuje: slovenský
Termín pre odovzdanie práce: 23.05.2025
Dátum zadania bakalárskej práce: 31.10.2024

TECHNICKÁ UNIVERZITA V KOŠICIACH FAKULTA ELEKTROTECHNIKY A INFORMATIKY -10-

.....
prof. Ing. Liberios Vokorokos, PhD.
dekan fakulty

Čestné vyhlásenie

Vyhlasujem, že som záverečnú prácu vypracoval samostatne s použitím uvedenej odbornej literatúry.

Poďakovanie

Na tomto mieste by som rád poďakoval svojmu vedúcemu práce za jeho čas a odborné vedenie počas riešenia mojej záverečnej práce. Rovnako by som sa rád poďakoval dobrovoľníkom, ktorí sa zúčastnili na prieskume, svojim rodičom, priateľom a kolegom za ich podporu a povzbudzovanie počas celého môjho štúdia.

Obsah

1 Úvod **1**

2 Kľúčové pojmy a princípy **3**

2.1 Webový server 3

2.2 Testovanie softvéru a jeho dôležitosť 4

2.3 Analýza existujúcich riešení pred testovaním vzdelávacích platforiem 5

2.3.1 Vysvetlenie potreby výskumu 5

2.3.2 Metodika získavania dát 6

2.3.3 Testovanie hardvérovo-softvérovej architektúry pre virtuálnu realitu 7

2.3.4 Automatické testovanie scén virtuálnej reality 8

2.3.5 Spoľahlivosť softvérovej architektúry 9

2.4 Koncepty softvérovej architektúry používané v iných projektoch 10

2.4.1 Vysvetlenie potreby výskumu 10

2.4.2 Metodika získavania dát 11

2.4.3 Kľúčové kritériá spoľahlivosti a efektivity softvérovej architektúry 12

3 Analýza aktuálneho riešenia **16**

3.1 Technické charakteristiky projektu 16

3.2 Hlavné funkcie vzdelávacieho systému 16

3.3 Hodnotenie nedostatkov a návrhy na zlepšenie 17

4 Návrh riešenia **19**

4.1 Testovanie 19

4.1.1 Testovanie činnosti Spring Boot servera 19

4.1.2 Testovanie strany klienta v Angular 20

4.2 Aktualizácia verzií závislostí 21

4.3 Prepracovanie a opravy kódu 22
4.4 Obnovenie hesla počas autorizácie 23
4.5 Využívanie poskytovateľov tretích strán na rýchlejšiu autorizáciu. . 24
4.5.1 Návrh integrácie Google OAuth2 25

5 Testovanie serverovej vrstvy 26
5.1 Návrh testovacieho prostredia 26
5.2 Charakteristiky implementácie 27
5.2.1 Jednotkové Testy 27
5.2.2 Integračné Testy 29

6 Testovanie klientskej vrstvy 32
6.1 Návrh testovacieho prostredia 32
6.2 Charakteristiky implementácie 33

7 Aktualizácia verzií modulov a zastaraného kódu 35
7.1 Aktualizácia závislostí servera 35
7.1.1 Metodika odstraňovania slabín modulov 35
7.1.2 Aktualizácia konfigurácie Maven a jej členov s ďalšou ich kompatibilitou 37
7.2 Modifikacie závislostí používatelského rozhrania 38

8 Nastavenie nového hesla pri autorizácii 41
8.1 Modifikácia Spring API pre implementáciu funkcionality obnovy hesla 42
8.1.1 Komunikácia s databázou pri spracovaní ResetPassword tokenu 42
8.1.2 Funkcie v službách a spracovaníe žiadostí správcom autorizácie 43
8.2 Klientske rozhranie na zmenu prihlasovacích údajov 43
8.3 Testovanie novovytvorených komponentov 45

9 Rýchla a bezpečná autorizácia s Google OAuth2 47
9.1 Google Developer Console: prístup a nastavenia klienta 47
9.2 Metodika rozvrhnutia služieb a ich logika na každej vrstve 48
9.2.1 Podrobný rozbor kľúčových komponentov 49
9.3 Kontrola funkčnosti novej funkcionality pomocou testov 50

10 Vyhodnotenie práce **52**
10.1 Zhodnotenie kľúčových výsledkov pri vývoji testovacieho prostredia 52
10.1.1 Prínosy testovania 53
10.2 Rozšírené možnosti autorizácie 56
10.2.1 Obnovenie hesla 57
10.2.2 Autorizacia s pomocou Google 58
10.3 Návrhy na ďalšie zlepšenia 59

11 Záver **61**

Zoznam použitej literatúry **63**

Zoznam skratiek **65**

Zoznam príloh **67**

A Príručka na testovanie **68**
A.1 Aplikácia testovacieho prostredia 68
A.2 Spustenie prostredia 68
A.2.1 Požiadavky na softvér 68
A.2.2 Testovanie s pomocou Docker 69
A.2.3 Testovanie s pomocou Integrated Development Environment (IDE) 69
A.2.4 Testovanie pomocou príkazového riadku 71
A.3 Analýza výsledkov testov 72
A.3.1 Hlavný cieľ a čo je potrebné snažiť sa dosiahnuť 72
A.3.2 Interpretácia chýb v testoch 73
A.3.3 Kontrola pokrytia kódu testami 74

B Systémova príručka testovacieho prostredia **76**
B.1 Architektúra testovacieho prostredia servera Spring Boot 76
B.1.1 Jednotkové testy na serveri Spring Boot 77
B.1.2 Integračné testy na serveri Spring Boot 82
B.2 Testovacie prostredie klientskej časti v Angular 87
B.2.1 Konfigurácia testovacieho prostredia v Angular 87
B.2.2 Štruktúra testov klientskej časti 88
B.3 Testovanie komponentov virtuálneho prostredia 93

Zoznam obrázkov

2.1 Princíp práce webového servera [5] 4
2.2 Prehľad rámca VRTest [9] 9
2.3 Proces inžinierstva spoľahlivosti softvéru [17] 15

4.1 Zobrazenie starých a zraniteľných verzií v prostredí IntelliJ IDEA . 21

7.1 Zoznam zraniteľností neaktualizovaných Maven modulov v termináli 36
7.2 Zoznam zastaralých závislostí a najnovších ich verzií 37
7.3 Zoznam starých verzií modulov JavaScript a ich dostupných aktualizácií 39

8.1 Prechody medzi medzi koncovými bodmi počas obnovy hesla . . . 41
8.2 Interakcia medzi stránkami pri zmene hesla 44
8.3 E-mailové oznámenie na zmenu hesla 45
8.4 Formulár na zadanie a potvrdenie nového hesla 45

9.1 Panel vývojára a parametre autorizačného klienta 48
9.2 Sekvenčný diagram pre autorizáciu s Google OAuth2 [19] 49

10.1 Zvýšenie pokrytia kódu servera testami 53
10.2 Zvýšenie pokrytia kódu klienta testami 53
10.3 Analýza názorov používateľov na dôležitosť obnovy hesla 57
10.4 Návrhy používateľov na zlepšenie metódy obnovy hesla 58
10.5 Analýza spôsobu prihlásenia, ktorý si používatelia vybrali ako prvý 58
10.6 Ukazovatele dôvery používateľov pri odosielaní údajov spoločnosti Google 59

A.1 Zobrazenie dialógového okna v prostredí IntelliJ IDEA 70
A.2 Okno na rýchle spustenie a debagovanie v prostredí IntelliJ IDEA . 70
A.3 Konfigurácia front-endu v prostredí IntelliJ IDEA 71

A.4 Výsledky testov Karma v prehliadači 72
A.5 Úspešne vysledky testov 73
A.6 Problémy zistené pri testoch 74
A.7 Postup pokrytia testmi v prostredí IntelliJ IDEA 74
A.8 Vizualizácia pokrytia testmi v prehliadači 75

Zoznam tabuliek

2.1 Typické faktory kvality softvéru [6] 5
2.2 Tabuľka výskumných otázok pre metódy testovania 6
2.3 Použité dopyty na databázy pre testovacie metódy 7
2.4 Tabuľka výskumných otázok pre spoľahlivosť softvérovej architektury 11
2.5 Tabuľka použitých dopytov na databázy pre softvérovú architektúru 12

10.1 Prehľad testov, nájdených problémov a riešení 54

Zoznam výpisov

5.1 Štruktúra súborov testovacieho prostredia LirkisEduVePn-service . 26
5.2 Šablóna unit testu na príklade triedy UserServiceTest.java. 28
5.3 Trieda BaseIntegrationTest na konfiguráciu Testcontainers 30
6.1 Príklad unit testu pre Angular komponent 33
7.1 Import nástroja org.owasp.dependency-check-maven. 36
B.1 Služby servera sú testované pomocou jednotkových testov 78
B.2 Ovládače servera sú testované pomocou jednotkových testov 80
B.3 Štruktúra súborov integračných testov v LirkisEduVePn-service . . 82
B.4 Štruktúra súborov klientskej časti v Angular 88
B.5 Štruktúra súborov klientskej časti v Angular 94

1 Úvod

s Táto bakalárska práca nadväzuje na diplomovú prácu Ing. Adama Kašelu [1], bakalárske práce Bc. Dmytra Demianenka [2] a Bc. Lukáša Pisarčíka [3], a diplomovú prácu Ing. Juraja Rudyho [4], ktoré sa zaoberali tvorbou webovej platformy s prvkami rozšírenej reality založenej na Petriho sieťach. Výsledkom ich prác bolo výučbové prostredie virtuálnej reality zamerané na interaktívnu výučbu pre žiakov škôl, kde si mohli overiť svoje znalosti prostredníctvom interaktívnych úloh.

Predchádzajúce riešenie však vykazovalo určité nedostatky v štruktúre projektu. Počas analýzy neboli zistené žiadne testy na overenie správnosti funkčnosti a autorizácia neposkytovala bezpečný spôsob obnovy hesla v prípade jeho straty.

Táto práca sa preto zameriava na doplnenie testov, ktoré pomôžu nájsť a odstrániť mnohé chyby. Testovanie zároveň uľahčí budúcim vývojárom pridávanie nových funkcií bez rizika narušenia predchádzajúcich verzií projektu. Jedným z kľúčových cieľov tejto práce je tiež aktualizovať kód projektu vrátane importovaných externých modulov s cieľom odstrániť zraniteľnosti zastaraných verzií.

Plánuje sa tiež zlepšiť systém autorizácie používateľov pridaním možnosti obnoviť heslo v prípade jeho zabudnutia a urýchliť tento proces prihlásením do systému pomocou dôveryhodných služieb tretích strán.

Formulácia úlohy

Cieľom tejto práce je vylepšiť a aktualizovať existujúci výučbový systém, zlepšiť jeho architektúru a uľahčiť jeho používanie, čo prispeje k efektívnejšiemu a príjemnejšiemu používateľskému zážitku. Tento proces bude založený na moderných prístupoch k vývoju webových serverov na platformách Java Spring Boot a Angular a implementácii preverených testovacích techník.

Prvým krokom bude integrácia osvedčených postupov do architektúry servera, popísaná v **časti 4.3**. Kľúčovým prínosom bude pridanie systému na zaznamenávanie akcií, ktorý umožní jednoduchšie monitorovanie a sledovanie aktivít používateľov.

Snáď najdôležitejším a najrozsiahlejším krokom bude implementácia jednotkových a integračných testov, opísané v **prvých častiach kapitoly 4 a v kapitolách 5 a 6** v plnom rozsahu, ktoré zabezpečia vysokú kvalitu softvéru a minimalizujú riziko chýb. Výhodou tohto kroku je, že systém bude spoľahlivejší a budú sa rýchlejšie identifikovať a riešiť prípadné problémy, čo zjednoduší ďalší vývoj a údržbu projektu.

Ďalší krok zahŕňa vylepšenie autorizácie používateľov, ktorého implementácia je podrobnejšie opísaná v kapitole **"8 Nastavenie nového hesla pri autorizácii"**. Pridaním funkcie obnovy hesla a podpory prihlásenia cez technológiu OAuth2 sa zvýši bezpečnosť systému a zjednoduší sa prístup používateľov.

Finálnym výsledkom bude modernizovaný a optimalizovaný školiaci systém, ktorý bude bezpečnejší, ľahšie použiteľný a flexibilnejší pre ďalší rozvoj.

2 Kľúčové pojmy a princípy

Aby sme lepšie pochopili hlavné technológie a princípy použité v tejto práci, prvá časť sa zameria na definovanie kľúčových tém a základných konceptov. Toto je nevyhnutné pre zabezpečenie jednotného a konzistentného pohľadu na problematiku, čo umožní lepšiu orientáciu v nasledujúcom texte a hlbšiu analýzu navrhovaných riešení.

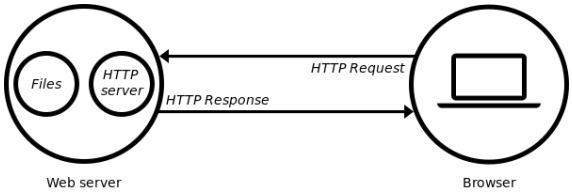
2.1 Webový server

Pojem webový server sa môže vzťahovať na hardvér alebo softvér, prípadne na obidva tieto pojmy. V článku [5] je webový server definovaný z oboch perspektív:

1. Z pohľadu hardvéru je webový server počítač, na ktorom je uložený softvér webového servera a súbory patriace k webovej stránke. Tento server je pripojený k internetu a umožňuje výmenu údajov s ďalšími zariadeniami pripojenými na web.

2. Z pohľadu softvéru webový server zahŕňa komponenty, ktoré určujú, ako používatelia webu získavajú prístup k nahratým súborom. Základnou súčasťou je Hypertext Transfer Protocol (HTTP) server – softvér, ktorý rozumie Uniform Resource Locator (URL) adresám a protokolu HTTP (ho prehliadač používa na zobrazovanie webových stránok). HTTP server spracúva požiadavky prostredníctvom doménových názvov webových stránok, ktoré uchováva, a doručuje obsah týchto stránok do zariadení používateľov.

Princíp webového servera môžeme znázorniť pomocou schémy, ako je znázornené na obrázku 2.1.



Obrázok 2.1: Princíp práce webového servera [5]

Projekt je založený na dynamickom webovom servere, ktorý pracuje s databázou a vykonáva predbežnú validáciu obsahu pred jeho doručením do prehliadača používateľa. V zmysle definície dynamického servera ide o kombináciu statického webového servera a dodatočného softvéru, ktorý zahŕňa aplikačný server a databázu. Aplikačný server interaguje s databázou, spracováva požiadavky používateľov a dynamicky generuje obsah na základe vstupov a údajov uložených v databáze. Tento obsah je následne pred odoslaním cez HTTP server validovaný, čím sa zabezpečí, že používateľ dostane správne a aktuálne údaje. Tieto vlastnosti sú kľúčové pre dynamické spracovanie údajov a interakciu medzi klientom a serverom, čo projekt plne spĺňa.

2.2 Testovanie softvéru a jeho dôležitosť

Testovanie programov je neoddeliteľnou súčasťou vývoja softvérových systémov a zohráva kľúčovú úlohu pri zabezpečovaní kvality a funkčnosti vyvíjaného riešenia. Aby sme si upevnili správnu definíciu testovania, je potrebné analyzovať používanie slov „úspešný“ a „neúspešný“ - najmä ich používanie pri kategorizácii výsledkov testovacích prípadov. Ako bolo povedané v knihe ”The Art of Software Testing” [6], testovací prípad, ktorý nájde novú chybu, nemožno považovať za neúspešný; skôr sa ukázal ako cenná investícia..

Kvalitu softvéru nemožno priamo merať, čo je dané už samotnou jeho nehmotnou povahou. Na jej opis boli rôznymi normami (napr. ISO 9126) definované atribúty kvality softvéru, ktoré opisujú vlastnosti softvérového produktu z rôznych pohľadov [7]. Kvalita má tri skupiny faktorov - funkčnosť, technické riešenie a prispôsobivosť. Tieto tri súbory faktorov si môžeme predstaviť ako rozmery v priestore kvality softvéru. Každú dimenziu možno rozdeliť na jej zložky, faktory a úvahy na postupne nižších úrovniach podrobnosti. Tabuľka 2.1 znázorňuje niektoré z najčastejšie uvádzaných aspektov kvality.

Funkčnosť (vonkajšia kvalita)	**Inžinierstvo (vnútorná kvalita)**	**Prispôsobiteľnosť (budúca kvalita)**
Správnosť	Efektivita	Flexibilita
Spoľahlivosť	Testovateľnosť	Znovupoužiteľnosť
Použiteľnosť	Dokumentácia	Udržateľnosť
Integrita	Štruktúra	

Tabuľka 2.1: Typické faktory kvality softvéru [6]

Dobré testovanie poskytuje opatrenia pre všetky relevantné faktory. Dôležitosť jednotlivých faktorov sa líši od aplikácie k aplikácii. Každý systém, v ktorom ide o ľudské životy, musí klásť mimoriadny dôraz na spoľahlivosť a integritu. V typickom obchodnom systéme sú kľúčovými faktormi použiteľnosť a udržateľnosť, zatiaľ, čo pre jednorazový vedecký program nemusí byť ani jeden z nich významný. Ak má byť naše testovanie plne účinné, musí byť zamerané na meranie každého relevantného faktora, a tak prinútiť kvalitu, aby sa stala hmatateľnou a viditeľnou [6].

2.3 Analýza existujúcich riešení pred testovaním vzdelávacích platforiem

V tejto časti budeme analyzovať dostupné zdroje informácií o princípoch testovania vo vzdelávacích platformách a popíšeme, či je v danom projekte skutočne potrebné zaoberať sa testovaním.

2.3.1 Vysvetlenie potreby výskumu

V prípade tohto projektu, je testovanie obzvlášť dôležité. Jeho hlavnou úlohou je overiť, či softvér správne funguje v prostredí Virtual Reality (VR), či poskytuje neprerušovaný a interaktívny proces výučby a či spĺňa všetky technické a pedagogické požiadavky. Efektívne testovanie pomáha identifikovať chyby a nedostatky systému ešte pred jeho uvedením do prevádzky, čím sa minimalizuje riziko narušenia počas vzdelávacích aktivít. Overuje sa, či softvér správne reaguje na rôzne vstupy od študentov, či je stabilný a bezpečný. V kontexte tejto platformy VR je testovanie kľúčové nielen na zabezpečenie bezproblémovej prevádzky, ale aj na vytvorenie pútavého a efektívneho vzdelávacieho zážitku, ktorý podporuje učenie a interakciu v novom prostredí.

Pre komplexné a spoľahlivé vypracovanie tohto projektu je nevyhnutné vykonať podrobnejší výskum dostupnej akademickej literatúry. Podrobná štúdia výskumov týkajúcich sa VR vo vzdelávaní a metód testovania v nových technológiách umožní lepšie pochopenie problémov a výziev, s ktorými sa môžeme stretnúť. Tento výskum by mohol poskytnúť cenné poznatky o najlepších praktikách a prístupoch, ktoré sú už osvedčené, čím sa podporí kvalitnejší vývoj a implementácia VR platformy.

2.3.2 Metodika získavania dát

Prvým krokom je formulovanie výskumných otázok na vytvorenie dotazov a nájdenie najrelevantnejších zdrojov. V tomto prípade analyzujeme nové testovacie metódy a koncepty používané vo vzdelávacích platformách. Hlavná otázka je uvedená v tabuľke 2.2:

Otázka	**Motivácia**
Aké testovacie metódy sa používajú v iných úspešných vzdelávacích platformách s technológiou VR?	Cieľom je zistiť použité koncepty testovania a ich prínos pre výučbovú platformu

Tabuľka 2.2: Tabuľka výskumných otázok pre metódy testovania

Po určení problému a správnom stanovení vedeckých otázok je nevyhnutné špecifikovať metodiku zberu dát. Na vyhľadávanie sme využili dotazy do databáz ACM Digital Library a Scopus. V tabuľke 2.3 sú uvedené hľadané výrazy týkajúce sa našej témy.

Databáza	Dopyt
ACM Digital Library	(TITLE-ABS-KEY(”testing methodologies” OR ”evaluation methods” OR ”assessment techniques” OR ”software testing”) AND (”educational platforms” OR ”e-learning systems” OR ”learning management systems” OR ”virtual learning environments” OR ”VR platforms”) AND (”virtual reality” OR ”VR” OR ”extended reality” OR ”XR”))
Scopus	(TITLE-ABS-KEY(”testing methodologies” OR ”evaluation methods” OR ”assessment techniques” OR ”software testing”) AND (”educational platforms” OR ”e-learning systems” OR ”learning management systems” OR ”virtual learning environments” OR ”VR platforms”) AND (”virtual reality” OR ”VR” OR ”extended reality” OR ”XR”))
Web of Science	TS=(”testing methodologies” OR ”evaluation methods” OR ”assessment techniques” OR ”software testing”) AND TS=(”virtual reality” OR ”VR” OR ”extended reality” OR ”XR”)

Tabuľka 2.3: Použité dopyty na databázy pre testovacie metódy

2.3.3 Testovanie hardvérovo-softvérovej architektúry pre virtuálnu realitu

Projekt opísaý v článku [8] sa zaoberal testovaním softvéru, pričom sa zameral na overovacie procesy nevyhnutné pre hardvérové aj softvérové komponenty v rámci VR. Tu je zhrnutie kľúčových aspektov ich prístupu k testovaniu a dôvodov, prečo by mohol byť užitočný pre vašu vzdelávaciu platformu založenú na VR:

1. **Testovanie založené na simulácii:** Projekt využíval simulácie na testovanie rôznych softvérových komponentov v kontrolovanom virtuálnom prostredí. Táto metóda im umožnila identifikovať chyby a problémy s výkonom pred nasadením v reálnom prostredí.

2. **Modulárne testovanie:** Zamerali sa na samostatné testovanie jednotlivých modulov v rámci systému (napríklad senzorov, grafických rozhraní a riadiacich mechanizmov). Tento modulárny prístup zabezpečil, aby každá časť architektúry fungovala podľa očakávaní pred jej integráciou s ostatnými.

3. **Testovanie výkonnosti v reálnom čase:** Keďže projekt zahŕňal systémy pracujúce v reálnom čase, osobitná pozornosť sa venovala testovaniu výkonnosti operácií citlivých na čas, čím sa zabezpečilo, že údaje zo senzorov, vstupy používateľa a procesy vykresľovania prebiehali bez oneskorenia.

4. **Testovanie integrácie hardvéru a softvéru:** Testovanie zahŕňalo posúdenie toho, ako dobre softvér spolupracuje s hardvérom (napr. snímačmi, procesormi), aby sa zabezpečil hladký používateľský zážitok. Vykonali sa kontroly kompatibility medzi hardvérom a softvérom VR.

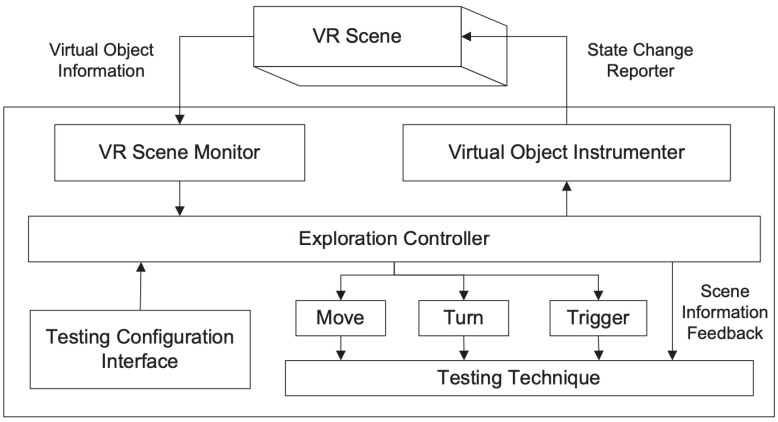
5. **Iteratívny proces testovania:** Po každom vývojovom cykle nasledovali kolá testovania, v ktorých sa odstraňovali chyby a systém sa postupne zdokonaľoval.

Ak hovoríme o projekte tejto práce, mohli by sme použiť simulačné testovanie na vyhodnotenie rôznych virtuálnych scenárov, ktoré bude platforma používať, napríklad prostredia triedy alebo interakcie študentov, aby sme včas odhalili problémy. Táto platforma zahŕňa rôzne VR prostredia, teda používateľské rozhrania a interaktívne vzdelávacie nástroje, pričom testovanie každej zložky jednotlivo by pomohlo zabezpečiť odolnosť pred úplnou integráciou systému. Vzhľadom na to, že vzdelávacie zážitky vo VR sú vysoko interaktívne, testovanie výkonnosti v reálnom čase, ktoré sa v projekte spomína, by bolo rozhodujúce pre zabezpečenie plynulého, pohlcujúceho zážitku pre študentov bez oneskorení alebo lagov. Iteratívny prístup k testovaniu by umožnil postupne zlepšovať a zdokonaľovať platformu a zabezpečiť, aby každá aktualizácia alebo nová funkcia dobre fungovala v rámci existujúceho systému.

2.3.4 Automatické testovanie scén virtuálnej reality

Aby sme pochopili prístupy používané pri testovaní rôznych scén vo virtuálnej realite, môžeme sa pozrieť na príklad práce od Xiaoyina Wanga [9], ktorá predstavuje rámec VRTest, ktorý automatizuje testovanie VR scén. Tento nástroj dokáže analyzovať VR prostredie, ovládať pohyb kamery a testovať interakcie s virtuálnymi objektmi podľa vopred definovaných stratégií, ako je znázornené na ob-

rázku 2.2. V rámci frameworku autori implementovali dve testovacie metódy: VRMonkey, ktorá využíva čisto náhodný prístup inšpirovaný „Monkey Testing“ známym z mobilného vývoja, a VRGreed, ktorá pomocou chamtivého algoritmu (greedy algorithm) cielene preskúmava interaktívne objekty s cieľom maximalizovať pokrytie testovaných prvkov.



Obrázok 2.2: Prehľad rámca VRTest [9]

Štúdia [9] ukázala, že VRTest je flexibilný a ľahko rozšíriteľný framework, ktorý sa dá prispôsobiť rôznym VR aplikáciám. Výsledky testovania potvrdili, že stratégia VRGreed výrazne zlepšuje efektivitu testovania – dokázala zvýšiť pokrytie interaktívnych objektov o 55 percentuálnych bodov v porovnaní s VRMonkey. To naznačuje, že cielené algoritmické prístupy môžu výrazne zlepšiť schopnosť automatizovaných nástrojov odhaľovať chyby v VR softvéri.

Integrácia podobných metód do testovania vzdelávacích VR platforiem by mohla výrazne uľahčiť proces validácie, znížiť potrebu manuálneho testovania a zároveň zvýšiť spoľahlivosť softvéru. Napríklad pri vývoji VR výučbových systémov by sa automatizované testovacie nástroje mohli použiť na overenie správneho fungovania interaktívnych prvkov, detekciu problémov s pohybom používateľskej kamery či analýzu použiteľnosti rozhrania. Tieto výsledky podčiarkujú dôležitosť ďalšieho rozvoja automatizovaných testovacích nástrojov pre VR softvér, aby bol schopný spĺňať rastúce nároky na kvalitu a stabilitu.

2.3.5 Spoľahlivosť softvérovej architektúry

Architektúra softvéru definuje štruktúru systému, jeho základné komponenty a spôsob, akým medzi sebou tieto komponenty interagujú. Kľúčovým aspektom je to, že architektúra vytvára základné rozhodnutia, ktoré ovplyvňujú kvalitu softvéru počas jeho celého životného cyklu. Tieto rozhodnutia sú neoddeliteľnou

súčasťou návrhu a vývoja, ale zároveň hrajú zásadnú úlohu aj v údržbe a evolúcii systému [10]. Tieto aspekty sa prejavujú v rôznych oblastiach softvérového životného cyklu, ako sú požiadavky na výkon, bezpečnosť, udržateľnosť a škálovateľnosť.

Správna architektúra má priamy vplyv na schopnosť systému prispôsobovať sa novým požiadavkám a meniacej sa technológii. V dynamickom prostredí, kde sa požiadavky rýchlo menia, musí byť architektúra navrhnutá tak, aby umožnila flexibilitu a reagovala na nové výzvy. Nesprávna architektúra vedie k zvýšenej zložitosti a nákladom pri rozvoji a údržbe systému, čo môže viesť k architektonickému dlhu, ktorý v konečnom dôsledku znižuje efektívnosť celého riešenia.

2.4 Koncepty softvérovej architektúry používané v iných projektoch

V tejto časti sa budeme zaoberať princípmi a postupmi uvedenými v iných prácach na zníženie zraniteľnosti softvérovej architektúry voči poruchám.

2.4.1 Vysvetlenie potreby výskumu

Efektívna softvérová architektúra podporuje nielen funkčnosť systému, ale aj jeho udržateľnosť a rozšíriteľnosť v budúcnosti. Tento projekt má ambíciu vytvoriť škálovateľnú platformu, ktorá bude schopná spracovávať rôzne scenáre výučby, pričom zabezpečí stabilný výkon bez ohľadu na počet používateľov alebo komplexnosť učebných modulov. Správne navrhnutá architektúra tiež zaisťuje, že systém je ľahko rozšíriteľný o nové funkcie a moduly, čo umožňuje ďalší vývoj a integráciu moderných vzdelávacích technológií.

Výskum správnych architektonických prístupov je kľúčový pre pochopenie osvedčených praktík, ktoré umožnia riešiť problémy spojené s výkonnosťou, spoľahlivosťou a bezpečnosťou. Štúdia literatúry o moderných architektonických vzoroch, komponentovej architektúre, mikroservisných riešeniach a cloudových platformách umožní identifikovať najlepšie riešenia, ktoré môžu byť aplikované v tomto projekte. Architektúra systému, ktorá je navrhnutá s ohľadom na bezpečnosť, testovateľnosť a odolnosť voči poruchám, bude prínosom nielen pre samotný systém, ale aj pre jeho užívateľov, ktorí sa naň budú môcť spoľahnúť počas vzdelávacích aktivít.

2.4.2 Metodika získavania dát

Prvým krokom je formulovanie výskumných otázok na vytvorenie dotazov a nájdenie najrelevantnejších zdrojov. V tomto prípade analyzujeme nové testovacie metódy a koncepty používané vo vzdelávacích platformách. Hlavná otázka je uvedená v tabuľke 2.4:

Otázka	**Motivácia**
Aké prístupy sa používajú na udržanie spoľahlivosti softvérovej architektúry v riešeniach s technológiou VR?	Motiváciou je preskúmanie používaných metód zabezpečenia spoľahlivosti softvérovej architektúry

Tabuľka 2.4: Tabuľka výskumných otázok pre spoľahlivosť softvérovej architectury

Po určení problému a správnom stanovení vedeckých otázok je nevyhnutné špecifikovať metodiku zberu dát. Na vyhľadávanie sme využili dotazy do databáz ACM Digital Library, Google Scholar a Scopus. V tabuľke 2.5 sú uvedené hľadané výrazy týkajúce sa našej témy.

Databáza	Dopyt
ACM Digital Library	TITLE-ABS-KEY(”reliability of software architecture” AND (”virtual reality” OR ”VR” OR ”extended reality” OR ”XR”) AND (”educational platforms” OR ”e-learning” OR ”education technology”) AND (scalability OR ”system performance” OR ”fault tolerance” OR maintainability))
Google Scholar	TITLE-ABS-KEY(”reliability of software architecture” AND (”virtual reality” OR ”VR” OR ”extended reality” OR ”XR”) AND (”educational platforms” OR ”e-learning” OR ”education technology”) AND (scalability OR ”system performance” OR ”fault tolerance” OR maintainability))
Scopus	TITLE-ABS-KEY(”reliability of software architecture” AND (”virtual reality” OR ”VR” OR ”extended reality” OR ”XR”) AND (”educational platforms” OR ”e-learning” OR ”education technology”) AND (scalability OR ”system performance” OR ”fault tolerance” OR maintainability))
Web of Science	TS=(”reliability of software architecture” AND (”virtual reality” OR ”VR” OR ”extended reality” OR ”XR”) AND (”educational platforms” OR ”e-learning” OR ”education technology”) AND (scalability OR ”system performance” OR ”fault tolerance” OR maintainability))

Tabuľka 2.5: Tabuľka použitých dopytov na databázy pre softvérovú architektúru

2.4.3 Kľúčové kritériá spoľahlivosti a efektivity softvérovej architektúry

Dobre navrhnutá softvérová architektúra musí spĺňať niekoľko kľúčových kritérií, ktoré sú nevyhnutné pre jej úspešnosť a spoľahlivosť:

1. **Škálovateľnosť:** Architektúra by mala byť schopná podporovať rastúci počet používateľov alebo rastúce množstvo spracovávaných dát bez zhoršenia výkonnosti [11]. To zahŕňa podporu horizontálnej aj vertikálnej škálovateľnosti, či už ide o rozširovanie infraštruktúry alebo optimalizáciu existujúcich zdrojov.

2. **Modularita a udržiavateľnosť:** Dobre navrhnutá architektúra rozdeľuje systém na menšie, nezávislé moduly, ktoré majú jasne definované zodpovednosti. Tým sa uľahčuje vývoj, údržba a testovanie jednotlivých častí systému. Modularita zároveň zvyšuje flexibilitu pri pridávaní nových funkcií alebo opravovaní chýb. [12]

3. **Bezpečnosť:** Softvérová architektúra musí zahŕňať prvky, ktoré chránia systém pred vonkajšími hrozbami. Toto zahŕňa šifrovanie údajov, autentifikáciu a autorizáciu, ale aj správne definované zásady prístupu (tzv. princíp minimálnych právomocí) [13].

4. **Výkon a efektívnosť:** Systém by mal byť optimalizovaný tak, aby správne využíval dostupné zdroje. To zahŕňa správne spracovanie požiadaviek, rovnomerné rozloženie záťaže a použitie cache pre zrýchlenie výkonu. Výkonnosť je jedným z faktorov, ktoré môžu priamo ovplyvniť spokojnosť koncových používateľov.

5. **Odolnosť a spoľahlivosť:** Architektúra musí byť navrhnutá tak, aby bola odolná voči chybám a dokázala sa vyrovnať s výpadkami časti systému bez zásadného narušenia jeho prevádzky. To zahŕňa prvky ako redundancia, automatické zotavenie a kontrola výpadkov (napr. pomocou techniky "circuit breaker") [14].

6. **Dokumentácia a sledovateľnosť rozhodnutí:** Aby bola architektúra udržateľná, je potrebné zabezpečiť kvalitnú a aktuálnu dokumentáciu. Tá slúži ako zdroj informácií pre ďalších vývojárov a pomáha im pochopiť, prečo boli určité rozhodnutia prijaté [15].

Jedným z prístupov na zvýšenie spoľahlivosti je využitie mechanizmov pre toleranciu chýb. V práci [16] sa uvádza, že medzi bežné techniky patrí napríklad spracovanie výnimiek, ktoré sa používa na úrovni zdrojového kódu na spracovanie nečakaných chýb a pokračovanie v operáciách, alebo zotavovacie bloky, ktoré sú implementované priamo na architektonickej úrovni. Ďalšími dôležitými prístupmi sú kontrolné mechanizmy ako watchdog alebo heart-beat, ktoré monito-

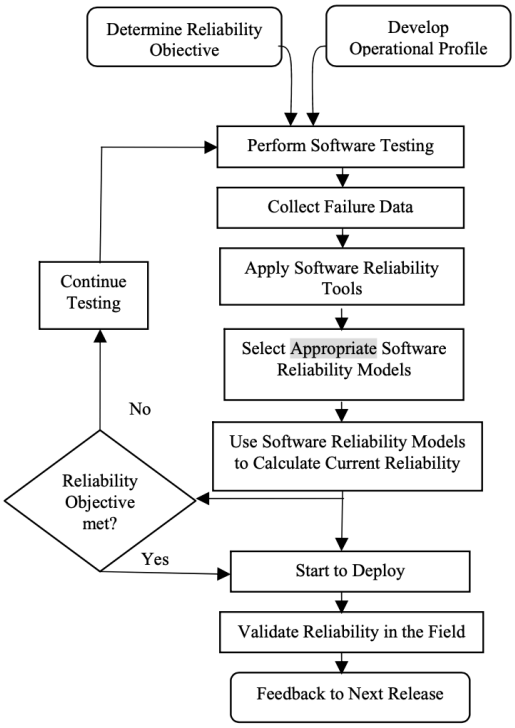
rujú stav systému a v prípade detekcie problému reštartujú procesy, aby nedošlo k zlyhaniu celého systému [16].

Na zvýšenie spoľahlivosti softvérovej architektúry je možné implementovať aj replikáciu komponentov, čo znamená, že rovnaká úloha sa vykonáva viacerými nezávislými komponentmi. V prípade, že jeden komponent zlyhá, systém môže pokračovať pomocou ďalšieho komponentu [16]. Táto technika znižuje pravdepodobnosť úplného zlyhania systému a tým priamo zvyšuje jeho spoľahlivosť.

Softvérové komponenty sú prideľované k softvérovým zdrojom, napríklad procesorom alebo pevným diskom, pričom každému zdroju patrí určitý čas, po ktorý vydrží bez poruchy (Mean Time To Failure (MTTF) - priemerný čas výskytu poruchy) a do doby, než ho bude možné opraviť (Mean Time To Repair (MTTR) – priemerný čas opráv daného zdroja) [16]. Kombinovaním týchto dvoch parametrov je možné tiež vypočítať celkovú spoľahlivosť systému a posúdiť, či je daný systém schopný sa uspokojivo udržiavať v chode.

Pretože pri návrhu softvérovej architektúry sa rozoberajú rôzne alternatívy a ich konfigurácie, nemožno v takom prípade hovoriť o štandardizácii softvérových komponentov. Rôzne architektúry môžu mať rôznu toleranciu na chyby a iné hardvérové architektúry, čo neoddeliteľne ovplyvňuje spoľahlivosť systému v konečnom dôsledku. Pri použití metód, akými sú napríklad Markovove reťazce [16], je možné štruktúry modelovať, teda aj ich poruchy a opravy a stanoviť medzi čím môžeme simulovať poruchy a opravy a predpovedať výsledné hodnoty spoľahlivosti pre každú zvolenú variantu.

V práce [17] je pekne opísaný prehľad procesov inžinierstva spoľahlivosti softvéru (pozrite si obrázok 2.3).



Obrázok 2.3: Proces inžinierstva spoľahlivosti softvéru [17]

Najprv sa kvantitatívne určí cieľ spoľahlivosti, a definuje sa využitie zákazníka vytvorením prevádzkového profilu. Softvér sa potom testuje podľa prevádzkového profilu, zbierajú sa údaje o poruchách a počas testovania sa sleduje spoľahlivosť s cieľom určiť čas vydania produktu. Táto činnosť sa môže opakovať, kým sa nedosiahne určitá úroveň spoľahlivosti.

3 Analýza aktuálneho riešenia

Softvérový projekt, ktorý budeme testovať a vylepšovať, vychádza z diplomovej práce Adama Kašelu [1] a je ďalej rozšírený bakalárskymi prácami Bc. Lukáša Pisarčíka, Bc. Dmytra Demianenka [2] a diplomovou pracou Ing. Juraja Rudyho [4]. Cieľom projektu je využitie Petriho sietí na vytvorenie efektívneho a interaktívneho virtuálneho vzdelávacieho prostredia. Zameriava sa na implementáciu flexibilného systému ovládania pre akékoľvek vzdelávacie scény, čo umožňuje prispôsobiť prostredie špecifickým potrebám používateľov. Okrem základných funkcií ponúka tento systém aj štatistické sledovanie a analýzu aktivity študentov, čím sa podporuje dynamické vzdelávanie.

3.1 Technické charakteristiky projektu

Projekt je postavený na kontajnerovej platforme Docker, čo umožňuje izoláciu jednotlivých častí systému, čím sa zjednodušuje správa komponentov, ako je backend (Java Spring) a frontend (Angular). Na vytváranie 3D a virtuálnych prostredí sa využíva rámec A-Frame, ktorý umožňuje jednoduché začlenenie do Hypertext Markup Language (HTML) prostredia. Kombináciou týchto technológií vzniká dynamické prostredie s vysokou mierou interaktivity a prístupnosti.

3.2 Hlavné funkcie vzdelávacieho systému

Po analýze súčasného stavu projektu sme identifikovali nasledujúce funkcie:

1. **Riadenie vzdelávacích scenárov prostredníctvom Petriho sietí**

   Petriho siete umožňujú efektívne riadenie priebehu jednotlivých vzdelávacích scenárov. Pomocou týchto sietí je možné definovať sekvencie úloh a riadiť ich priebeh v prostredí, ktoré reaguje na študentove akcie. Implementácia Petriho sietí zabezpečuje logickú správu úloh a interakcií medzi jednotlivými prvkami prostredia.

2. **Rozšírenie scén a pridanie technických prvkov**

   Projekt zahŕňa rozšírenie pôvodnej scény o nové technické prvky a interaktívne možnosti. Pridané ovládacie centrum umožňuje sledovanie stavu jednotlivých úloh a poskytuje používateľovi prehľad o tom, ktoré úlohy sú splnené a ktoré je potrebné dokončiť. Okrem toho boli doplnené ovládacie tlačidlá a popisy prvkov, ktoré pomáhajú študentom lepšie sa orientovať.

3. **Komunikácia s backendom a úprava obsahu**

   Pre flexibilitu systému bola implementovaná možnosť, aby si používateľ mohol načítať vlastnú scénu, scenár alebo jazykové nastavenie. Týmto spôsobom je možné prispôsobiť obsah vzdelávacieho prostredia podľa aktuálnych potrieb používateľa, čo podporuje personalizované vzdelávanie.

4. **Interakcia pomocou VR ovládačov**

   Pre zvýšenie interaktivity bola do scény pridaná podpora VR ovládačov, vrátane implementácie gest ako Hover, Grab, Stretch a Drag-drop [4]. Tento prístup umožňuje používateľovi aktívnejšie zapojiť sa do prostredia a manipulovať s prvkami priamo rukami, čo posilňuje ich zážitok a efektivitu vzdelávania.

3.3 Hodnotenie nedostatkov a návrhy na zlepšenie

Prvým zisteným problémom bolo, že v priebehu analýzy neexistovali žiadne testy na overenie správnosti funkčnosti systému. Chýbajúce testovacie postupy znamenajú, že neboli zavedené žiadne mechanizmy na kontrolu, či jednotlivé časti systému fungujú správne a bez chýb. Nedostatok testov znižuje dôveryhodnosť celého riešenia a sťažuje identifikáciu potenciálnych problémov alebo nedostatkov ešte predtým, ako sa systém dostane do reálneho použitia. V dôsledku toho môže dochádzať k neočakávaným chybám pri používaní, čo môže narušiť učebný proces a spôsobiť frustráciu u učiteľov aj študentov.

Ďalším dôležitým aspektom, ktorý je potrebné zvážiť pri vylepšovaní systému, je implementácia zaznamenávania akcií na aplikačnom serveri. To je kľúčovým mechanizmom na monitorovanie a zaznamenávanie činností vykonávaných používateľmi a samotným systémom, čo môže výrazne prispieť k zvýšeniu bezpečnosti, transparentnosti a diagnostiky prípadných problémov. V prípade neočakávaných chýb alebo nesprávneho správania systému, záznamy akcií umožňujú rýchlu identifikáciu príčiny a tým aj efektívnejšie riešenie problémov. Pre

učiteľov by navyše poskytovali prehľad o tom, kedy a aké úlohy študenti plnia, čo by pomohlo lepšie monitorovať ich progres a efektívnejšie riadiť vyučovací proces. Zaznamenávanie by takisto umožnilo auditovanie citlivých operácií, čím by sa posilnila celková dôveryhodnosť systému a jeho odolnosť voči zneužitiu.

Ešte jedným dôležitým vylepšením systému by bolo pridať funkciu obnovenia hesla v prípade, že ho užívateľ zabudne. V aktuálnej verzii sa autorizácia vykonáva pomocou e-mailu a hesla, avšak chýba možnosť obnovy hesla, čo predstavuje vážny problém, keďže ak užívateľ zabudne svoje prihlasovacie údaje, nemá žiadny spôsob, ako získať prístup k svojmu účtu. To môže viesť k frustrácii a nutnosti kontaktovať technickú podporu, čím sa zvyšuje zbytočná záťaž na obe strany. Implementácia jednoduchého mechanizmu, ako je odkaz na obnovenie hesla, ktorý by bol zaslaný na e-mailovú adresu užívateľa, by výrazne zvýšila užívateľský komfort a umožnila rýchle riešenie problémov s prihlásením, čím by sa predišlo strate prístupu k dôležitým údajom alebo funkciám platformy. Tento krok by tiež prispel k zvýšeniu bezpečnosti a celkovej efektívnosti systému, pretože užívatelia by mali možnosť kedykoľvek obnoviť svoje heslo bez potreby externej pomoci.

Počas úprav nášho projektu sme narazili na problémy súvisiace s nekompatibilitou niektorých verzií importovaných modulov s automaticky aktualizovanou závislosťou node.js. To znamená, že závislosti v našom projekte je potrebné aktualizovať na najnovšie možné verzie a určite skontrolovať kompatibilitu. To isté platí aj pre verzie v kontajneroch v prostredí Docker.

Nasledujúcou možnosťou na zlepšenie systému by bolo zavedenie zjednodušenej autorizácie pomocou Google OAuth2. Táto funkcia by umožnila používateľom prihlásiť sa do platformy bez toho, aby museli vytvárať nové heslo alebo účet, čo by výrazne zjednodušilo celý proces registrácie a prihlásenia. Pomocou Google OAuth2 sa používatelia budú môcť rýchlo a bezpečne prihlásiť pomocou svojho existujúceho účtu Google, čím sa odstráni potreba pamätať si ďalšie prihlasovacie údaje. Týmto krokom sa zlepší nielen používateľský zážitok, ale aj bezpečnosť, keďže Google OAuth2 poskytuje spoľahlivé mechanizmy ochrany údajov a dvojfaktorovú autentifikáciu. Zavedenie tejto funkcie by mohlo prispieť aj k širšiemu prijatiu platformy, keďže mnohí používatelia už majú účty Google, čo im umožní rýchly a bezproblémový prístup bez potreby ďalšej registrácie.

4 Návrh riešenia

Hlavnou úlohou tejto práce je udržiavať a zlepšovať stav existujúcej platformy, preto je veľmi dôležité aplikovať nové prístupy a nástroje tak, aby sa rozšírila existujúca funkcionalita. V tejto časti bude podrobne opísaná potreba a plán vytvorenej architektúry a spôsob, akým zlepšuje výkonnosť existujúceho riešenia.

4.1 Testovanie

Ako sa podrobnejšie uvádza v časti o analýze projektu, neboli vykonané žiadne testy a v projekte sa neoverilo, či bude fungovať v kritických podmienkach. To vytvára riziko z hľadiska spoľahlivosti a dlhodobej udržateľnosti softvéru. Testovanie je však už neoddeliteľnou súčasťou vývoja, pretože testovanie by malo zachytiť chyby počas vývoja a zaručiť tak správne vykonanie všetkých jednotlivých komponentov a celej aplikácie tak, aby spĺňala požiadavky a očakávania používateľov.

4.1.1 Testovanie činnosti Spring Boot servera

Pri testovaní činnosti backendovej časti sa nezávisle od seba kontroluje činnosť jednotlivých prvkov alebo modulov v rôznych testovacích situáciách vrátane kritických. Na tento účel sa využíva kombinácia dvoch testovacích nástrojov: JUnit a Mockito.

- *JUnit*[1] je osvedčený rámec na písanie a vykonávanie jednotkových testov, ktorý prináša množstvo výhod pre vývojárov. Umožňuje jasnú štruktúru testov, zjednodušuje ich správu a poskytuje širokú škálu funkcií, ako sú testovacie anotácie, výnimky alebo tvrdenia na overenie výsledkov. Automatizované testy písané v JUnit výrazne šetria čas, pretože sa môžu spúšťať pri každej zmene kódu, čím zaručujú rýchle odhalenie a opravu chýb.

[1]Oficiálna webová stránka JUnit: `https://junit.org/junit5/`

- *Mockito*[2] je užitočný nástroj na simuláciu závislostí (mocking), ktorý eliminuje potrebu používania reálnych implementácií počas testovania. To umožňuje testovanie komponentov v izolácii, čím sa zameriava výhradne na ich vlastnú logiku bez rizika nepredvídateľných správaní závislostí. Mockovanie navyše pomáha vytvárať rôzne scenáre, simulovať výpadky alebo výnimočné situácie, čo by bolo s reálnymi komponentmi ťažké alebo nemožné dosiahnuť.

V našom projekte, ktorý je založený na architektúre Model - View - Controller (MVC), sú kľúčové prvky, ktoré sa majú testovať:

- **služby**
- **ovládače**
- **úžitkové triedy**
- **ovládanie chyb**
- **konfigurácie**

Ďalším dobrým prístupom k testovaniu je kontrola vzájomnej komunikácie rôznych komponentov, v našom prípade databázy, kontrolérov a služieb, ktoré sú súčasťou súčasnej architektúry MVC. Integračné testy v prostredí Spring Boot sú zamerané na overenie správneho fungovania aplikácie ako celku, nielen jej jednotlivých častí v izolácii. Tieto testy umožňujú overiť, či komponenty spolu správne komunikujú, či sú dáta správne prenášané medzi vrstvami a či aplikácia reaguje očakávaným spôsobom na rôzne scenáre.

Pri integračnom testovaní použijeme Spring Test framework v kombinácii s JUnit. Testuje sa celý aplikačný kontext, čo umožňuje testovať aplikáciu v prostredí veľmi podobnom produkčnému. Týmto spôsobom je možné overiť nielen logiku jednotlivých komponentov, ale aj ich vzájomnú interakciu a integráciu s externými systémami, ako je napríklad databáza.

4.1.2 Testovanie strany klienta v Angular

Testovanie klientských aplikácií v Angular frameworku je neoddeliteľnou súčasťou zabezpečenia správnej funkčnosti používateľského rozhrania a logiky aplikácie. Angular poskytuje viacero nástrojov a prístupov na efektívne testovanie,

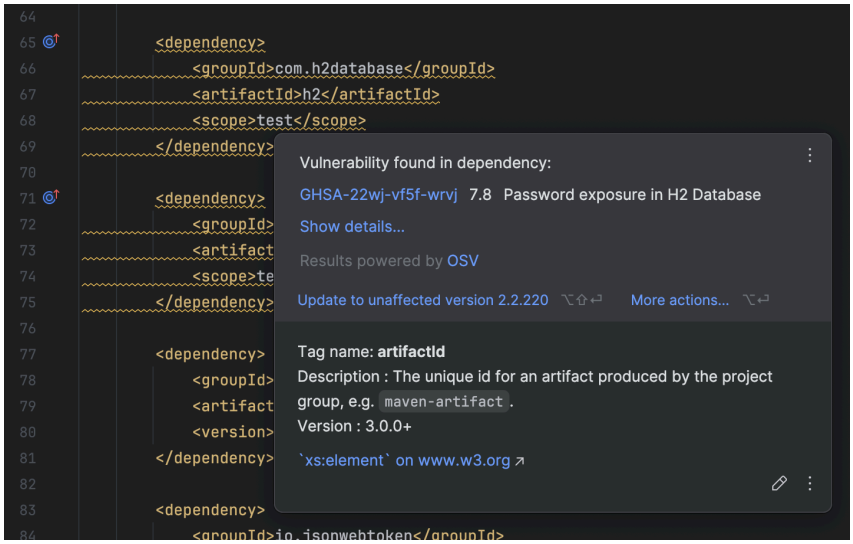
[2]Oficiálna webová stránka Mockito: `https://site.mockito.org`

ktoré zahŕňa testovanie jednotlivých komponentov, služieb a celých používateľských scenárov.

Jednotkové testy v Angulari využívame framework *Jasmine*[3], ktorý je integrovaný do Angular Command Line Interface (CLI) a umožňuje písať detailné testy pre komponenty, služby či direktívy. V kombinácii s knižnicou TestBed, ktorá poskytuje izolované testovacie prostredie simulujúce Angular modul spolu so závislosťami, je možné efektívne overovať správne správanie komponentov a služieb. Tento prístup umožňuje rýchlo a spoľahlivo identifikovať chyby pri rôznych vstupoch a testovacích scenároch.

4.2 Aktualizácia verzií závislostí

Pri úprave nášho projektu sme narazili na problémy súvisiace s nekompatibilitou niektorých verzií importovaných modulov s automaticky aktualizovanou závislosťou Node.js. Keď sme začali pozornejšie kontrolovať verzie závislostí, zistili sme, že mnohé z nich sú zastarané a obsahujú množstvo zraniteľností, ako je tá, ktorá je zobrazená na obrázku 4.1.



Obrázok 4.1: Zobrazenie starých a zraniteľných verzií v prostredí IntelliJ IDEA

V modernom vývoji softvéru je nevyhnutné pravidelne aktualizovať a kontrolovať kompatibilitu aplikácií postavených na Spring Boot a Angular frameworkoch. Tieto technológie sa neustále vyvíjajú a poskytujú nové funkcie, bezpečnostné opravy a výkonové vylepšenia. Pravidelná aktualizácia závislostí je preto

[3]Dokumentácia Angular pre testovanie v Jasmine: `https://angular.dev/guide/testing`

kľúčová nielen pre zabezpečenie stability, ale aj pre minimalizáciu bezpečnostných rizík spojených s používaním zastaraných knižníc.

Pri vývoji aplikácií Spring Boot je veľmi dôležité pravidelne sledovať najnovšie verzie frameworku a súvisiacich závislostí, ako sú Hibernate, Spring Security a databázové ovládače, aby sa predišlo nekompatibilite a zabezpečila sa stabilita aplikácie. Proces aktualizácie by mal zahŕňať podrobnú analýzu zmien uvedených v protokoloch zmien, ktoré poskytujú podrobný prehľad nových funkcií, opráv a potenciálne nebezpečných zmien. Na tento účel je užitočné používať oficiálny *repozitár Maven*[4], kde sú k dispozícii všetky aktuálne verzie spolu s protokolmi modificacii, čo uľahčuje kontrolu súladu a plánovanie aktualizácií.

V prípade Angular aplikácií je rovnako kľúčové sledovať modernizáciu frameworku a pridružených knižníc, napríklad Angular Material či RxJS. Na zabezpečenie obnovenia importovaných modulov v projekte Angular na najnovšie kompatibilné verzie je užitočné použiť príkaz **npm outdated** na zistenie zastaraných balíkov. Potom sa príkazom **npm update** automaticky aktualizujú závislosti na najnovšie kompatibilné verzie, čím sa minimalizuje riziko chýb spôsobených nekompatibilitou. Súčasťou tohto procesu je aj kontrola zmien v dokumentácii aktualizovaných balíkov a testovanie celej aplikácie, aby sa zabezpečilo jej správne fungovanie

4.3 Prepracovanie a opravy kódu

Aj v aktuálnych verziách modulov sa stále nachádzajú metódy označené ako zastarané, čo naznačuje, že ich používanie by malo byť postupne obmedzené a nahradené modernejšími alternatívami. Po vykonaní aktualizácií závislostí sa počet takýchto zastaraných metód ešte zvýšil, čo nás viedlo k rozhodnutiu ich systematicky nahradiť novými prístupmi a funkciami, ktoré sú podporované v aktuálnych verziách modulov.

Proces nahrádzania zastaraných metód sme realizovali dôkladnou analýzou kódu, aby sme identifikovali všetky potenciálne miesta s použitím zastaraných prvkov. Následne sme zaviedli nové funkcie, ktoré reflektujú moderné programátorské prístupy a lepšie podporujú aktuálne technologické štandardy.

Pri písaní a rozširovaní testov sme však narazili na situácie, kde kritické scenáre nefungovali podľa očakávania. To odhalilo slabé miesta v logike aplikácie, ktoré by mohli v produkčnom prostredí viesť k neočakávaným chybám. Na základe týchto zistení sme implementovali nové validácie a pridali výnimky, kto-

[4]Oficiálna webová stránka MVN Repository: `https://mvnrepository.com`

ré zlepšili spoľahlivosť a predvídateľnosť aplikácie. Tento prístup nám umožnil nielen riešiť aktuálne problémy, ale aj pripraviť aplikáciu na lepšiu odolnosť voči budúcim zmenám a výzvam. Zároveň sme posilnili kvalitu kódu zavedením prísnejších testovacích scenárov, ktoré pokrývajú širšie spektrum situácií.

4.4 Obnovenie hesla počas autorizácie

Tento krok sa prelína so zabezpečením autorizácie v bakalárskej práce Dmytra Demianenka [2] a rozširuje jej možnosti. Obnovenie hesla pri prihlásení je kľúčovým prvkom moderných systémov, ktoré používateľom poskytujú pohodlný a bezpečný prístup k ich účtom. Táto funkcia výrazne znižuje problémy spojené so zabudnutými prihlasovacími údajmi a zároveň zlepšuje používateľský komfort a eliminuje zbytočné zaťaženie technickej podpory.

V navrhovanom riešení pre projekt je proces obnovy hesla rozdelený do niekoľkých fáz:

1. Na strane backendu projektu sa pri požiadavke používateľa na obnovenie hesla vygeneruje nový jedinečný token. Tento token sa vytvorí pomocou JSON Web Token (JWT) a bude obsahovať krátku dobu platnosti, napríklad 15 až 30 minút, ako aj zašifrovaný e-mail. Token sa potom uloží do databázy spolu s časovou pečiatkou platnosti, čím sa zabezpečí jeho jednorazové použitie.

2. Ďalším krokom je odoslanie e-mailu na registrovanú e-mailovú adresu používateľa. Odkaz na prepísanie hesla sa vygeneruje pomocou už využitého JavaMail Application Programming Interface (API) a tento odkaz bude obsahovať token ako parameter URL (napr. `/reset-password?token=xyz`). Používateľ by mohol kliknúť na tento odkaz, aby sa dostal na stránku s obnovením hesla.

3. Validácia tokenu a následné nastavenie nového hesla by prebiehalo po otvorení príslušného odkazu. Backend by overil platnosť tokenu, jeho prítomnosť v databáze a zhodu s používateľskými údajmi. V prípade, že by všetky podmienky boli splnené, nové heslo by bolo uložené v databáze po bezpečnom hashovaní a starý token by sa zneplatnil.

4. Na strane klienta, implementovaného v Angular, by riešenie obsahovalo jednoduchý formulár na zadanie e-mailu používateľom. Tento formulár by

odosielal požiadavku na backend, ktorý by spustil proces generovania tokenu. Po prijatí e-mailu by používateľ klikol na odkaz, ktorý by ho presmeroval na stránku resetovania hesla. Táto stránka by obsahovala dve textové polia na zadanie nového hesla a jeho potvrdenie. Token by sa automaticky získal z URL a bol by priložený k požiadavke na backend.

5. Pri návrhu frontendu by sa kládol dôraz na validáciu vstupov priamo na strane klienta, kde by sa kontrolovali požiadavky na minimálnu dĺžku hesla či použitie rôznych typov znakov. Používateľovi by boli zobrazené chybové hlášky v prípade nesprávne zadaných údajov, čím by sa zlepšila jeho skúsenosť.

Riešenie by zahŕňalo aj niekoľko bezpečnostných opatrení. Tokeny by boli časovo obmedzené a jednorazové, aby sa minimalizovalo riziko ich zneužitia. Heslá by nikdy neboli ukladané v podobe obyčajného textu, ale bezpečne hashované. Celková komunikácia medzi klientom a serverom by bola zabezpečená prostredníctvom Hypertext Transfer Protocol Secure (HTTPS) protokolu, čím by sa zaistila ochrana citlivých údajov počas prenosu.

4.5 Využívanie poskytovateľov tretích strán na rýchlejšiu autorizáciu.

Používanie OAuth2 ako mechanizmu na autorizáciu prináša významné výhody, medzi ktoré patrí zvýšená bezpečnosť a pohodlie pre používateľov. Táto technológia umožňuje používateľom autentifikáciu prostredníctvom existujúcich účtov u dôveryhodných poskytovateľov, čím sa eliminuje potreba vytvárať nové heslá a účty, čo zlepšuje ich používateľskú skúsenosť.

Najprv sme si vybyrali jedného poskytovateľa z niekoľkých podľa nášho názoru najpopulárnejších poskytovateľov OAuth2:

- **Google**
- **Microsoft**
- **Facebook**

Rozhodli sme sa pre *Google OAuth2*[5], pretože poskytuje široké možnosti integrácie, je bezplatný a používa ho veľká časť študentov a učiteľov, ktorí už majú

[5]Dokumentácia pre integráciu Google OAuth2: `https://developers.google.com/identity/protocols/oauth2`

svoje účty Google. Tento poskytovateľ navyše ponúka spoľahlivé mechanizmy zabezpečenia, ako je dvojfaktorová autentifikácia, čo zvyšuje dôveryhodnosť celej platformy.

4.5.1 Návrh integrácie Google OAuth2

Na začiatku je potrebné zaregistrovať aplikáciu v Google Developer Console, kde sa získa klientské ID a tajný kľúč aplikácie. Tieto údaje sú nevyhnutné na komunikáciu s API poskytovateľa. Na strane backendu, implementovaného v Spring Boot, sa použije knižnica Spring Security s modulom pre OAuth2, ktorý uľahčuje správu autentifikácie a spracovanie tokenov od Google.

Frontend, bude obsahovať tlačidlo na prihlásenie cez Google. Po kliknutí na toto tlačidlo sa používateľ presmeruje na autentifikačnú stránku Google, kde sa overí jeho totožnosť. Po úspešnom prihlásení Google vráti token, ktorý sa spracuje na strane backendu a použije na overenie alebo vytvorenie nového používateľa v databáze.

Implementácia bude zahŕňať aj bezpečnostné opatrenia, ako sú overovanie platnosti tokenov a ich expiračný čas, aby sa minimalizovalo riziko neoprávneného prístupu. Integrácia bude testovaná pomocou simulovaných autentifikačných scenárov, aby sa zabezpečilo správne fungovanie vo všetkých možných prípadoch. Ako výsledok očakávame, rýchly, bezpečný a pohodlný spôsob prístupu, ktorý prispeje k lepšej akceptácii platformy medzi používateľmi.

5 Testovanie serverovej vrstvy

V tejto kapitole sa pozrieme na to, ako boli uvedené problémy technicky vyriešené v kóde a popíšeme výsledky, ako sa projekt po ich implementácii zlepšil.

5.1 Návrh testovacieho prostredia

V aplikáciách Spring Boot je testovacie prostredie štandardne lokalizované v adresári `test`. Do tohto prostredia boli systematicky začlenené všetky testovacie scenáre, pričom boli rozdelené podľa typu a testovaného objektu. Pri jednotkových testoch, ktoré skúmajú jednotlivé komponenty izolovane, je nevyhnutné zabezpečiť pokrytie všetkých základných prvkov, ako sú služby, kontroléry, pomocné nástroje, konfigurácie a mechanizmy spracovania chýb. Naopak, integračné testy sú zamerané na verifikáciu správnej spolupráce komponentov v rámci architektúry a na validáciu prenosu údajov medzi databázou a aplikáciou. Na základe uvedených kritérií boli testovacie súbory štruktúrované a kategorizované do systému, ako je znázornené vo výpise 5.1.

Výpis 5.1: Štruktúra súborov testovacieho prostredia LirkisEduVePn-service

```
LirkisEduVePn-service
├── documentation
├── src
│   ├── main
│   │   ├── java
│   │   └── resources
│   └── test
│       ├── java
│       │   ├── com.tuke.lirkiseduvepnservice
│       │   │   ├── integration
│       │   │   │   ├── controllers
│       │   │   │   ├── repositories
│       │   │   │   ├── services
```

```
│       │   │   │   └── BaseIntegrationTest
│       │   │   └── unit
│       │   │       ├── config
│       │   │       ├── controllers
│       │   │       ├── mappers
│       │   │       ├── services
│       │   │       └── ErrorHandlerTest
│       │   └── LirkisEduVePnServiceApplicationTests
│       └── resources
│           ├── .testcontainers.properties
│           └── application-test.properties
├── Dockerfile
└── pom.xml
```

Metóda Arrange-Act-Assert (AAA) bola použitá pre jednotkové aj integračné testy, ktorá zdôrazňuje jasnú štruktúru a čitateľnosť testovacieho kódu. Tento prístup delí test na tri časti:

1. **Arrange** (**príprava**), kde sa nastaví testovacie prostredie, inicializujú potrebné objekty a vstupy
2. **Act** (**vykonanie**), kde sa spustí testovaná metóda alebo funkcia
3. **Assert** (**overenie**), kde sa kontrolujú očakávané výstupy voči skutočným výsledkom

Dodržiavanie tohto štandardu jasne štruktúruje operácie v testoch a zjednodušuje ich vnímanie.

5.2 Charakteristiky implementácie

V tejto časti sa budeme venovať hlavným charakteristikám implementácie infraštruktúry na testovanie aplikácií spolu s opisom použitých technológií a metodík.

5.2.1 Jednotkové Testy

Testy si najprv vyžadovali stiahnutie potrebných modulov z oficiálneho repozitára Maven. Nasledujúce moduly sú pridané do `pom.xml`:

- **org.junit.jupiter.junit-jupiter** (verzia 5.11.4) - je základný modul JUnit 5, ktorý poskytuje anotácie, metódy životného cyklu testov a tvrdenia na písanie a vykonávanie jednotkových testov.
- **org.mockito.mockito-core**, (verzia 5.15.2) - poskytuje nástroje na vytváranie simulovaných objektov (mockov) a testovanie interakcií medzi nimi, čo umožňuje efektívne písanie izolovaných jednotkových testov.

Ako sa uvádza v pláne implementácie, na ladenie jednotlivých komponentov sa použil rámec Mockito, ktorý umožňuje vytváranie a simuláciu závislostí prostredníctvom tzv. mock objektov. Tento rámec poskytuje nástroje na izolovanie testovaných tried od ich závislostí, čím sa zabezpečuje, že testy sú zamerané výlučne na konkrétnu funkcionalitu. Mockito zároveň umožňuje kontrolovať interakcie medzi objektmi, overovať volania metód a jednoducho simulovať rôzne scenáre, čo z neho robí výkonný nástroj pre písanie efektívnych a spoľahlivých jednotkových testov.

Na príklade fragmentu kódu testovacej triedy pre UserService vo výpise 5.2 môžete vidieť všeobecný vzor, podľa ktorého boli jednotkové testy vyvinuté:

Výpis 5.2: Šablóna unit testu na príklade triedy UserServiceTest.java.

```
class UserServiceTest {

    @Mock
    private UserMapper userMapper;

    @Mock
    private UserRepository userRepository;

    @Mock
    private JwtService jwtService;

    // iné objekty simulovaných závislosti ...

    @InjectMocks
    private UserService userService; // cielová trieda testu

    // sa vykoná pred kazdou testovacou metódou v tejto triede
    @BeforeEach
    void setUp() {
```

```
        MockitoAnnotations.openMocks(this);
    }

    @Test
    void get_ValidJwt_ReturnsUserProfile() {
        // Arrange
        String jwt = "Bearer valid.jwt.token";
        String email = "test@example.com";

        User user = new User();
        user.setEmail(email);

        UserProfileDto userProfileDto = new UserProfileDto();

        when(jwtService.extractEmail("valid.jwt.token"))
            .thenReturn(email);

        when(userRepository.findByEmail(email))
            .thenReturn(Optional.of(user));

        when(userMapper.daoToDto(user))
            .thenReturn(userProfileDto);

        // Act
        UserProfileDto result = userService.get(jwt);

        // Assert
        assertEquals(userProfileDto, result);
        verify(jwtService).extractEmail("valid.jwt.token");
        verify(userRepository).findByEmail(email);
    }
}
```

5.2.2 Integračné Testy

Integračné testy boli navrhnuté tak, aby overili správnu spoluprácu medzi jednotlivými komponentmi aplikácie, pričom sa zameriavali na simuláciu reálnych podmienok behu aplikácie. Na oddelenie testovacieho prostredia od produkčnej databázy bol použitý nástroj `Testcontainers`, ktorý umožňuje spúšťať izolo-

vané kontajnery pre databázy a iné služby potrebné na testovanie.

Testcontainers je softvérový rámec, ktorý využíva Docker kontajnery na vytváranie čistého a izolovaného testovacieho prostredia.

Do súboru `pom.xml` bolo potrebné pridať nasledujúce závislosti:

- **org.testcontainers.postgresql** (verzia 1.20.4) - je závislosť pre knižnicu Testcontainers, ktorá umožňuje spúšťať izolované PostgreSQL databázové kontajnery na účely testovania
- **org.testcontainers.junit-jupiter**, (verzia 1.20.4) - integruje framework Testcontainers s JUnit 5, umožňujúc jednoduché používanie Docker kontajnerov
- **org.testcontainers.testcontainers**, (verzia 1.20.4) - je základná knižnica pre framework Testcontainers, ktorá poskytuje podporu na spúšťanie a správu Docker kontajnerov v testovacom prostredí
- **org.springframework.boot.spring-boot-testcontainers**, (verzia 3.4.2) - je knižnica, ktorá uľahčuje integráciu Testcontainers s aplikáciami Spring Boot, poskytujúc jednoduchú konfiguráciu a podporu pre testovanie s Docker kontajnermi

Použitie Testcontainers bolo implementované pomocou anotácií a tried na inicializáciu kontajnerov. V konfiguračných súboroch bola zabezpečená dynamická výmena pripojení k databáze tak, aby sa testovacie triedy pripojili k databáze spustenej v kontajneri. V súlade s dobrými praktikami Object Oriented Programming (OOP) bola konfigurácia kontajnera presunutá do rodičovskej triedy a následne všetky triedy s integračnými testami zdedia spoločné vlastnosti. Implementácia triedy `BaseIntegrationTest` popísaná vo výpise 5.3:

Výpis 5.3: Trieda BaseIntegrationTest na konfiguráciu Testcontainers

```
@Testcontainers
@AutoConfigureTestDatabase(replace =
   AutoConfigureTestDatabase.Replace.NONE)
public class BaseIntegrationTest {

    @Container
    @ServiceConnection
    private static final PostgreSQLContainer<?> postgres =
        new PostgreSQLContainer<>("postgres:15.1");
```

```
        @Test
        void connectionEstablished(){
            assertThat(postgres.isCreated()).isTrue();
            assertThat(postgres.isRunning()).isTrue();
        }
}
```

6 Testovanie klientskej vrstvy

Táto kapitola sa venuje praktickému testovaniu a úpravám klientskej strany aplikácie vytvorenej v Angulari. Postup popisuje, ako efektívne zabezpečiť kvalitu kódu a správnu funkcionalitu jednotlivých častí aplikácie.

6.1 Návrh testovacieho prostredia

Pre Angular modul je testovacie prostredie štandardne pripravené vďaka Angular CLI, ktorá poskytuje predkonfigurovanú štruktúru testov. Konfigurácia testovacieho prostredia je rozdelená medzi niekoľko kľúčových súborov:

- **`package.json`** – Tento súbor obsahuje skripty pre spustenie testov (napr. `"test": "ng test"`) a definuje závislosti, medzi ktoré patria aj knižnice jasmine-core, karma, karma-chrome-launcher či karma-coverage. Vďaka Karma frameworku sú testy spúšťané v reálnom prehliadači, čo zaručuje čo najvernejšie simulovanie reálneho prostredia.
- **`angular.json`** – Konfiguračný súbor, ktorý okrem iného definuje ciele pre testovanie, vrátane nastavení kompatibility s rôznymi prehliadačmi, zdrojov, štýlov a ďalších parametrov potrebných pre správnu funkčnosť testovacieho prostredia.
- **`karma.conf.js`** – Tento súbor je špeciálne určený na konfiguráciu testovacieho behu pomocou nástroja Karma, ktorý nielen spúšťa testy, ale aj zabezpečuje detailné zaznamenávanie a výpis výsledkov.

Testy, ktoré sú zvyčajne uložené v súboroch s príponou `.spec.ts`, sa spúšťajú príkazom **ng test**. Tento príkaz automaticky zkompiluje aplikáciu, spustí všetky testy a zobrazí výsledky buď v prehliadači, alebo v príkazovom riadku. Takto integrované riešenie umožňuje konzistentnú a ľahko udržiavateľnú testovaciu infraštruktúru, podobne ako to bolo úspešne realizované pri testovaní serverovej vrstvy.

6.2 Charakteristiky implementácie

Unit testy v Angulari sa riadia osvedčeným princípom Arrange-Act-Assert (AAA), čo zaručuje prehľadnosť a konzistenciu testov. Pri testovaní jednotlivých komponentov, služieb či direktív je kľúčovým nástrojom TestBed. Tento nástroj umožňuje jednoduchú konfiguráciu testovacieho modulu a vytvorenie inštancie testovaného objektu.

Typický testovací súbor (uložený ako `.spec.ts`) môže vyzerať napríklad ako vo výpise 6.1:

Výpis 6.1: Príklad unit testu pre Angular komponent

```
describe('ExampleComponent', () => {
  let component: ExampleComponent;
  let fixture: ComponentFixture<ExampleComponent>;

  // Arrange: Konfigurácia testovacieho modulu a vytvorenie
     instancie komponentu
  beforeEach(async () => {
    await TestBed.configureTestingModule({
      declarations: [ ExampleComponent ],
      providers: [ ExampleService ]
    }).compileComponents();
  });

  beforeEach(() => {
    fixture = TestBed.createComponent(ExampleComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  // Act & Assert: Overenie, e sa komponent úspesne vytvoril
  it('should create', () => {
    expect(component).toBeTruthy();
  });
});
```

V tomto príklade je podobne ako pri testovaní Spring Boot postup testovania Arrange - Act - Assert rozdelený do troch fáz: najprv sa pripraví testovacie prostredie - nakonfiguruje sa testovací modul a vytvorí sa inštancia testovaného

komponentu, potom nasleduje fáza akcie, v ktorej sa komponent skutočne inicializuje, a nakoniec fáza porovnania, v ktorej sa skontroluje, či bol komponent úspešne vytvorený a správne inicializovaný.

Týmto spôsobom zabezpečujeme, že každý test je nielen prehľadný a konzistentný, ale aj ľahko udržiavateľný, čo výrazne prispieva k celkovej kvalite a spoľahlivosti aplikácie.

7 Aktualizácia verzií modulov a zastaraného kódu

Počas spolupráce na projekte sa vývojári stretli s konfliktmi medzi verziami závislostí JavaScript a s nekompatibilitou niektorých modulov. Staršie verzie kódu navyše predstavovali bezpečnostné zraniteľnosti, ktoré boli následne odstránené. V záujme zvýšenia udržateľnosti a bezpečnosti aplikácie preto boli aktualizované závislosti na serverovej strane (Spring Boot) aj na klientskej strane (Angular), vrátane tých, ktoré sa týkajú virtuálnej reality.

7.1 Aktualizácia závislostí servera

Ešte pred aktualizáciou sa v závislostiach vývojového prostredia odrážali nedostatky zdokumentované vývojármi týchto modulov. Vzhľadom na dôležitosť identifikácie týchto nedostatkov sme im venovali samostatnú podsekciu, v ktorej podrobne rozoberáme metodiku ich analýzy a následného odstránenia.

7.1.1 Metodika odstraňovania slabín modulov

Na preštudovanie s následným odstránením sme použili doplnok *OWASP Dependency-Check*[6]. Tento nástroj automaticky identifikuje bezpečnostné zraniteľnosti v knižniciach a komponentoch používaných v projekte. Pracuje na základe porovnania týchto prvkov s verejne dostupnými databázami zraniteľností, ako je napríklad National Vulnerability Database [18], čím poskytuje prehľad o potenciálnych vedomých rizikách. Aby sme mohli používať tento doplnok, najprv sme ho importovali do súboru `pom.xml` z oficiálneho repozitára Maven a pridali kód uvedený vo výpise 7.1:

[6]Oficiálna webová stránka OWASP Dependency-Check: `https://owasp.org/www-project-dependency-check/`

Výpis 7.1: Import nástroja org.owasp.dependency-check-maven.

```
<plugins>
        ...
        <plugin>
          <groupId>org.owasp</groupId>
          <artifactId>dependency-check-maven</artifactId>
          <version>12.1.0</version>
          <executions>
              <execution>
                  <goals>
                      <goal>check</goal>
                  </goals>
              </execution>
          </executions>
        </plugin>
        ...
    </plugins>
```

Potom, po zadaní príkazu **mvn org.owasp:dependency-check-maven:check**, terminál vypísal ďalší zoznam slabých miest modulov, ktorý je znázornený na nasledujúcom obrázku 7.1:

```
[WARNING]

One or more dependencies were identified with known vulnerabilities in LirkisEduVePn-service:

logback-core-1.5.6.jar (pkg:maven/ch.qos.logback/logback-core@1.5.6, cpe:2.3:a:qos:logback:1.5.6:*:*:*:*:*:*:*) :
 CVE-2024-12798, CVE-2024-12801
spring-core-6.1.8.jar (pkg:maven/org.springframework/spring-core@6.1.8, cpe:2.3:a:pivotal_software:spring_framewo
rk:6.1.8:*:*:*:*:*:*:*, cpe:2.3:a:springsource:spring_framework:6.1.8:*:*:*:*:*:*:*, cpe:2.3:a:vmware:spring_fram
ework:6.1.8:*:*:*:*:*:*:*) : CVE-2024-38820
spring-security-config-6.3.0.jar (pkg:maven/org.springframework.security/spring-security-config@6.3.0, cpe:2.3:a:
pivotal_software:spring_security:6.3.0:*:*:*:*:*:*:*, cpe:2.3:a:vmware:spring_security:6.3.0:*:*:*:*:*:*:*) : CVE
-2024-38810
spring-security-web-6.3.0.jar (pkg:maven/org.springframework.security/spring-security-web@6.3.0, cpe:2.3:a:pivota
l_software:spring_security:6.3.0:*:*:*:*:*:*:*, cpe:2.3:a:vmware:spring_security:6.3.0:*:*:*:*:*:*:*, cpe:2.3:a:w
eb_project:web:6.3.0:*:*:*:*:*:*:*) : CVE-2024-38821
spring-web-6.1.8.jar (pkg:maven/org.springframework/spring-web@6.1.8, cpe:2.3:a:pivotal_software:spring_framework
:6.1.8:*:*:*:*:*:*:*, cpe:2.3:a:springsource:spring_framework:6.1.8:*:*:*:*:*:*:*, cpe:2.3:a:vmware:spring_framew
ork:6.1.8:*:*:*:*:*:*:*, cpe:2.3:a:web_project:web:6.1.8:*:*:*:*:*:*:*) : CVE-2024-38809, CVE-2024-38820
spring-webmvc-6.1.8.jar (pkg:maven/org.springframework/spring-webmvc@6.1.8, cpe:2.3:a:pivotal_software:spring_fra
mework:6.1.8:*:*:*:*:*:*:*, cpe:2.3:a:springsource:spring_framework:6.1.8:*:*:*:*:*:*:*, cpe:2.3:a:vmware:spring_
framework:6.1.8:*:*:*:*:*:*:*, cpe:2.3:a:web_project:web:6.1.8:*:*:*:*:*:*:*) : CVE-2024-38816, CVE-2024-38820
swagger-ui-4.18.2.jar: swagger-ui-bundle.js (pkg:javascript/DOMPurify@3.0.1) : CVE-2024-45801, CVE-2024-47875
swagger-ui-4.18.2.jar: swagger-ui-es-bundle.js (pkg:javascript/DOMPurify@3.0.1) : CVE-2024-45801, CVE-2024-47875
```

Obrázok 7.1: Zoznam zraniteľností neaktualizovaných Maven modulov v termináli

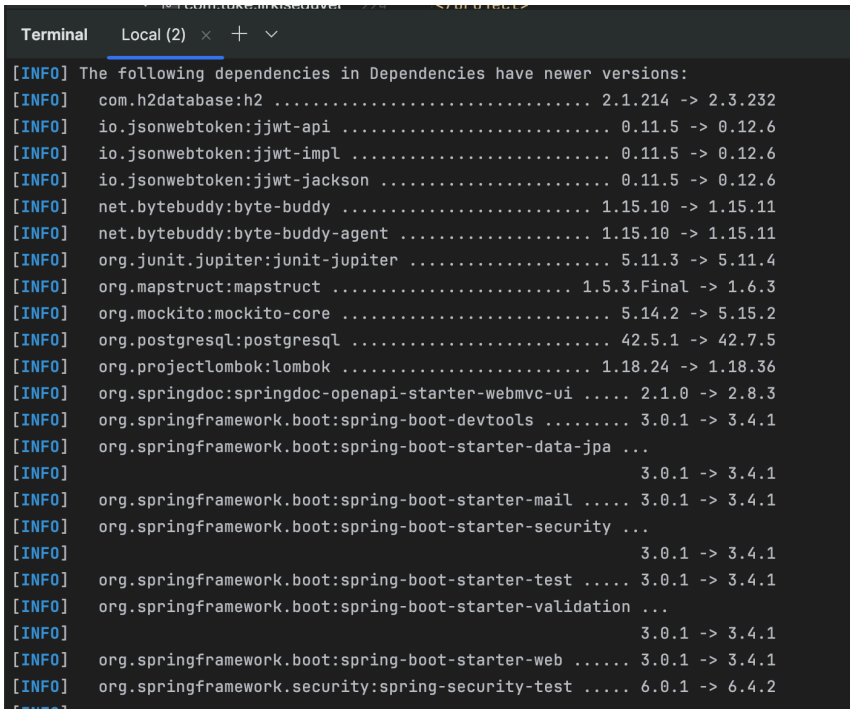
Vďaka znalosti identifikátorov zraniteľností, využitiu oficiálneho repozitára

Maven a zoznamu zmien konkrétnych závislostí sa nám tak podarilo obnoviť všetky slabé miesta použitých modulov. Na tento účel boli pridané aj nasledujúce moduly:

- **ch.qos.logback.logback-core** (verzia 1.5.16) - je hlavná implementácia Logback, spoľahlivého, všeobecného, rýchleho a flexibilného logovacieho rámca.
- **ch.qos.logback.logback-classic** (verzia 1.5.16) - implementácia SLF4J API pre vyššie uvedený Logback.
- **org.apache.commons.commons-compress** (verzia 1.27.1) - API na prácu s kompresnými a archivačnými formátmi.

7.1.2 Aktualizácia konfigurácie Maven a jej členov s ďalšou ich kompatibilitou

V rámci ďalšej údržby a rozvoja projektu sme pristúpili k dôkladnej analýze a modernizácii verzií Maven závislostí, aby sme eliminovali prípadné konflikty a súčasne posilnili bezpečnosť aplikácie. Najskôr sme využili príkaz **mvn versions:display-dependency-updates**, ktorý nám prehľadne zobrazil všetky zastarané knižnice a odporúčané novšie verzie (obrázok 7.2).



Obrázok 7.2: Zoznam zastaralých závislostí a najnovších ich verzií

Týmto krokom sme dokázali získať ucelený prehľad o tom, ktoré moduly vyžadujú okamžitú pozornosť, a mohli sme prioritizovať ich aktualizáciu podľa závažnosti zistených problémov.

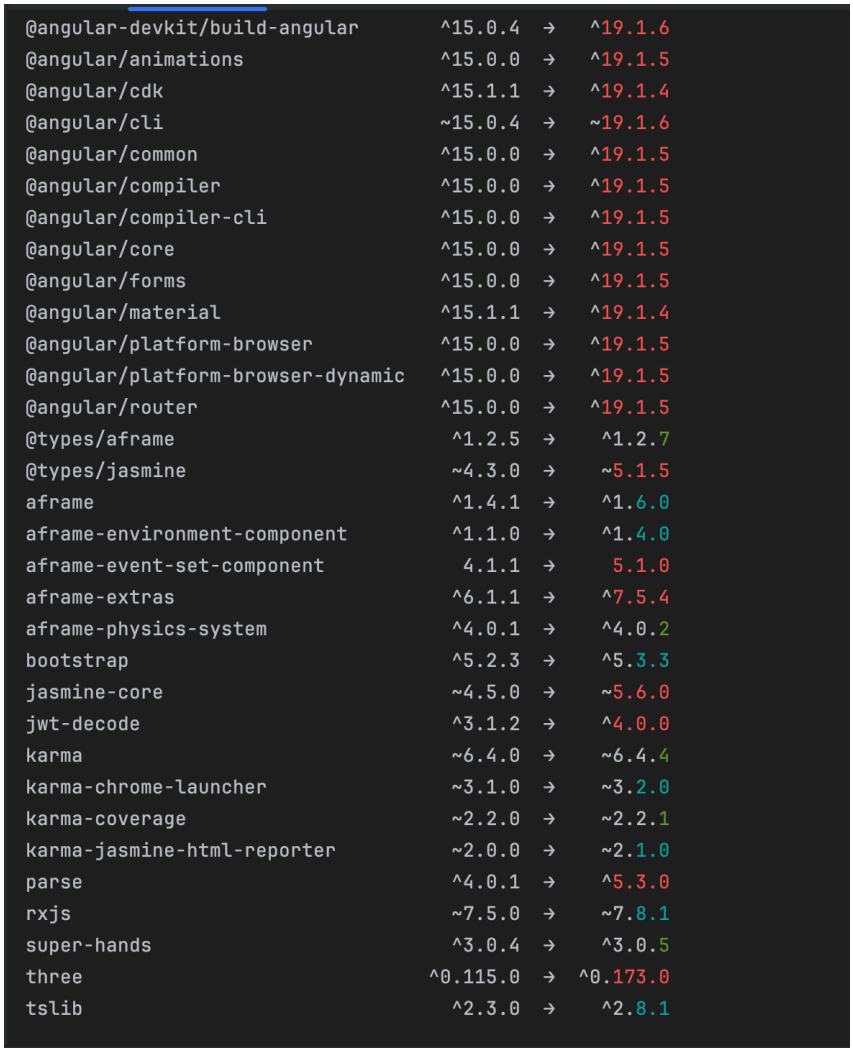
Nasledoval proces systematického prechodu na novšie verzie jednotlivých komponentov, pri ktorom sme priebežne kontrolovali aj bezpečnostné aspekty pomocou už spomínaného nástroja OWASP Dependency-Check. Ten sme spúšťali opakovane, aby sme sa uistili, že novšie verzie knižníc neobsahujú skryté zraniteľnosti alebo iné slabé miesta. Zároveň sme sa opierali o oficiálny repozitár Maven, kde sme dôkladne skúmali zoznam zmien (`changelog`) pre každú knižnicu. Týmto prístupom sme predišli tomu, aby sme síce vyriešili staršie chyby, no nevedomky zaviedli nové.

Po úspešnej modernizácii verzií nastala potreba vykonať zmeny aj v zdrojovom kóde, konkrétne pri práci s JWT. Metódy na generovanie tokenov, ktoré predtým využívali staršie prístupy, boli nahradené novšími atribútmi pre `Jwts.builder()` a `Jwts.parserBuilder()`. Rovnako sme pri prijímaní a validácii `claims` upravili kód tak, aby sa opieral o aktuálnu syntaktickú podobu `Jwts.builder()`, čím sme zachovali konzistentnosť v rámci celej autentifikačnej logiky.

Okrem toho bolo potrebné zrevidovať aj nastavenia bezpečnosti v triedach konfigurácie. Bola aktualizovaná metóda `securityFilterChain(HttpSecurity http)`, aby sme zaistili bezproblémovú kompatibilitu s novšími verziami Spring Boot.

7.2 Modifikacie závislostí používatelského rozhrania

Okrem serverovej časti aplikácie bolo potrebné pristúpiť aj k modernizácii a úprave závislostí v `package.json` pre klientskú stranu, ktorá využíva Angular a knižnicu A-Frame na implementáciu funkcií virtuálnej reality. Na tento účel sme zvolili nástroj `npm-check-updates`, ktorý dokáže analyzovať existujúce verzie balíkov, porovnať ich s najnovšími dostupnými variantmi a automaticky navrhnúť zmeny v súbore `package.json`. Po úvodnej inštalácii pomocou príkazu **npm install -g npm-check-updates** sme spustili príkaz **ncu**, aby sme získali prehľad o dostupných novších verziách všetkých knižníc používaných v projekte (obrazok 7.3).



Obrázok 7.3: Zoznam starých verzií modulov JavaScript a ich dostupných aktualizácií

Vygenerovaný zoznam poskytol jasný obraz o tom, ktoré balíky boli zastarané a aké konkrétne novinky prinášajú ich novšie verzie. V prípade Angular frameworku sme narazili na výrazný posun z Angular 15 na Angular 19, čo prinieslo zásadné zmeny v architektúre, najmä pokiaľ ide o `standalone` komponenty. Tieto komponenty umožňujú definovať a používať komponent bez potreby zahrnutia v tradičných `NgModule`, čo výrazne zjednodušuje modularizáciu kódu a uľahčuje správu závislostí. Avšak vzhľadom na prechod z konvenčného prístupu s modulmi na `standalone` verzie komponentov bolo nevyhnutné pristúpiť k úprave viacerých tried v našom projekte, najmä tých, ktoré sa spoliehali na importy deklarované v `NgModule` súboroch.

Po tom, čo sme potvrdili prechod na novšie verzie knižníc príkazom **ncu -u**, došlo k automatickej aktualizácii `package.json`, a následne sme spustili **npm install**, aby sme nové verzie reálne stiahli a nainštalovali. Nasledovala séria

testov (**npm run build** a **npm run test**), ktoré sme vykonali s cieľom overiť, či všetky moduly spolupracujú v súlade s očakávaniami a nedochádza k žiadnym konfliktom. V rámci týchto testov sme postupne identifikovali viacero nekompatibilít, pričom väčšina z nich súvisela s novou štruktúrou komponentov a prepojením medzi nimi.

V časti projektu venujúcej sa virtuálnej realite sme zaznamenali výrazný nárast verzie aj v prípade knižnice Aframe, ktorá ponúka rozhranie na vytváranie 3D a VR prvkov priamo v HTML. Novšie verzie Aframe priniesli upravené API na manipuláciu s komponentmi. Zmeny sa dotkli hlavne spôsobu, akým sme definovali interaktívne objekty a prepojovali ich so zvyškom aplikácie. Z dôvodu zachovania konzistencie kódu sme museli aktualizovať referencie na staršie Aframe komponenty a prispôsobiť ich moderným funkciám, čo zahŕňalo aj prispôsobenie parametrov niektorých HTML atribútov.

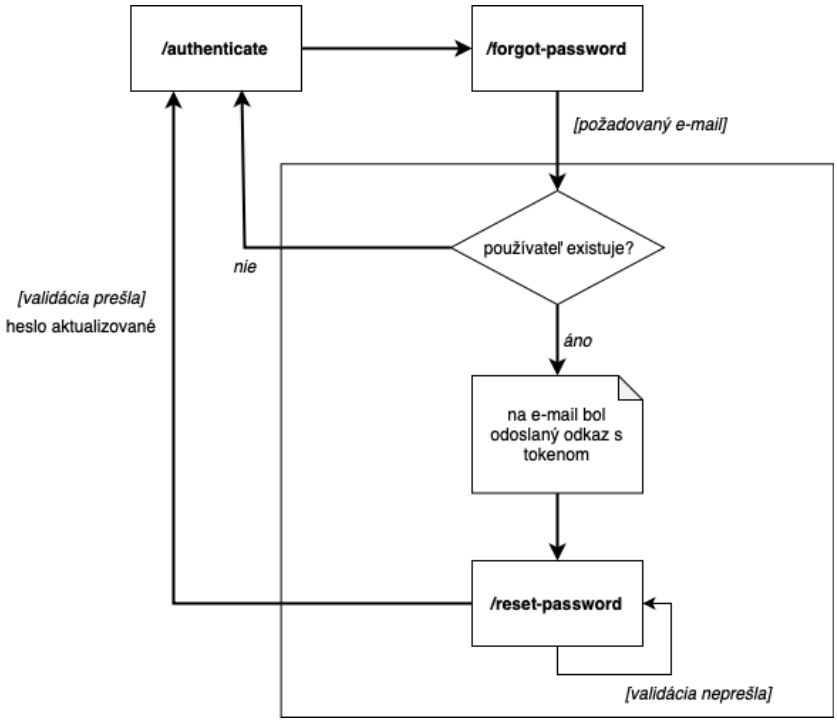
Pri migrácii na novšie verzie Angularu a Aframe bolo nevyhnutné sledovať i vzájomnú kompatibilitu medzi knižnicami, aby sme predišli situácii, kedy by najnovší balík Angularu nebol kompatibilný s verziou Aframe alebo ďalšími závislosťami (napr. `rxjs`, `zone.js` a podobne). V tomto smere nám pomohla priebežná kontrola `changelog`, ako aj sledovanie dokumentácie ku každej knižnici, kde sme sa oboznámili s prípadnými prelomovými zmenami.

Výsledkom uvedeného postupu bola úspešná migrácia na novšié verzie Angularu a Aframe, pričom sme súčasne zefektívnili štruktúru `standalone` komponentov a prispôsobili projekt novým trendom v oblasti vývoja aplikácií pre virtuálnu realitu. Tým sme zabezpečili, že používateľské rozhranie zostane nielen funkčné a prehľadné, ale aj dostatočne flexibilné pre budúce rozširovanie a udržiavanie.

8 Nastavenie nového hesla pri autorizácii

Po napísaní mnohých testovacích scenárov a podrobnej analýze používateľského prostredia sme dospeli k záveru, že systém nemá „záložnú“ možnosť prihlásenia v prípade zabudnutia hesla. Preto táto časť opisuje proces implementácie možnosti obnovenia hesla prostredníctvom potvrdenia e-mailom, ktorá sa používa na mnohých známych platformách, ako sú LinkedIn, Facebook, Coursera a mnohé ďalšie.

V prvom rade sa premyslel samotný proces a jeho fázy, aby sa zabezpečilo, že táto funkcia bude dobre rozdelená na jednotlivé komponenty. Na obrázku 8.1 sú znázornené prechody medzi koncovými bodmi v prehliadači z pohľadu používateľa.



Obrázok 8.1: Prechody medzi medzi koncovými bodmi počas obnovy hesla

Z uvedeného diagramu je už zreteľné, ktoré jednotlivé adresy budú implementované v kontroléri API, ktoré metódy je potrebné pridať do služieb a ktoré jednotlivé stránky je potrebné zobraziť v klientskej vrstve ako komponenty front-endu.

8.1 Modifikácia Spring API pre implementáciu funkcionality obnovy hesla

Podobne ako pri prístupe s potvrdzovacím tokenom pri prvom prihlásení, ktorý bol vyvinutý v bakalárskej práce Dmytra Demianenka [2], bol pridaný špeciálny token `ResetPassword`, ktorý sa rovnakým spôsobom odosiela na e-mail registrovaného používateľa a generuje sa pre každú úspešnú žiadosť o obnovenie prihlasovacích údajov.

8.1.1 Komunikácia s databázou pri spracovaní ResetPassword tokenu

Implementácia možnosti obnovenia hesla vychádza z existujúcej architektúry, kde je kľúčovým prvkom entita `PasswordResetToken` (viď súbor `PasswordResetToken.java`). Táto entita uchováva informácie o vygenerovanom tokenu, čase jeho vytvorenia (`createdAt`), čase expirácie (`expiresAt`) a prípadnom použití (`usedAt`), pričom je prepojená s konkrétnym používateľom prostredníctvom vzťahu definovaného v atribúte `user`. V metóde `forgotPassword` triedy `AuthenticationService` (súbor `AuthenticationService.java`) sa pomocou funkcie `UUID.randomUUID()` generuje jedinečný resetovací token, ktorý je následne zapísaný do databázy prostredníctvom služby `passwordResetTokenService`.

Komunikácia s databázou je podporená aj prostredníctvom repozitára `PasswordResetTokenRepository`, ktorý zabezpečuje efektívne vyhľadávanie, ukladanie a aktualizáciu tokenov, čím prispieva k zachovaniu konzistencie a integrity údajov. Pri overovaní tokenu v metóde `resetPassword` sa kontroluje, či token nevypršal (`expiresAt` je po aktuálnom čase) a či ešte nebol použitý, čo zaručuje bezpečné spracovanie žiadosti o obnovenie hesla.

8.1.2 Funkcie v službách a spracovaníe žiadostí správcom autorizácie

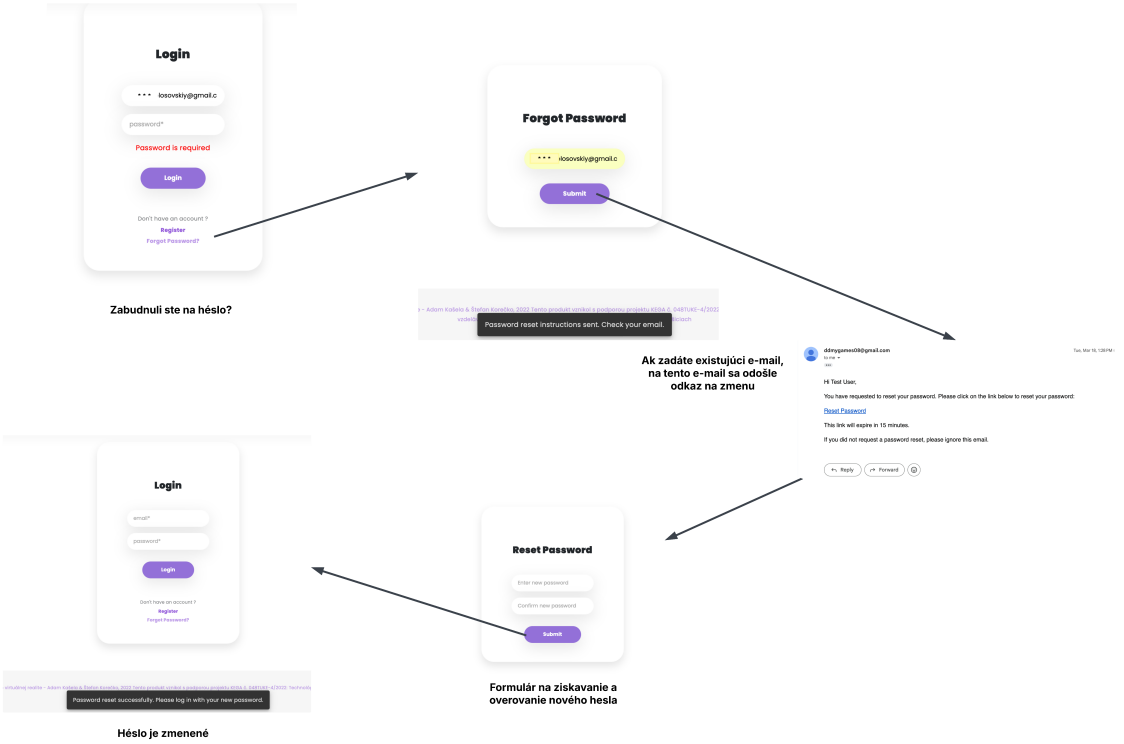
Na úrovni Spring API sú reset hesla a jeho následná aktualizácia implementované prostredníctvom dvoch hlavných Representational State Transfer (REST) koncových bodov, ktoré sú definované v triede `AuthenticationController` (súbor `AuthenticationController.java`).

1. Prvá adresa ***/forgot-password*** prijíma požiadavku v podobe objektu `ForgotPasswordRequest`, obsahujúceho e-mail používateľa. Po overení existencie e-mailu je volaná metóda `forgotPassword` v `AuthenticationService`, ktorá nielenže vytvorí nový `PasswordResetToken`, ale aj zabezpečí jeho uloženie do databázy a následné odoslanie resetovacieho e-mailu cez službu `EmailService` (viď súbor `EmailService.java`).

2. Druhá adresa ***/reset-password*** spracováva požiadavku vo forme `ResetPasswordRequest`, ktorá obsahuje resetovací token a nové heslo. Metóda `resetPassword` následne overí platnosť tokenu, aktualizuje heslo používateľa (s využitím komponentu `PasswordEncoder`) a označí token ako použitý, čím sa zabezpečí, že resetovací token nebude možné opätovne využiť.

8.2 Klientske rozhranie na zmenu prihlasovacích údajov

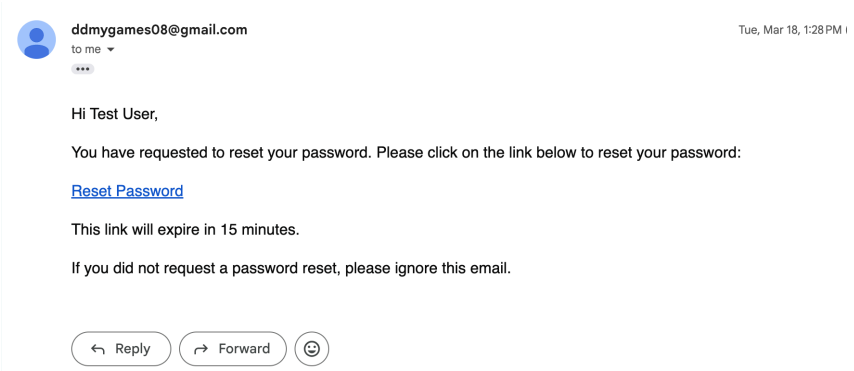
Na základe už vytvorených kontrolérov v našej aplikácii, ktoré sú znázornené v diagrame na obrázku 8.1, bolo potrebné pridať do frontendovej vrstvy len dve komponenty: `forgot-password.component` na vyžiadanie e-mailu používateľa a `reset-password.component` na prijatie, overenie a potvrdenie novej verzie tajného reťazca. Nasledujúci obrázok 8.2 vizuálne znázorňuje interakciu medzi stránkami



Obrázok 8.2: Interakcia medzi stránkami pri zmene hesla

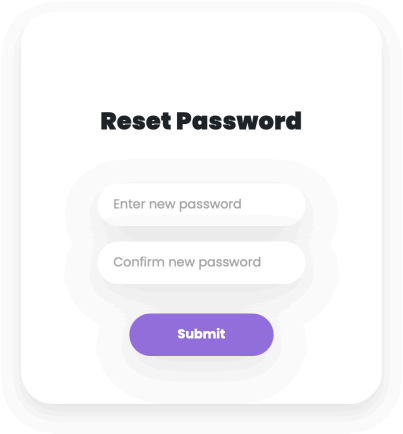
Presnejšie povedané, odkaz ***Zabudli ste heslo?*** presmeruje používateľov na stránku s jediným poľom na zadanie e-mailu. Ak je e-mailová adresa v správnom formáte a existuje v databáze, odošle sa na ňu e-mail s odkazom na ďalší krok a zobrazí sa správa ***„Password reset instructions sent. Check your email.“***, v opačnom prípade sa zobrazí chybové hlásenie.

E-mailové oznámenie odoslané rozhraním API na požadovanú adresu ***/api-/v1/auth/forgot-password*** bude vyzerať podobne ako na ilustrácii 8.3, a bude obsahovať odkaz na ďalšiu stránku vo forme ***https://localhost:4200/reset-password?token=xxx***, kde *xxx* znamená jedinečný resetovací token pre používateľa vygenerovaný a odoslaný zo servera. Je dôležité poznamenať, že platnosť tohto tokenu vyprší po `15 minútach` od jeho generovania.



Obrázok 8.3: E-mailové oznámenie na zmenu hesla

Potom sa používateľ presunie na formulár na zadanie a potvrdenie hesla (viď obrázok 8.4), kde musí byť dlhšie ako 6 znakov a musí sa opakovať rovnako dvakrát, aby sa zabránilo náhodným a neočakávaným kliknutiam.



Obrázok 8.4: Formulár na zadanie a potvrdenie nového hesla

Po zadaní platného tajného reťazca sa požiadavka na zmenu odošle do rozhrania API, kde príslušná služba zmení heslo v databáze a aktualizuje údaje používateľa. Ak je proces správny, zobrazí sa správa ***"Password reset successfully. Please log in with your new password"***.

8.3 Testovanie novovytvorených komponentov

Tak ako pri všetkých moduloch je potrebné otestovať pridané komponenty, aby sa zabezpečilo ich správne zapojenie a funkčnosť. Na tento účel sme pridali testy na strane servera a klienta v súlade so zásadami opísanými v kapitolách 5 a 6.

Na serverovej strane sme implementovali jednotkové aj integračné testy, ktoré sa zameriavali najmä na služby, akými sú napríklad `PasswordResetTokenService`. Jednotkové testy, realizované v súbore `PasswordResetTokenServiceTest.java`, overovali, či metóda `save` správne deleguje ukladanie tokenu do databázy pomocou volania repository, pričom sa kontrolovalo aj správne spracovanie metódy `findByToken`. Konkrétne sa testovalo, že ak je token nájdený, jeho hodnota je vrátená ako typ Optional, alebo ak token neexistuje, je vrátený prázdny objekt. Ďalej boli otestované aj scenáre, kde sa pomocou metódy `setUsed` aktualizuje atribút `usedAt` – ak je token platný, mal by sa tento atribút nastaviť na aktuálny čas, pričom neexistujúci token vyvolal výnimku s hlásením ***„Token not found“***. Integračné testy, prezentované v súbore `PasswordResetTokenServiceIntegrationTest.java`, simulovali reálnu komunikáciu s databázou, kde sa testovalo nielen ukladanie a vyhľadávanie tokenov, ale aj ich následné označenie ako použité, čo odráža reálny tok autentifikácie a resetovania hesla v produkčnom prostredí.

Na strane klienta boli implementované Angular špecifikačné testy, konkrétne v súbore `reset-password.spec.ts`, ktoré detailne kontrolovali funkčnosť komponentu pre resetovanie hesla. Tieto testy simulovali užívateľské interakcie so zadávaním nového hesla a jeho potvrdením, pričom sa overovala správna validácia formulárov – napríklad kontrola zhody hesiel, dodržanie minimálnych bezpečnostných požiadaviek a správne spracovanie chybových hlásení. Testy tiež kontrolovali, či komponent korektne odosiela HTTP požiadavky na server a či sú prijaté odpovede správne interpretované, čo umožnilo aktualizáciu stavu používateľa na prihláseného. Navyše, v rámci týchto testov bolo zabezpečené aj simulovanie oneskorenia (5-sekundovej pauzy po kliknutí na tlačidlo pre potvrdenie), aby sa zabránilo opakovaniu požiadaviek a neúmyselnému odoslaniu viacerých dopytov.

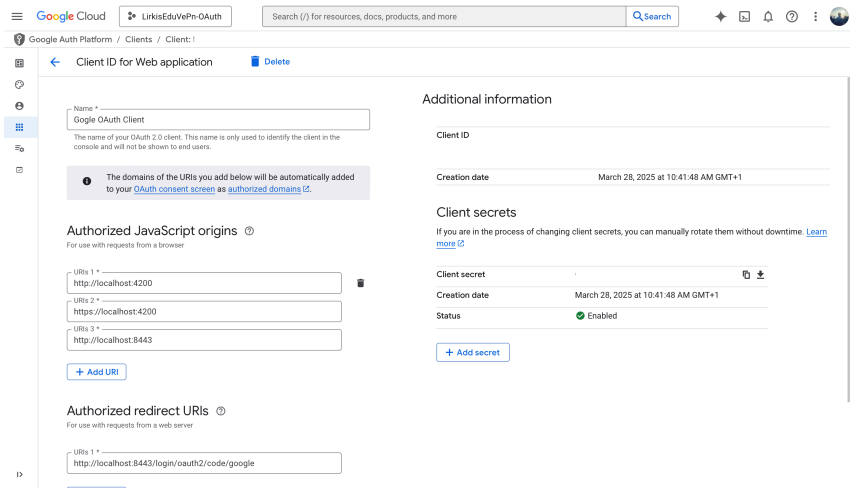
Počas tejto fázy vývoja sa jasne prejavili výhody paralelného písania kódu riadeného testami, pretože bolo oveľa jednoduchšie nájsť logické chyby v predpripravenom kóde a tento prístup zabezpečil, že novo pridaná logika a rôzne zmeny nenarušili správne a očakávané fungovanie existujúcich komponentov.

9 Rýchla a bezpečná autorizácia s Google OAuth2

Ako je uvedené v kapitole 4, Google OAuth2 bol vybraný kvôli jeho dostupnosti, jednoduchej integrácii a rozšírenosti medzi cieľovou skupinou nášho projektu: učiteľmi a študentmi. V tejto kapitole preto opíšeme proces integrácie tejto služby, ako aj možnosti a výhody, ktoré priniesla systému.

9.1 Google Developer Console: prístup a nastavenia klienta

Hlavným cieľom použitia Google Developer Console platformy je vytvoriť bezpečné prostredie, v ktorom sú autentifikačné požiadavky a prístupové práva riadené prostredníctvom overených mechanizmov, čo zvyšuje dôveru používateľov a znižuje riziko neoprávneného prístupu k citlivým údajom. Samotný proces začína vytvorením nového projektu v Google Developer Console, kde je možné centralizovane spravovať všetky priradené API, ktoré sú nevyhnutné pre autentifikáciu – napríklad povolením prístupu k užívateľským údajom v rámci služieb Google. V rámci tohto prostredia sú následne definované podrobnosti o obrazovke súhlasu, kde je používateľovi jasne prezentované, aké údaje budú zhromažďované a ako budú spracovávané, čo zabezpečuje transparentnosť a posilňuje vzájomnú dôveru. Súčasťou tejto konfigurácie je aj vytvorenie klientských poverení, konkrétne klientského identifikátora (Client ID) a tajomného kodu (Client Secret), ktoré predstavujú autentifikačné kľúče pre samotnú komunikáciu medzi serverom a klientom (pozrite si obrázok 9.1, kde citlivé údaje sú skryté).

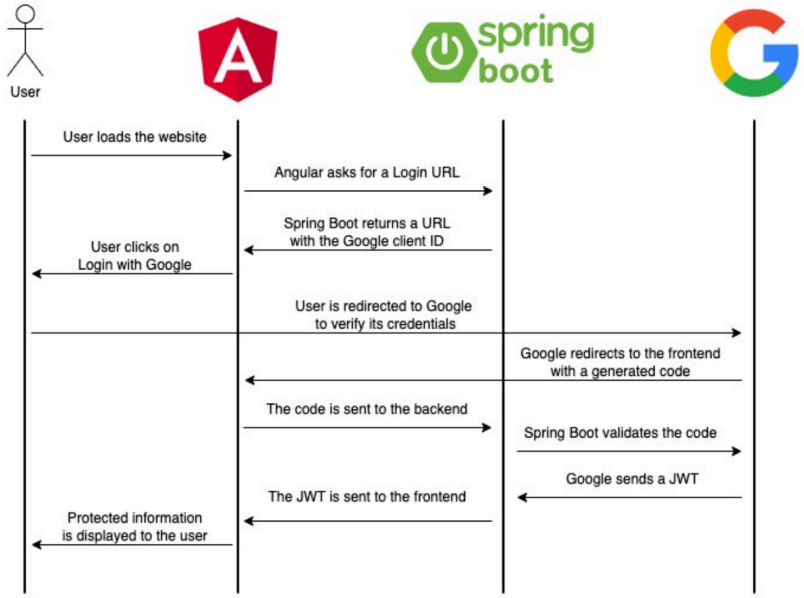


Obrázok 9.1: Panel vývojára a parametre autorizačného klienta

Tieto údaje sa potom integrujú priamo do autorizačného modulu, s pomocou API kodov, vyvinutého na platforme Spring Boot a Angular, čo umožňuje automatizovanú kontrolu prístupových požiadaviek a riadenie prístupu k systémovým zdrojom. Okrem samotného nastavenia poverení poskytuje Google Developer Console aj užitočné nástroje pre monitorovanie prístupov a audítorské funkcie, ktoré umožňujú rýchlo identifikovať a reagovať na možné bezpečnostné incidenty. Tým je zabezpečená nielen ochrana pred aktuálnymi hrozbami, ale aj flexibilná adaptácia na budúce bezpečnostné požiadavky a aktualizácie API. Celý tento systém spája proces konfigurácie s jasným cieľom – vytvoriť bezpečný, prehľadný a riadený autentifikačný rámec, ktorý slúži ako základ pre moderné aplikačné architektúry. Výsledkom je integrácia Google OAuth2, ktorá umožňuje vývojárom sústrediť sa na rozvoj ďalších funkčných a inovatívnych častí aplikácie, pričom je vždy zabezpečená transparentnosť a bezpečnosť používateľských dát.

9.2 Metodika rozvrhnutia služieb a ich logika na každej vrstve

Základný princíp integrácie klienta Google OAuth2 do systému bol dobre znázornený v článku Sergia Lema[19] v nasledujúcom sekvenčnom diagrame na obrázku 9.2:



Obrázok 9.2: Sekvenčný diagram pre autorizáciu s Google OAuth2 [19]

Proces OAuth2 integrácie v našom systéme prebieha v niekoľkých logicky prepojených krokoch. Najprv používateľ klikne na tlačidlo na prihlasovacej stránke, čím spustí autentifikačný proces. Následne je presmerovaný na autorizačnú stránku Google, kde zadá svoje prihlasovacie údaje a využije tak svoj existujúci Google účet bez nutnosti ďalšej registrácie. Po úspešnom overení zo strany Google sú späť odozvané dôležité údaje, ako sú email, meno a priezvisko, ktoré sa ďalej spracovávajú na Spring Boot serveri pomocou služby `OAuthUserService`. Služba najprv spustí metódu `loadUser`, ktorá dedí svoje správanie z `DefaultOAuth2UserService`, pričom overuje, či požiadavka pochádza od Google (na základe atribútu `registrationId`) a či je prítomný email; v prípade jeho absencie je vyvolaná výnimka. Ak daný používateľ ešte neexistuje v databáze, systém automaticky vytvorí nový záznam s údajmi z Google (ako sú meno, priezvisko a email). Nakoniec služba `JwtService`, vytvorí presmerovanie URL, do ktorej tento token pripojí, a presmeruje používateľa na túto adresu. Týmto spôsobom Angular klient získa token potrebný na ďalšie autorizované požiadavky.

9.2.1 Podrobný rozbor kľúčových komponentov

Pozrime sa bližšie na hlavné triedy, ktoré tvoria backend časť integrácie:

- **OAuthUserService.java:**
  Táto trieda rozširuje `DefaultOAuth2UserService` a jej hlavným cieľom

je:

  - Načítať údaje používateľa získané od Google.
  - Overiť, či požiadavka pochádza od Google prostredníctvom atribútu `registrationId`.
  - Získať email z atribútov a vyhodiť chybu, ak email nie je k dispozícii.
  - Zaregistrovať nového používateľa v databáze, ak ešte neexistuje, alebo načítať existujúce údaje.

- **OAuthSuccessHandler.java:**
  Po úspešnej autentifikácii táto trieda zabezpečuje:
  - Extrakciu používateľských údajov z objektu `OAuth2User`.
  - Overenie existencie používateľa v databáze, alebo vytvorenie nového záznamu.
  - Generovanie JWT tokenu cez `JwtService`.
  - Tvorbu presmerovanie URL, do ktorej je pripojený token, a následné presmerovanie používateľa späť na klientsku aplikáciu.

- **SecurityConfig.java:**
  Táto konfigurácia je základom zabezpečenia celej aplikácie a obsahuje:
  - Cross-origin Resource Sharing (CORS) konfiguráciu – povolené sú konkrétne pôvody, HTTP metódy a hlavičky.
  - Definíciu verejných a chránených koncových bodov (napríklad dokumentácia API je prístupná bez autentifikácie).
  - Integráciu OAuth2 login s využitím `OAuthUserService` a `OAuthSuccessHandler`.
  - Nastavenie bezstavového riadenia sedení, čím sa zabezpečí, že každý dopyt je autentifikovaný pomocou JWT tokenu.

9.3 Kontrola funkčnosti novej funkcionality pomocou testov

V testovacom prostredí pre implementáciu Google OAuth2 autorizácie bolo vytvorených niekoľko úrovní testovania, ktoré zabezpečujú správnu integráciu a funkčnosť celého systému. Na backendovej strane sa využívali jednotkové aj integračné

testy, ktoré sa zameriavali najmä na služby `OAuthUserService` a `OAuthSuccessHandler`. Testy triedy `OAuthUserService` overovali správnosť metódy `loadUser`, ktorá je zodpovedná za načítanie údajov používateľa získaných od Google, validáciu povinných atribútov, ako je email, a správne registrovanie nových používateľov do databázy. V prípade, že atribút email chýbal, testy kontrolovali vyvolanie príslušnej výnimky, čím sa zabezpečila integrita dát a bezpečnostné opatrenia na strane servera. Integrácia s databázou bola testovaná simuláciou existujúcich aj nových používateľov pomocou knižnice Mockito, pričom sa overovalo, že nedochádza k duplicitnému ukladaniu a že sú správne volané metódy repository. Ďalšia skupina testov, implementovaná v rámci tried `OAuthSuccessHandlerTest` a `OAuthSuccessHandlerIntegrationTest`, simulovala celý autentifikačný proces, od overenia používateľa až po generovanie JWT tokenu pomocou služby `JwtService` a následné presmerovanie používateľa na klientskú aplikáciu. Tieto testy zároveň overovali správne spracovanie chybových stavov, ako je neúspešné presmerovanie, ktoré môže vyvolať `IOException`. Na frontendovej strane boli implementované špecifikačné testy v súboroch `auth-callback.component.spec.ts` a `login.component.spec.ts`, ktoré zaisťovali, že Angular aplikácia korektne spúšťa proces OAuth2, prijíma token a následne aktualizuje stav používateľa na prihláseného. Tieto testy tiež kontrolovali, či sa vo správnom poradí vykonávajú volania REST API a či komponenty správne reagujú na prípadné chyby.

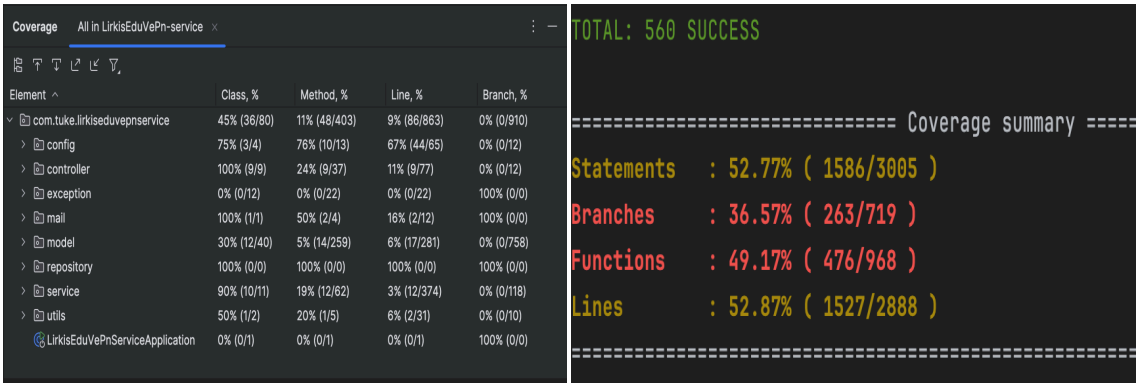
Celkový testovací prístup zabezpečil nielen funkčnosť jednotlivých komponentov, ale aj ich vzájomnú komunikáciu a integritu celého autentifikačného mechanizmu. Týmto spôsobom bolo možné identifikovať aj prípadné slabé miesta v procese autentifikácie a zabezpečiť, že prihlasovánie bude fungovať spoľahlivo aj v produkčnom prostredí.

10 Vyhodnotenie práce

Hlavným cieľom tejto bakalárskej práce bolo rozsiahle otestovať a vylepšiť vytvorené virtuálne vzdelávacie prostredie, aby sa dosiahla jeho udržateľnosť a spoľahlivosť. Výsledkom bolo vytvorenie komplexného testovacieho prostredia, ktoré pomohlo odstrániť niektoré slabé miesta a identifikovať možnosti pre ďalší vývoj. V nasledujúcich častiach sú zhrnuté vykonané zlepšenia, ich prínosy a navrhované vylepšenia.

10.1 Zhodnotenie kľúčových výsledkov pri vývoji testovacieho prostredia

Po implementácii testov sa pokrytie kódu zvýšilo **z pôvodných 9% na 97%** (porovnanie na obrázkoch 10.1a a 10.1b) zo všetkých riadkov kódu na serveri a **z pôvodných 0% na 52.87%** na úrovni webového klientskeho a virtuálneho prostredia (porovnanie na obrázkoch 10.2a a 10.2b). Tento významný nárast nielenže výrazne znižuje pravdepodobnosť výskytu nepredvídaných chýb v budúcnosti, ale zároveň demonštruje oveľa vyššiu spoľahlivosť a kvalitu celého riešenia. Týmto spôsobom sme dosiahli spoľahlivejši kód, ktorý je lepšie pripravený na zvládanie rôznych scenárov a potenciálnych problémov, čím sa zvyšuje dôvera v dlhodobú udržateľnosť a stabilitu aplikácie.



(a) pred písaním testov (b) po napísaní testov

Obrázok 10.1: Zvýšenie pokrytia kódu servera testami

(a) pred písaním testov (b) po napísaní testov

Obrázok 10.2: Zvýšenie pokrytia kódu klienta testami

10.1.1 Prínosy testovania

Ako sa uvádza v knihe „The Art of Software Testing“ od Glenforda J. Myersa, Coreyho Sandlera a Toma Badgetta [6], testovací prípad, ktorý odhalí chybu v systéme, je hodnotnejšou investíciou ako testovací prípad, ktorý len potvrdzuje, že systém neobsahuje chyby. V súlade s touto myšlienkou boli objavené viaceré nedostatky v logike servera, uvedené v tabuľke 10.1, ktoré obmedzovali jeho efektívnosť a mohli by v budúcnosti viesť k závažnejším problémom.

Tabuľka 10.1: Prehľad testov, nájdených problémov a riešení

Test	**Problém**	**Riešenie**
update_User-NotFound() v UserServiceTest	Funkcia `update()` aktualizuje údaje, aj keď používateľský objekt, ktorý sa má zmeniť, neexistuje v databáze.	Pridanie validácie pred zmenou údajov používateľa a vyvolanie výnimky v prípade nesprávnych údajov.
update_NullFields() v UserServiceTest	Polia používateľa môžu byť aktualizované na hodnotu `null`.	Validácia nových údajov pred aktualizáciou používateľských polí pomocou triedy `Optional`.
ConfirmationToken-ServiceTest	`confirmationToken` môže byť vygenerovaný aj pre nedefinovaného (`null`) používateľa.	Pred generovaním tokenu overiť platnosť používateľa a vyvolať výnimku v prípade neplatných údajov.
GroupServiceTest	Aktualizácia skupiny s prázdnym zoznamom úloh spôsobovala neočakávané chyby.	Pridanie validácie a správne spracovanie prázdnych zoznamov úloh.
ImageResizerTest	Funkcia `resizeImage()` nespracúva správne neplatné obrazové údaje.	Skontrolovať obrazové údaje pri ich načítaní a vyvolať výnimku pri chybných údajoch.
send_Null-Recipient() v EmailServiceTest	Funkcia `send()` operuje, aj keď premenná `recipient` je `null` alebo prázdna.	Skontrolovať, či premenná `recipient` nie je prázdna a či je definovaná na začiatku funkcie `send()`.
send_EmptyLink() v EmailServiceTest	Funkcia `send()` operuje, aj keď premenná `link` je `null` alebo prázdna.	Overiť, či je premenná `link` správne inicializovaná pred pokračovaním operácie.

Pokračovanie tabuľky

Test	Problém	Riešenie
validate_NullFilename() v PhotoExtractorTest	Funkcia `validatePhotoFile()` akceptuje neplatné názvy súborov.	Skontrolovať, či je názov súboru zadaný a vyvolať výnimku v prípade chýbajúcich údajov.
validate_UpperCase() v PhotoExtractorTest	Funkcia `validatePhotoFile()` neakceptuje prípony súborov s veľkými písmenami (napr. `.JPG`).	Pred overovaním konvertovať prípony na malé písmená.
TaskSessionService-IntegrationTest	Ak je sedenie úlohy už dokončené, keď sa ju znova pokúšate dokončiť, vyhodí sa všeobecná výnimka `RuntimeException` bez náležitého spracovania.	Mala by sa zaviesť špeciálna výnimka `TaskSession-AlreadyFinished-Exception` a podľa toho odoslať službe `TaskSessionService`, odoslaním príslušného stavu `BAD_REQUEST` do kontroléra.
FiringAttempt-Repository-IntegrationTest	Repozitár pokusov v metóde `findBySessionId-AndTrue` vráti všetky pokusy, nielen tie úspešné, ako by sa očakávalo.	Pridať dodatočnú kontrolu `f.successful = true` do dotazu na databázu pre metódu `findBySessionId-AndTrue` v repozitári pokusov.

Na strane klienta bolo opravených niekoľko typových chýb, napríklad **„TypeError: Cannot read properties of undefined“** v `task-files.model.ts` a **„TypeError: transitions.forEach is not a function“** v `cpnLoader.mjs` a `petriNetLoader.mjs`. Tieto chyby neboli zrejmé pri písaní kódu JavaScriptu kvôli voľnému typovaniu tohto jazyka a neviedli k poruchám v štandardných prípadoch použitia, ale mohli viesť k poruchám systému pri neočakávaných spôsoboch jeho použitia.

Pri testovaní systému s kolegom sme narazili na ďalší problém s nesprávnym

prepojením medzi reláciami úloh a používateľmi, ktorí ich začali alebo ukončili. Táto chyba umožňovala študentovi ukončiť reláciu, ktorá nebola jeho, a pokračovať v relácii niekoho iného, pričom získal celý pokrok druhej osoby. Riešením tohto problému bolo pridanie ďalšieho parametra na vyhľadávanie sedenias v databáze, a to ID používateľa. Pri prijímaní údajov z úložiska sa teda porovnávajú ID používateľa, ktorému je sessia priradená, a používateľa, ktorý bol v tom čase prihlásený.

Preto by na zníženie výskytu takýchto chýb v budúcnosti bolo dobré prepísať skripty pre úlohy vo virtuálnom prostredí do prísnejšie typovaného TypeScriptu, ktorý by zachoval rovnakú štruktúru kódu na celej vrstve klienta s jedným programovacím jazykom, dodatočným spracovaním typových chýb a podporil by údržbu systému ako celku.

Hoci sa v kóde našlo pomerne málo závažných chýb, testovanie prinieslo značné výhody a v budúcnosti ich bude ešte viac. Malý počet nájdených chýb poukazuje na dobrú kvalitu pôvodne napísaného kódu, ktorý opísali Adam Kašela a Dmytro Demianenko vo svojich prácach, no zároveň zdôrazňuje potrebu systematického prístupu k testovaniu na zachovanie a ďalšie zlepšenie stability systému. Aj pri nízkom počte zistených problémov sú testy kľúčové nielen na odhalenie existujúcich nedostatkov, ale aj na prevenciu budúcich komplikácií, ktoré sa môžu objaviť pri rozširovaní alebo modifikácii systému.

10.2 Rozšírené možnosti autorizácie

Na vyhodnotenie účinnosti a použiteľnosti pridaných funkcií prihlásenia bolo naživo a online vyspovedaných 20 ľudí vo veku 20 - 25 rokov, ktorí individuálne a v skupinách do 5 osôb skúšali prihlásenie, registráciu a novo pridané funkcie obnovenia hesla a prihlásenia cez Google. Každý z nich bol požiadaný, aby sa prihlásil alebo vyskúšal jednu funkciu bez podrobných pokynov, aby sa zabezpečilo, že prihlasovanie bude pre používateľov jednoduché a intuitívne. Potom každý z nich vyplnil *dotazník vo formulári Google*[7], v ktorom opísal svoje skúsenosti. Analýza prieskumu a spätná väzba od respondentov sú podrobnejšie opísané v nasledujúcich častiach.

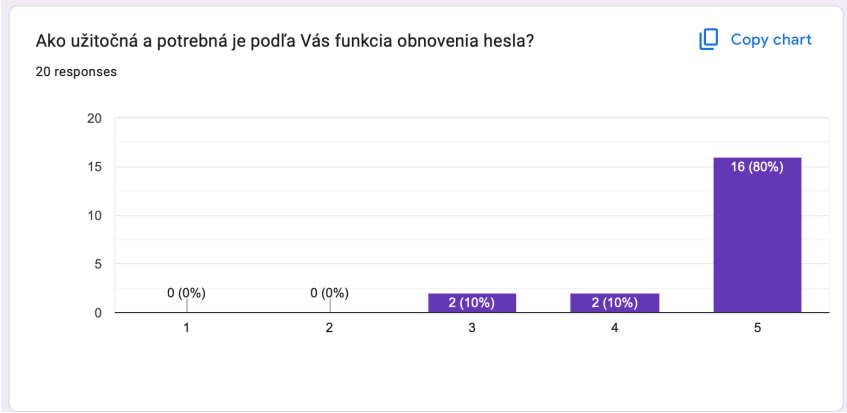
[7]Dotazník je k dispozícii na adrese: `https://forms.gle/xTaz6QqQjaD4onNJ8`

10.2.1 Obnovenie hesla

Nasledujúce otázky boli položené s cieľom posúdiť pohodlie a užitočnosť funkcie obnovy hesla po tom, ako ju používatelia otestovali na lokálnom počítači:

- Ako užitočná a potrebná je podľa Vás funkcia obnovenia hesla?
- Ako dlho Vám trvalo získať odkaz na obnovenie hesla na Váš email?
- Do akej miery bol proces obnovy hesla jednoduchý?
- Ak by ste mohly zmeniť jednu vec na stránke pre reset hesla, čo by to bolo?

Na základe analýzy odpovedí, 16 osôb ohodnotilo dôležitosť obnovy hesla na 5 z 5, čo je **80% všetkých respondentov**, 2 ľudia ohodnotili rovnaké kritérium na 4 z 5 a ďalší 2 zvolili priemernú možnosť 3 z 5 (viď obrázok 10.3). Podľa týchto údajov je priemerné hodnotenie významu respondentmi **4,7**.



Obrázok 10.3: Analýza názorov používateľov na dôležitosť obnovy hesla

V **75% pokusov** o autorizáciu sa odkaz na nastavenie nového hesla odoslal za menej ako minútu a v ostatných prípadoch to netrvalo dlhšie ako 5 minút. Táto nejednoznačnosť bola spôsobená rôznou kvalitou internetového pripojenia, keďže prieskum sa vykonával na rôznych miestach.

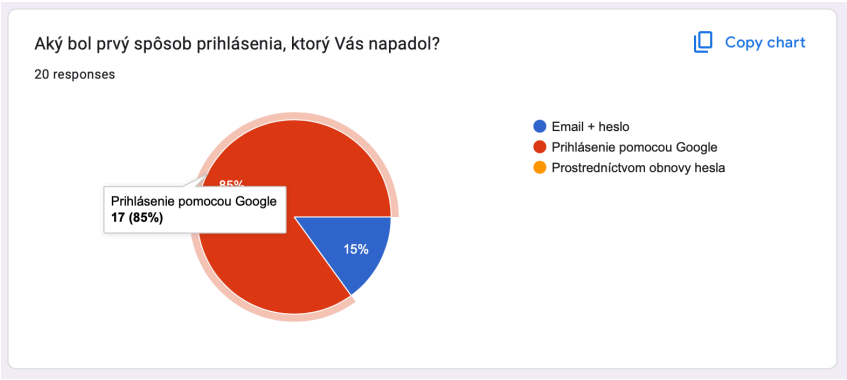
Medzi respondentmi bolo niekoľko ľudí so vzdelaním v odboroch informatiky a kybernetickej bezpečnosti. Niektorí z nich prišli so zaujímavým nápadom pridať indikátor sily novovytvoreného hesla (obrázok 10.4), aby sa zvýšila bezpečnosť prihlasovacích údajov používateľov a znížilo riziko ich uhádnutia neoprávnenými osobami.

Obrázok 10.4: Návrhy používateľov na zlepšenie metódy obnovy hesla

Ak to zhrnieme, žiadny z opýtaných používateľov nemal s touto funkciou žiadne ťažkosti, takže je možné ju považovať za jednoduchú a potrebnú na používanie.

10.2.2 Autorizacia s pomocou Google

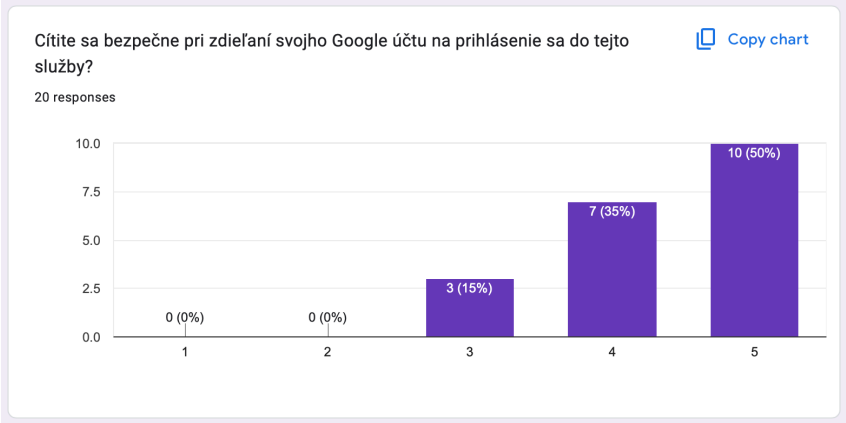
V prvom rade by som chcel poznamenať, že **85% respondentov** si ako prvý a najpohodlnejší spôsob prihlásenia zvolilo Google (obrázok 10.5), ostatní sa prihlásili pomocou e-mailu a hesla.



Obrázok 10.5: Analýza spôsobu prihlásenia, ktorý si používatelia vybrali ako prvý

To naznačuje, že prihlasovanie pomocou dôveryhodných služieb tretích strán (v našom prípade Google) je pre väčšinu používateľov známe a jednoduché, čo celkovo spríjemní proces autorizácie.

Na druhej strane nie všetci používatelia plne dôverovali prenosu svojich údajov do inej organizácie (pozri obrázok 10.6)



Obrázok 10.6: Ukazovatele dôvery používateľov pri odosielaní údajov spoločnosti Google

Hoci nikto nebol úplne proti tomuto spôsobu prihlasovania, len **50% ľudí** sa cítilo úplne bezpečne pri zdieľaní svojich údajov so službou tretej strany, ostatní trochu váhali.

Vo výsledku boli používatelia so systémom autorizácie spokojní, ľahko sa používal a mali na výber z viacerých možností.

10.3 Návrhy na ďalšie zlepšenia

V procese vývoja projektu zostali niektoré nevyriešené nedostatky a našli sa nové oblasti na ďalšie zlepšenie:

- **Zvážiť iné alternatívy k Aframe pre VR prostredie**: vzhľadom na problémy, ktoré sa objavili v priebehu tejto práce a v priebehu práce mojich kolegov, by bolo dobré zvážiť prepísanie virtuálneho prostredia na iné rámce. V štúdii sa by mohli analyzovať iné alternatívy s pravidelnými aktualizáciami a podporou, lepšou synchronizáciou so systémom Angular a širšími možnosťami testovania.

- **Dosiahnutie 100-percentného pokrytia kódu pomocou nových testovacích techník**: komplexnosť projektu a časové obmedzenia nám umožnili použiť len jednotkové a funkčné testovanie. Nebol však otestovaný každý riadok kódu a neboli použité všetky dostupné techniky.

- **Prepis modulov scén virtuálnej reality do jazyka TypeScript**: systém by sa ľahšie udržiaval, keby bola celá architektúra front-endu napísaná v jednom programovacom jazyku. Okrem toho prísne typovanie v jazyku

TypeScript pridá viac typovej kontroly a zníži počet skrytých chýb v JavaScript.

- **Ďalší vývoj projektu podľa princípu Test Driven Development (TDD)**: TDD metodika predpokladá, že testovacie scenáre sa pridávajú s vývojom novej funkcionality, aby sa zohľadnili všetky aspekty naraz a aby sa táto funkcionalita pokryla testami čo najefektívnejším spôsobom. V tejto práci bolo testovacie prostredie pridané až po naprogramovanej logike, ale ak ich ďalší vývojári pridajú paralelne, dosiahne sa efektivita testovania a častejšia detekcia chýb.

- **Pridanie interakcií medzi študentmi a učiteľmi na virtuálnych scénach**: v súčasnosti platforma neumožňuje žiakom vzájomnú interakciu počas dokončovania úlohy. Možno by bolo dobré pridať do niektorých scén možnosť pre viacerých hráčov a umožniť tímovú prácu na niektorých úlohách pridaním komunikačných kanálov, ako je chat alebo hlas.

- **Kontrola a zobrazenie pevnosti nového hesla počas registrácie a obnovy hesla**: dobre by bolo pridať vizuálny indikátor sily nového hesla. Táto funkcia by užívateľom okamžite ukázala, aké silné je heslo, ktoré zadali (napríklad pomocou farebného prúžku a krátkeho textu, ako napríklad „slabé“, „stredné“ a „silné“). Zvýšila by sa tým bezpečnosť autorizácie.

11 Záver

V rámci tejto práce sme pokračovali v rozpracovaní projektu, na ktorom predtým pracovali Ing. Adam Kašela, Bc. Dmytro Demianenko, Ing. Juraj Rudy a Bc. Lukáš Pisarčik. Práca bola zameraná na testovanie a vylepšovanie už vyvinutého virtuálneho výučbového prostredia, odstraňovanie chýb a zraniteľností, zabezpečenie väčšej podpory systému v budúcnosti a rozšírenie používateľského pohodlia. Vyvinuli sme komplexné testovacie prostredie pre server, webového klienta a čiastočne virtuálne prostredie, aktualizovali a vylepšili kód, rozšírili možnosti autorizácie a našli nové oblasti pre ďalší výskum.

Najskôr sa podrobne preskúmalo vytvorené učebné prostredie a zistili sa prvé nedostatky. Potom sme analyzovali existujúce architektonické koncepcie, nástroje a metodiky testovania, ktoré nám pomohli pri vývoji testovacieho prostredia a vylepšení architektúry projektu. Testovaním sa tiež odstránili viaceré nedostatky a slabiny systému.

V priebehu tejto práce boli aktualizované viacere závislosti používané v prostredí Spring backend a Angular. V rámci toho sme odstránili nedostatky externých modulov, na ktoré ich vývojári upozornili v dokumentácii zastaraných verzií. Táto aktualizácia priniesla niekoľko výhod - najmä zvýšila bezpečnosť systému odstránením známych zraniteľností, najmä v oblasti šifrovania a autorizačných mechanizmov.

Pri spolupráce s ostatnými členmi vývojového tímu projektu sa tiež rozhodlo o nahradení modula zodpovedného za fyziku virtuálneho prostredia, čím sa kolegom otvorili nové možnosti modelovania virtuálnych scén. Nový modul poskytuje presnejšiu fyzikálnu simuláciu a lepšiu integráciu s modernými technológiami. Pri aktualizácie modulov vzniklo najviac komplikácií s kompatibilitou modulov a komponentov Aframe, pretože nie všetky sú pravidelne aktualizované a kompatibilné s novou syntaxou a technológiami Angular. Napriek tomu tieto zmeny vytvárajú pevnejší základ pre budúci vývoj a umožňujú tímu efektívnejšie reagovať na nové požiadavky projektu.

V rámci zlepšenia používateľského prostredia bola pridaná možnosť obnoviť

heslo prostredníctvom e-mailu a prihlásiť sa pomocou služby tretej strany Google. Tieto nové funkcie sa ukázali ako pohodlné a užitočné počas prieskumu medzi používateľmi, v ktorom sa zisťovali ich názory na túto záležitosť.

Hlavným cieľom tejto práce bolo zlepšiť virtuálne vzdelávacie prostredie prípravou testovacieho prostredia a optimalizáciou projektu pre ďalší vývoj. Celkovo bola práca úspešná a bola prínosom pre existujúci projekt, hoci v budúcnosti je ešte priestor na ďalšie vylepšenia systému.

Zoznam použitej literatúry

1. KAŠELA, Adam. *Výučbové scenáre v rozšírenej realite využívajúce procesné grafy*. Košice, 2022. Dipl. pr. Technická univerzita v Košiciach. Vedúci práce: Ing. Štefan Korečko, PhD.
2. DEMIANENKO, Dmytro. *Spracovanie údajov z virtuálnych výučbových prostredí*. Košice, 2023. Vedúci práce: Ing. Štefan Korečko, PhD.
3. PISARČÍK, Lukáš. *Scenárom riadené výučbové prostredia vo virtuálnej realite*. Košice, 2023. Vedúci práce: Ing. Štefan Korečko, PhD.
4. RUDY, Ing. Juraj. *Systém virtuálnych výučbových prostredí riadených procesnými grafmi*. Košice, 2024. Dipl. pr. Technická univerzita v Košiciach. Vedúci práce: Ing. Štefan Korečko, PhD.
5. CONTRIBUTORS, MDN. *What is a web server?* [B.r.]. Dostupné tiež z: `https://developer.mozilla.org/en-US/docs/Learn/Common_questions/Web_mechanics/What_is_a_web_server`. Visited on: 07-10-2024.
6. MYERS, Glenford J.; SANDLER, Corey; BADGETT, Tom. *The Art of Software Testing*. Wiley, 2011.
7. CHUNG, Lawrence; NIXON, Brian A; YU, Eric; MYLOPOULOS, John. *Non-Functional Requirements in Software Engineering*. Springer, 1999.
8. FRANCESCO CHIONNA Piero Cirillo, Vito Palmieri; BELLONE, Mauro. A Proposed Hardware-Software Architecture for Virtual Reality in Industrial Applications. In: *Proceedings of the 2015 IEEE Conference on Industrial Technology*. IEEE, 2015.
9. WANG, Xiaoyin. VRTest: An Extensible Framework for Automatic Testing of Virtual Reality Scenes. In: *Companion Proceedings of the 44th International Conference on Software Engineering (ICSE '22 Companion)*. New York, NY, USA: ACM, 2022, s. 5. Dostupné z DOI: `10.1145/3510454.3516870`.

10. WAN, Zhiyuan; ZHANG, Yun; XIA, Xin; JIANG, Yi; LO, David. Software Architecture in Practice: Challenges and Opportunities. 2023.

11. MEDHAT, Nader. *Scaling Monolithic Applications* [Online;]. 2023. Dostupné tiež z: `https://medium.com/swlh/scaling-monolithic-applications-3c69193f942a`. Visited on: 07-10-2024.

12. PITÁK, F. *Systém pre dlhodobú správu zadaní úloh* [`https://dai.fmph.uniba.sk/~petrovic/th23/Pitak_listng.pdf`]. 2023.

13. PALÁT, M. *Zabezpečení komunikace Push To Talk* [`https://is.muni.cz/th/qrbuh/Zabezpeceni_komunikace_Push_to_talk.pdf`]. 2006.

14. 2024 IEEE/ACM International Workshop New Trends in Software Architecture. In: *SATrends 2024*. Lisbon, Portugal, 2024.

15. VAN HEESCH, Uwe; AVGERIOU, Paris; HILLIARD, Richard. A Documentation Framework for Architecture Decisions. *Journal of Systems and Software*. 2012, roč. 85, č. 3, s. 795–820. Dostupné z DOI: `10.1016/j.jss.2011.09.017`. Visited on: 02-01-2025.

16. BROSCH, Franz; BUHNOVA, Barbora; KOZIOLEK, Heiko; REUSSNER, Ralf. Reliability Prediction for Fault-Tolerant Software Architectures. *IEEE Transactions on Software Engineering*. 2009, roč. 35, č. 4, s. 492–508.

17. LYU, Michael R. Software Reliability Engineering: A Roadmap. In: *Proceedings of the Future of Software Engineering*. IEEE, 2007.

18. OWASP FOUNDATION, INC. *OWASP Dependency-Check*. 2025. Dostupné tiež z: `https://owasp.org/www-project-dependency-check/`. Visited on: 05-03-2025, 2025.

19. SERGIO, Lema. *OAuth2 with Google, Spring Boot, and Angular*. 2024-07-22. Dostupné tiež z: `https://sergiolema.dev/2024/07/22/oauth2-with-google-spring-boot-and-angular/`. Visited on: 14-05-2025.

Zoznam skratiek

AAA Arrange-Act-Assert.

API Application Programming Interface.

CLI Command Line Interface.

CORS Cross-origin Resource Sharing.

CRUD Create, Read, Update, Delete.

DAO Data Access Object.

DOM Document Object Model.

DTO Data Transfer Object.

ECS Entity-Component-System.

HTML Hypertext Markup Language.

HTTP Hypertext Transfer Protocol.

HTTPS Hypertext Transfer Protocol Secure.

IDE Integrated Development Environment.

JWT JSON Web Token.

MTTF Mean Time To Failure.

MTTR Mean Time To Repair.

MVC Model - View - Controller.

OOP Object Oriented Programming.

REST Representational State Transfer.

TDD Test Driven Development.

URL Uniform Resource Locator.

VR Virtual Reality.

Zoznam príloh

Príloha A Príručka na testovanie

Príloha B Systémova príručka testovacieho prostredia

Príloha C CD médium – záverečná práca v elektronickej podobe

A Príručka na testovanie

A.1 Aplikácia testovacieho prostredia

Hlavným cieľom tejto bakalárskej práce je zdokonaliť a otestovať virtuálne výučbové prostredie, aby sa eliminovala možnosť výskytu rôznych typov chýb v hotovom produkte, ako aj v procese zmien a ďalšieho vývoja. Vytvorené prostredie kontroluje serverovú časť spolu s komunikáciou s databázou, ako aj frontend používateľského panela a samotné virtuálne scény na správnu a očakávanú prevádzku. Preto je táto testovacia príručka určená skôr pre vývojárov a technických správcov systému.

A.2 Spustenie prostredia

A.2.1 Požiadavky na softvér

Nižšie je uvedených niekoľko spôsobov spustenia testov, ale pre každý z nich je potrebné stiahnuť archív s kódom projektu a rozbaliť ho.

- Ak spúšťate projekt pomocou nástroja Docker, okrem jeho softvérového balíka, nepotrebujete žiadny iný softvér.
- Ak spúšťate projekt prostredníctvom príkazového riadka alebo IDE, budete potrebovať nainštalovať nasledujúci zoznam softvéru:
  - **Java JDK** (odporúčaná verzia 17)
  - **Maven**
  - **Node.js** (odporúčaná verzia 22.12.0)
  - **PostgreSQL** (minimálna verzia 15.0)

A.2.2 Testovanie s pomocou Docker

Jedným z najjednoduchších spôsobov je spustenie s pomoocou softvérovej kontajnerovej aplikácie Docker. keďže testovacie príkazy boli zakompilované do Dockerfile každého z certifikovaných modulov, kompletné testovacie prostredie sa dá jednoducho otestovať pomocou `docker-compose` podobne, ako je to uvedené v systémovej príručke v bakalárskej práci Dmytra Demianenka [2].

1. Stiahnite si bezplatný balík Docker z *oficiálnej stránky aplikácie*[8]
2. Po inštalácii do počítača sa uistite, že Docker Server beží v aktívnom režime.
3. V príkazovom riadku prejdite do koreňového priečinka (predvolene LirkisEduVePn).
4. Spustiť všetky kontajnery paralelne pomocou príkazov na príkazovom riadku:
   - pri prvom alebo štandardnom spustení bez zmien v kóde zadajte: **docker compose up**
   - ak ste v projekte vykonali nejaké zmeny, prestavte všetky kontajnery na najnovšiu verziu kódu pomocou: **docker compose up –build**
5. Potom v termináli vyhľadajte výsledky testov zo všetkých častí a analyzujte ich (analýza je podrobne opísaná v nasledujúcej časti 3)

A.2.3 Testovanie s pomocou IDE

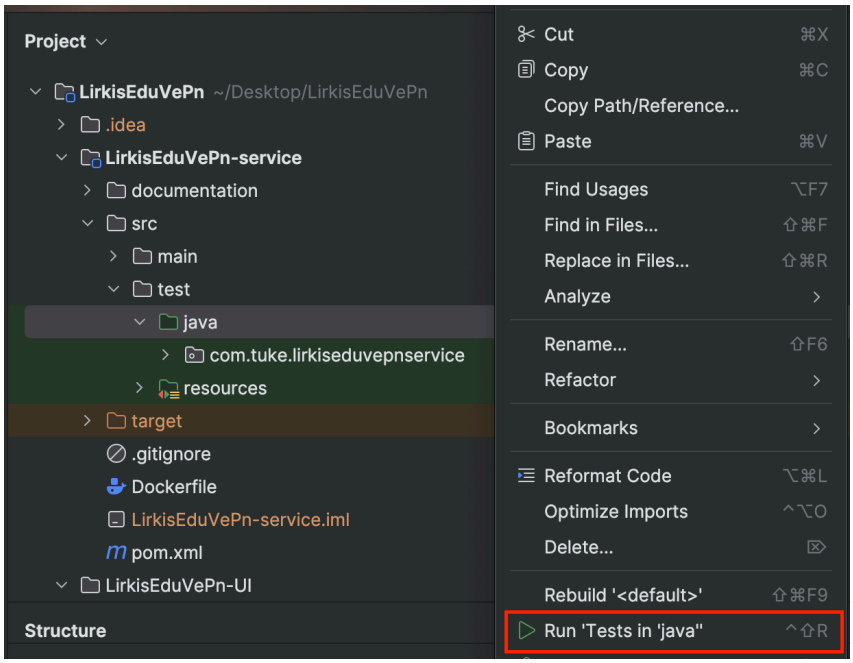
Ďalším jednoduchým spôsobom je testovanie priamo vo vývojovom prostredí (IDE). Najviac odporúčame *IntelliJ IDEA*[9], pretože dobre podporuje jazyky Java a JavaScript a jeho používateľsky prívetivé rozhranie. Ďalšie kroky budú znazornene práve v ňom:

1. Ak ešte nemáte nainštalované vhodné IDE, musíte ako prvý krok urobiť toto.
2. Pri tejto metóde je potrebné spustiť testy oddelene na strane servera a na strane klienta. Preto sú v nasledujúcich podkapitolách popísané obe:
   - V module servera

[8]Oficiálná stránka Docker: `https://www.docker.com`
[9]Oficiálná stránka IntelliJ IDEA: `https://www.jetbrains.com/idea/`

(a) Vyhľadajte priečinok `LirkisEduVePn-service/src/test/java` a kliknite pravým tlačidlom myši.

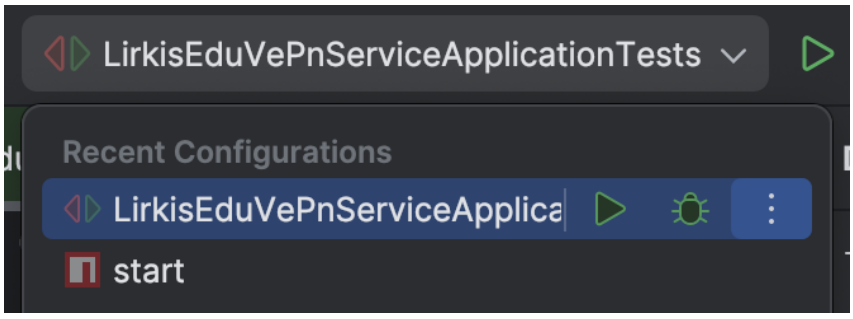
(b) V dialógovom okne, ako je znázornené na obrázku A.1 , kliknite na možnosť **"Run 'Tests in java'"**.



Obrázok A.1: Zobrazenie dialógového okna v prostredí IntelliJ IDEA

(c) Potom v konzole, ktorá sa objaví, vyhľadajte výsledky testov zo všetkých častí a analyzujte ich (analýza je podrobne opísaná v nasledujúcej časti 3)

(d) Následne môžete rýchlo spustiť testy z horného menu, ako je znázornené na obrázku A.2

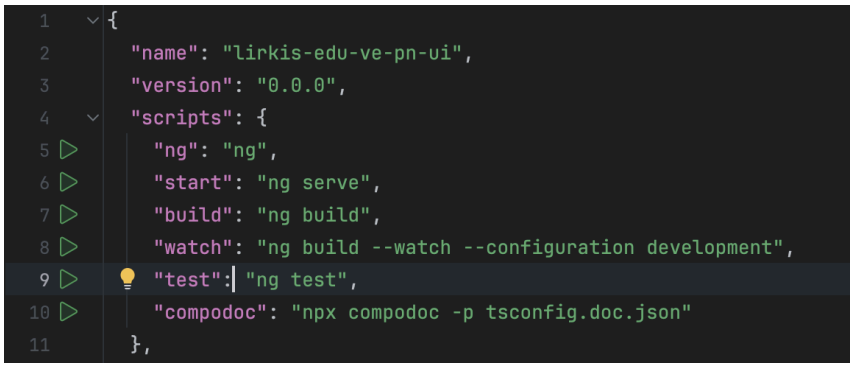


Obrázok A.2: Okno na rýchle spustenie a debagovanie v prostredí IntelliJ IDEA

- Na strane klienta:

(a) Vyhľadajte súbor `LirkisEduVePn-UI/package.json` a kliknite pravým tlačidlom myši.

(b) Na 9. riadku (ako je zvýraznené na obrázku A.3) kliknite na zelený trojuholník, aby sa spustili testy.



Obrázok A.3: Konfigurácia front-endu v prostredí IntelliJ IDEA

(c) Potom v konzole, ktorá sa objaví, vyhľadajte výsledky testov zo všetkých častí a analyzujte ich (analýza je podrobne opísaná v nasledujúcej časti 3). Otvorí webovú stránku na adrese: **http://localhost:9876/?id=** s grafickými výsledkami testov.

(d) Následne môžete rýchlo spustiť testy z horného menu, ako je znázornené na obrázku A.2

A.2.4 Testovanie pomocou príkazového riadku

Pre túto metódu by ste potrebovali mať osvojené Maven CLI a Angular CLI. Terminál môžete použiť buď vo Vašom systéme, alebo v už spomínaných IDE.

1. Pri tejto metóde je potrebné spustiť testy oddelene na strane servera a na strane klienta. Preto sú v nasledujúcich podkapitolách popísané obe:

   - V module servera

     (a) Prejdite na priečinok `LirkisEduVePn-service` s pomocou: **cd LirkisEduVePn-service**.

     (b) Spustite príkaz **mvnw install** aby prebudovať projekt a spustiť všetky testy paralelne z testovacieho prostredia Spring.

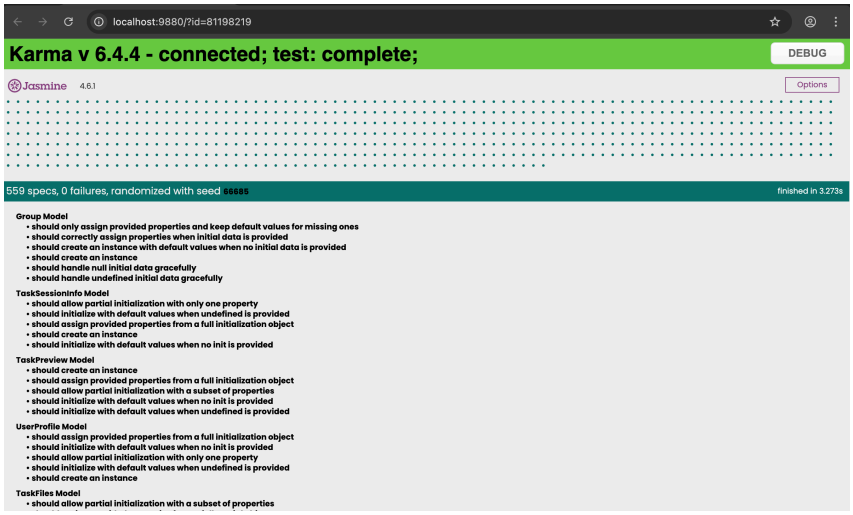
     (c) Potom v konzole, ktorá sa objaví, vyhľadajte výsledky testov zo všetkých častí a analyzujte ich (analýza je podrobne opísaná v nasledujúcej časti 3)

   - Na strane klienta:

     (a) Prejdite na priečinok `LirkisEduVePn-UI` s pomocou: **cd LirkisEduVePn-UI**.

(b) Spustite príkaz **ng test** na spustenie testov v prostredí Karma.

(c) Potom v konzole, ktorá sa objaví, vyhľadajte výsledky testov zo všetkých častí a analyzujte ich (analýza je podrobne opísaná v nasledujúcej časti 3). Otvorí webovú stránku na adrese: **http://localhost:9876/?id=** s grafickými výsledkami testov (obrázok A.4).



Obrázok A.4: Výsledky testov Karma v prehliadači

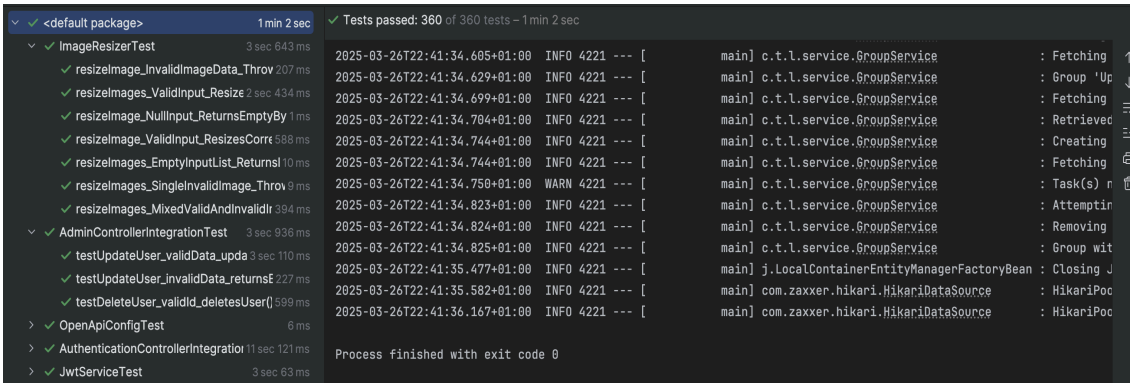
A.3 Analýza výsledkov testov

Testy sa už spustili, ale kľúčový výsledok prinesie ich analýza a prípadné úpravy výrobku. Preto v ďalšej časti vysvetlíme, ako interpretovať protokoly testov, aby ste z nich vyťažili čo najviac.

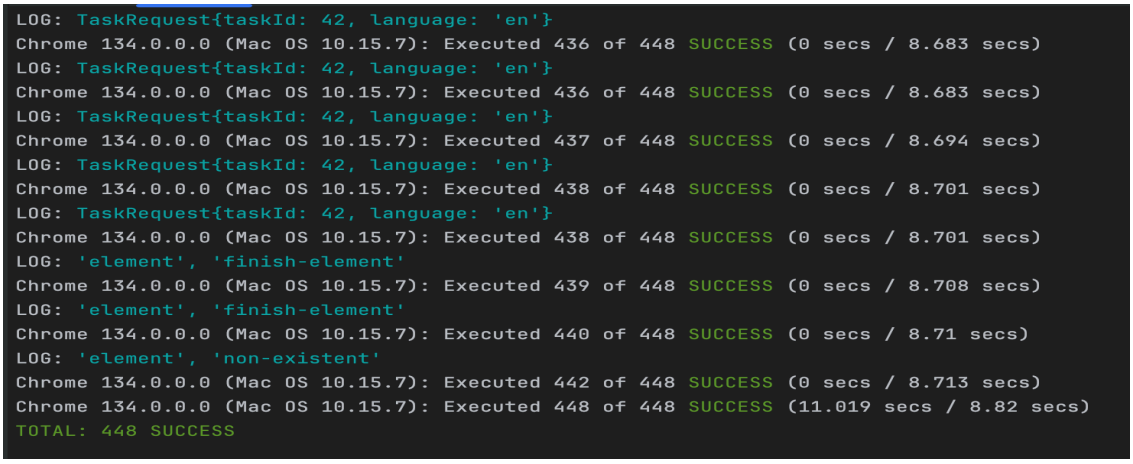
A.3.1 Hlavný cieľ a čo je potrebné snažiť sa dosiahnuť

Úspešné testovanie predstavuje základný kameň kvality vyvíjaného systému, pričom jeho výsledky potvrdzujú očakávanú a správnu funkčnosť produktu. Pri jednotkovom testovaní boli jednotlivé časti kódu a ich funkcie dôkladne preverené, čím sme zabezpečili, že každý modul pracuje presne podľa definovaných špecifikácií. Tento prístup umožnil identifikovať a odstrániť potenciálne chyby už v počiatočných fázach vývoja, čo výrazne prispieva k stabilite a udržateľnosti softvéru. Na druhej strane, integračné testy sa zameriavali na overenie korektnej spolupráce medzi jednotlivými komponentmi systému. Tieto testy preukázali, že navzájom prepojené moduly správne komunikujú a spoločne vytvárajú konzistentný a funkčný celok bez chýb. Celkový úspech testov potvrdzuje, že

implementované riešenia spĺňajú stanovené požiadavky a zabezpečujú vysokú mieru spoľahlivosti produktu. Tento systematický prístup k testovaniu nielenže umožňuje efektívne odhaliť a opraviť nedostatky, ale zároveň slúži aj ako dôležitý nástroj pre kontinuálne zlepšovanie kvality softvéru v celom vývojovom procese. Nasledujúce obrázky preto ukazujú, ako vyzerá výstup terminálu, ktorý indikuje úspešné prejdenie všetkých predvolených nastavení a zabudovaných systémových testov jednotlivých modulov:



(a) backend (Maven)



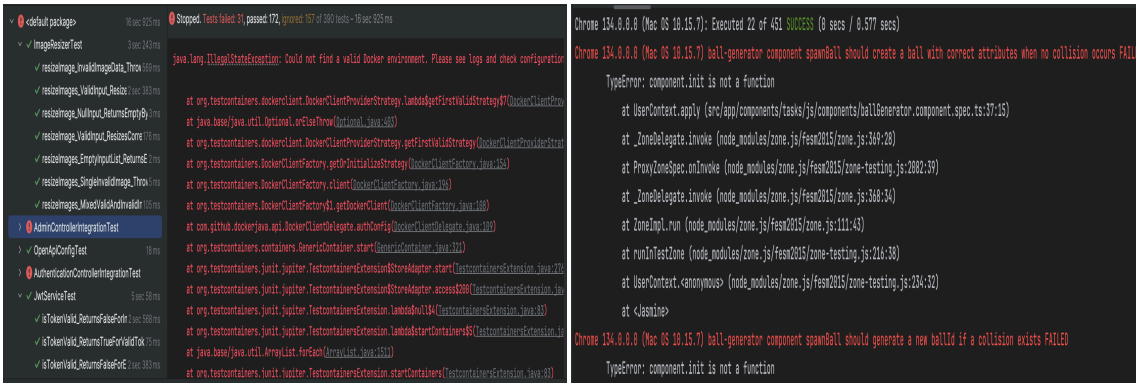
(b) frontend (Karma)

Obrázok A.5: Úspešne vysledky testov

A.3.2 Interpretácia chýb v testoch

Hoci sa na prvý pohľad môže zdať, že chyby v testoch sú niečo nežiaduce, nie je to celkom pravda, pretože takýto test splnil svoju úlohu - našiel nedostatok. Ako sa uvádza v knihe „The Art of Software Testing“ od Glenforda J. Myersa, Coreyho Sandlera a Toma Badgetta [6], testovací prípad, ktorý odhalí chybu v systéme, je hodnotnejšou investíciou ako testovací prípad, ktorý len potvrdzuje, že systém neobsahuje chyby. Preto musíte **správne interpretovať chybové hlásenia,**

nájsť miesto chyby, či už v teste alebo v kóde produktu, a zaviesť najúčinnejšiu metódu na odstránenie problému. Nasledujúce obrázky ukazujú príklady toho, ako vyzerajú chybné testy v testovacom prostredí servera a v testovacom prostredí klienta:



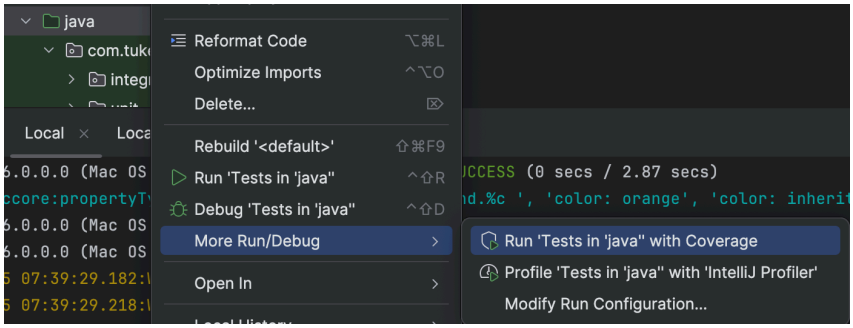
(a) backend (Maven) (b) frontend (Karma)

Obrázok A.6: Problémy zistené pri testoch

A.3.3 Kontrola pokrytia kódu testami

Je veľmi dôležité, aby sa čo najviac kódu otestovalo, čím sa zníži počet netestovaných zraniteľností v systéme. Pokrytie kódu je miera toho, koľko percent príkazov, logických vetiev, funkcií a riadkov kódu má zodpovedajúci testovací skript, ktorý overuje ich funkčnosť.

- V module servera
    1. Vyhľadajte priečinok `LirkisEduVePn-service/src/test/java` a kliknite pravým tlačidlom myši.
    2. V dialógovom okne, ako je znázornené na obrázku A.7, kliknite na možnosť **"More Run/Debug" a "Run 'Tests in java' with Coverage"**.

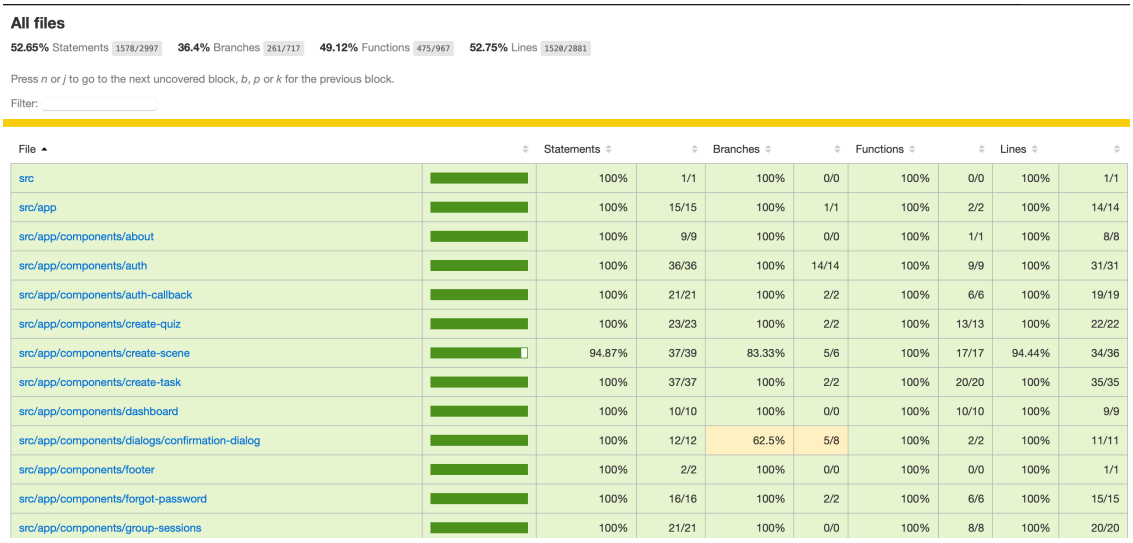


Obrázok A.7: Postup pokrytia testmi v prostredí IntelliJ IDEA

3. Po úspešnom vykonaní všetkých testov sa v paneli vpravo zobrazia podrobnosti o pokrytí testov, ako je znázornené na obrázku 10.1b v kapitole ”10 Vyhodnotenie práce”.

- Na strane klienta:

1. Prejdite na priečinok `LirkisEduVePn-UI` s pomocou: **cd LirkisEduVePn-UI**.
2. Spustite príkaz **ng test –code-coverage=true** na spustenie testov v prostredí Karma so s zobrazením detailov pokrytia.
3. Potom v konzole, ktorá sa objaví, sa zobrazí tabuľka s podrobnosťami o pokrytí, podobne ako na obrázku 10.2b v kapitole ”10 Vyhodnotenie práce”.
4. Vo vytvorenom priečinku `coverage/lirkis-edu-ve-pn-ui` otvorte súbor `index.html` v ľubovoľnom prehliadači a uvidíte grafickú vizualizáciu s podrobným popisom, ako je znázornené na obrázku A.8



File	Statements		Branches		Functions		Lines	
src	100%	1/1	100%	0/0	100%	0/0	100%	1/1
src/app	100%	15/15	100%	1/1	100%	2/2	100%	14/14
src/app/components/about	100%	9/9	100%	0/0	100%	1/1	100%	8/8
src/app/components/auth	100%	36/36	100%	14/14	100%	9/9	100%	31/31
src/app/components/auth-callback	100%	21/21	100%	2/2	100%	6/6	100%	19/19
src/app/components/create-quiz	100%	23/23	100%	2/2	100%	13/13	100%	22/22
src/app/components/create-scene	94.87%	37/39	83.33%	5/6	100%	17/17	94.44%	34/36
src/app/components/create-task	100%	37/37	100%	2/2	100%	20/20	100%	35/35
src/app/components/dashboard	100%	10/10	100%	0/0	100%	10/10	100%	9/9
src/app/components/dialogs/confirmation-dialog	100%	12/12	62.5%	5/8	100%	2/2	100%	11/11
src/app/components/footer	100%	2/2	100%	0/0	100%	0/0	100%	1/1
src/app/components/forgot-password	100%	16/16	100%	2/2	100%	6/6	100%	15/15
src/app/components/group-sessions	100%	21/21	100%	0/0	100%	8/8	100%	20/20

Obrázok A.8: Vizualizácia pokrytia testmi v prehliadači

B Systémova príručka testovacieho prostredia

Táto príloha predstavuje systémovú príručku testovacieho prostredia, ktoré zohráva kľúčovú úlohu pri zabezpečení kvality softvéru. Testovacie prostredie je navrhnuté tak, aby podporovalo jednotkové a integračné testy, ktoré sú nevyhnutné na overenie správnosti jednotlivých komponentov systému a ich vzájomnej spolupráce. Jednotkové testy umožňujú identifikovať chyby na úrovni jednotlivých modulov, zatiaľ čo integračné testy sa zameriavajú na overenie správnej interakcie medzi týmito modulmi.

Nasledujúce kapitoly podrobne opisujú architektúru testovacieho prostredia, jednotlivé testovacie súbory a ich konfiguráciu. Časť servera Spring Boot, časť webového klienta Angular z časťou virtuálneho prostredia popíšeme samostatne, pretože testy týcht komponentov projektu si potrebovali rôzne technológie, techniky a nastavenia na ich realizáciu.

B.1 Architektúra testovacieho prostredia servera Spring Boot

Testovacie prostredie servera Spring Boot je postavené na základe architektúry MVC (Model-View-Controller), ktorá umožňuje efektívne oddelenie logiky aplikácie od jej prezentácie. Táto architektúra zabezpečuje, že jednotlivé komponenty systému sú navzájom nezávislé, čo uľahčuje testovanie a údržbu kódu. Testovacie prostredie je rozdelené do niekoľkých hlavných častí, ktoré sú zamerané na rôzne aspekty testovania. Vo výpise 5.1 v časti **"5 Testovanie serverovej vrstvy"** sme už spomenuli všeobecnú kostru súborovej architektúry testov.

Predovšetkým sa konfiguračné súbory skladajú z nasledujúcich častí:

- **.testconteiners.properties** - konfiguračná vlastnosť zapína opätovné používanie bežiacich kontajnerov Testcontainers medzi jednotlivými spusteniami

testov. Jej cieľom je minimalizovať čas potrebný na štart a ukončenie Docker kontajnerov, čím sa zrýchli celkový priebeh testov. V testovacom prostredí zabezpečuje, že namiesto opakovaného vytvárania nových kontajnerov.

- **application-test.properties** - tento súbor definuje špecifické nastavenia Spring Boot aplikácie pre testovacie prostredie. Nastavuje pripojenie k PostgreSQL databáze spustenou v Testcontainers kontajneri (`jdbc:tc:postgresql:15.1:///test-db`), automatické vytváranie a odstraňovanie schémy (`spring.jpa.hibernate.ddl-auto=create-drop`), ako aj parametre poolovania pripojení HikariCP. Okrem toho obsahuje pevne dané prihlasovacie údaje a URL adresy servera a klienta, čo zabezpečuje izolované a deterministické správanie počas testov. Výhodou je plná autonómnosť testov (nie je potrebná externá databáza), reprodukovateľnosť výsledkov a stabilné prostredie bez rušivých vplyvov z produkcie či lokálnych strojov.

Priečinok `LirkisEduVePn-service/src/test/java/com.tuke.lirkiseduvepnservice/` je rozdelený na dve časti - jednotkové a integračné testy.

B.1.1 Jednotkové testy na serveri Spring Boot

Jednotkové testy sú zamerané na overenie správnosti jednotlivých komponentov systému, ako sú služby, kontroléry a pomocné nástroje. Tieto testy sa vykonávajú izolovane, bez závislosti na vonkajších systémoch, čo umožňuje rýchle a efektívne overenie funkčnosti jednotlivých častí aplikácie.

Pri Entity-Component-System (ECS) architektúre servera nášho projektu boli jednotkovými testami pokryté tieto prvky:

- **ovládanie chyb**
- **služby**
- **konfigurácie**
- **ovládače**
- **úžitkové triedy**
- **mapovače**

V súbore `ErrorHandlerTest.java` sa testuje spracovanie rôznych výnimiek v rámci vlastnej triedy `ErrorHandler`, konkrétne mapovanie výnimiek ako `PasswordMismatchedException`, `ResourceNotFoundException`, `IOException`, `ZipException`, `NoSuchElementException` či `IllegalStateException` na príslušné HTTP status kódy a telá odpovedí. Cieľom týchto testov je overiť, že pri výskyte každej z týchto výnimiek sa vráti konzistentná a predvídateľná HTTP odpoveď s očakávaným stavovým kódom a textovou správou, čo zabezpečuje plynulosť a spoľahlivosť API pri chybových stavoch.

Testy služieb

Služby vo Spring Boot vrstva služieb (Service Layer) predstavuje centrálny modul, ktorý zoskupuje a realizuje obchodnú logiku aplikácie, aby sa rozdelili zodpovednosti medzi prezentáciu a perzistenciu. Táto vrstva zároveň zabezpečuje konzistentné manažovanie transakcií, validáciu vstupných dát a orchestráciu volaní medzi repozitármi, čím zvyšuje testovateľnosť a udržiavateľnosť systému. Výpis B.1 zobrazuje zoznam testovacích jednotiek pre služby:

Výpis B.1: Služby servera sú testované pomocou jednotkových testov

```
LirkisEduVePn-service
├── documentation
├── src
│   ├── main
│   └── test
│       ├── java
│       │   ├── com.tuke.lirkiseduvepnservice
│       │   │   ├── integration
│       │   │   └── unit
│       │   │       ├── config
│       │   │       ├── controllers
│       │   │       ├── mappers
│       │   │       ├── services
│       │   │       │   ├── AdminServiceTest.java
│       │   │       │   ├── AuthenticationServiceTest.java
│       │   │       │   ├── ConfirmationTokenServiceTest.java
│       │   │       │   ├── EmailServiceTest.java
│       │   │       │   ├── FiringAttemptServiceTest.java
│       │   │       │   ├── GroupServiceTest.java
```

```
│       │   │       │   ├── ImageResizerTest.java
│       │   │       │   ├── JwtServiceTest.java
│       │   │       │   ├── OAuthSuccessHandlerTest.java
│       │   │       │   ├── OAuthUserServiceTest.java
│       │   │       │   ├── PasswordResetTokenServiceTest.java
│       │   │       │   ├── PhotoExtractorTest.java
│       │   │       │   ├── ScenarioServiceTest.java
│       │   │       │   ├── SceneServiceTest.java
│       │   │       │   ├── TaskServiceTest.java
│       │   │       │   ├── TaskSessionServiceTest.java
│       │   │       │   └── UserServiceTest.java
│       │   │       └── ErrorHandlerTest
│       │   └── LirkisEduVePnServiceApplicationTests
│       └── resources
└── ...
```

V jednotkových testoch služieb servera sa overuje správna implementácia hlavnej logiky servera, vrátane spravovania autentifikácie, generovania a validácie resetovacích tokenov, extrakcie a spracovania fotografií, ako aj Create, Read, Update, Delete (CRUD) operácií pre scenáre, scény, úlohy a používateľov. Testy ďalej kontrolujú konzistentné manažovanie relácií úloh, správnu manipuláciu s entitami používateľa a spracovanie výnimiek, čím zabezpečujú transakčnú integritu a spoľahlivú spoluprácu so Spring Data repozitármi.

Testy konfiguračných tried

Konfiguračné testy sú zamerané na overenie správnosti nastavení a konfigurácií aplikácie. Tieto testy zabezpečujú, že všetky potrebné komponenty sú správne nakonfigurované a pripravené na použitie. Triedy konfiguračných testov sú nasledovné:

- **ApplicationConfigTest.java** – overuje, či sú v triede `ApplicationConfig` správne definované Spring beany, najmä že metóda `passwordEncoder()` vracia inštanciu `BCryptPasswordEncoder` a že `authenticationProvider()` inicializuje `DaoAuthenticationProvider` s platnou službou `UserDetailsService` a kódovačom hesla. Benefitom týchto testov je včasné zachytenie nesprávnej konfigurácie bezpečnostných komponentov, čo znižuje riziko chýb pri autentifikácii v produkčnom prostredí.

- **JwtAuthenticationFilterTest.java** – testuje spracovanie HTTP požiadaviek filtrom `JwtAuthenticationFilter`, konkrétne že pri platnom JWT tokene sa do `SecurityContextHolder` uloží korektný `Authentication` objekt a spracovanie pokračuje bez prerušenia, zatiaľ čo pri chýbajúcom alebo neplatnom hlavičkovom poli zostáva kontext prázdny bez vyhadzovania neželaných výnimiek. Tieto testy zabezpečujú konzistenciu a spoľahlivosť JWT autentifikácie, čo je kľúčové pre bezpečnosť API.

- **OpenApiConfigTest.java** – overuje správanie statického inicializačného bloku v triede `OpenApiConfig` pri načítaní systémovej premennej `server.url` z konfiguračného súboru, a to v scenároch absencie súboru, neplatného formátu, prázdnej hodnoty či pri výnimke počas čítania. Výhodou týchto testov je predvídateľnosť a odolnosť generovanej OpenAPI dokumentácie naprieč rôznymi prostrediami bez nečakaných zmien systémových premenných.

- **OpenApiConfigTestHelper.java** – poskytuje pomocné metódy na simuláciu načítania súboru `application.properties` cez vlastný `ClassLoader`, čím umožňuje izolovane testovať statický blok inicializácie `OpenApiConfig` bez úprav produkčného kódu. Vďaka tomu dokážeme v testoch elegantne nasimulovať rôzne konfigurácie a chybové stavy, čím sa zvyšuje pokrytie a kvalita overenia konfigurácie.

Testy ovládačov

Ovládače vo Spring Boot aplikácii sú zodpovedné za spracovanie HTTP požiadaviek a odpovedí. Tieto komponenty zabezpečujú, že požiadavky sú správne smerované na príslušné služby a že odpovede sú vrátené v správnom formáte. Ovládače sú testované pomocou mock objektov, ktoré simulujú správanie závislostí, čím sa zabezpečuje izolované testovanie jednotlivých komponentov. Výpis B.2 obsahuje zoznam pridaných testovacích tried

Výpis B.2: Ovládače servera sú testované pomocou jednotkových testov

```
LirkisEduVePn-service
├── documentation
├── src
│   ├── main
│   └── test
│       ├── java
```

```
│       │   ├── com.tuke.lirkiseduvepnservice
│       │   │   ├── integration
│       │   │   └── unit
│       │   │       ├── config
│       │   │       ├── controllers
│       │   │       │   ├── AdminControllerTest.java
│       │   │       │   ├── AuthenticationControllerTest.java
│       │   │       │   ├── FiringAttemptControllerTest.java
│       │   │       │   ├── GroupControllerTest.java
│       │   │       │   ├── ScenarioControllerTest.java
│       │   │       │   ├── SceneControllerTest.java
│       │   │       │   ├── TaskControllerTest.java
│       │   │       │   ├── TaskSessionControllerTest.java
│       │   │       │   └── UserControllerTest.java
│       │   │       ├── mappers
│       │   │       ├── services
│       │   │       └── ErrorHandlerTest
│       │   └── LirkisEduVePnServiceApplicationTests
│       └── resources
└── ...
```

Testy mapovačov

Mapovače v aplikácii slúžia na prevod medzi entitami - Data Access Object (DAO), a prenosovými objektmi - Data Transfer Object (DTO), pričom zabezpečujú korektné mapovanie základných i vnorených polí, transformáciu kolekcií aj ošetrenie `null` vstupov. V testovacích triedach sa overuje nielen správne namapovanie identifikátorov a textových polí, ale aj konverzia výčtových hodnôt, filtrovanie alebo rozbalenie vnorených entít, čím sa garantuje správnosť a konzistentnosť dátovej vrstvy pri všetkých bežných i hranových scenároch.

- **GroupMapperTest.java** – overuje metódy `daoToDto()` a `daoToDtoWithGroups()`, kontroluje správne namapovanie `id` a `name`, konverziu zoznamu `Task` na `TaskNames`, spracovanie prázdnych a `null` vstupov vrátane špecifického prípadu `ArrayList` a mapovanie úloh s `null` poľami.

- **ScenarioMapperTest.java** – testuje metódu `daoToScenarioPreview()`, overuje mapovanie zoznamu `LanguageFile` na reťazce s kódom jazyka

a zoznamu `ScenarioPhoto` na polia bajtov, ošetruje prázdne kolekcie, `null` vstup aj prvky so `null` poľami.

- **SceneMapperTest.java** – pokrýva funkciu `daoToScenePreview()`, zabezpečuje mapovanie všetkých polí entity `Scene` (vrátane binárneho poľa `photo`), testuje správanie pri `null` vstupoch a zachovanie `null` hodnôt v DTO.
- **TaskMapperTest.java** – skúma metódu `daoToTaskPreview()`, kontroluje mapovanie základných polí (`id`, `name`, `description`), vkladanie vnorených DTO pre `Scenario` a `Scene`, filtrovanie otvorených relácií `TaskSession`, spracovanie prázdnych a dokončených sedení, ako aj zachovanie `null` polí a mapovanie identifikátorov aj pri chýbajúcich vnorených entitách.
- **TaskSessionMapperTest.java** – overuje `daoToTaskSessionFinishRequest()`, mapovanie `id` → `taskSessionId`, `user.id` → `userId`, `successful` a `finishTime`, ošetruje `null` vstup (vracia `null`), a testuje `populateUser()` hook pre správne priradenie `User` pri nenulovom `userId`.
- **UserMapperTest.java** – overuje `daoToDto()` v `UserMapper`, mapovanie všetkých skalárnych polí vrátane prevodu `Role` na reťazec s predvolenom `"STUDENT"` pri `null`, spracovanie kolekcie `Group` (aj s prázdnymi hodnotami) a správnu konverziu prvkov na `GroupDto`.

B.1.2 Integračné testy na serveri Spring Boot

Integračné testy sú navrhnuté na overenie správnej spolupráce medzi jednotlivými komponentmi aplikácie v prostredí, ktoré verne simuluje produkčné podmienky behu. V projekte sa využíva framework Testcontainers na spúšťanie izolovaných Docker kontajnerov s PostgreSQL databázou, čím je zabezpečená čistota a reprodukovateľnosť testovacieho prostredia. Trieda `BaseIntegrationTest` centralizuje konfiguráciu kontajnerov pomocou anotácií `@Testcontainers` a `@ServiceConnection`, pričom následné testovacie triedy dedičia spoločné nastavenia. Výpis B.3 ukazuje, ako sú integračné testy organizované.

Výpis B.3: Štruktúra súborov integračných testov v LirkisEduVePn-service

```
LirkisEduVePn-service
├── documentation
├── src
│   ├── main
```

```
│   └── test
│       ├── java
│       │   ├── com.tuke.lirkiseduvepnservice
│       │   │   ├── integration
│       │   │   │   ├── controllers
│       │   │   │   ├── repositories
│       │   │   │   ├── services
│       │   │   │   └── BaseintegrationTest.java
│       │   │   └── unit
│       │   └── LirkisEduVePnServiceApplicationTests
│       └── resources
└── ...
```

Integračné testy ovládačov

Integračné testy ovládačov overujú správnu expozíciu REST API adries a ich interakciu so službami, bezpečnostnou vrstvou a databázou v reálnom testovacom prostredí založenom na module Testcontainers.

- **ControllerBaseIntegrationTest.java** – zdrojová trieda, ktorá centralizuje konfiguráciu Testcontainers, Spring Test MVC, autentifikácie a spoločné nastavenia pre všetky integračné testy kontrolérov.
- **AdminControllerIntegrationTest.java** – overuje koncové body pre správu administrátorských operácií, vrátane získania zoznamu používateľov s rolou ADMIN, pridávania a aktualizácie admin účtov a správneho odhadzovania oprávnení pri neautorizovaných požiadavkách.
- **AuthenticationControllerIntegrationTest.java** – testuje koncové body pre registráciu, prihlásenie a obnovu JWT tokenov; konfirmuje vrátenie platného prístupového a refresh tokenu, ako aj spracovanie neplatných prihlasovacích údajov s očakávanými stavmi 401 či 400.
- **FiringAttemptControllerIntegrationTest.java** – overuje koncové body na evidenciu neúspešných pokusov o autentifikáciu, vrátane vytvorenia novej položky v databáze pri neplatnom hesle a možnosti načítania agregovaných štatistík o počte pokusov.
- **GroupControllerIntegrationTest.java** – kontroluje CRUD operácie skupín: vytvorenie novej skupiny, načítanie existujúcej skupiny podľa identifikátora, aktualizáciu a vymazanie, vrátane overenia správnych HTTP stavov.

- **ScenarioControllerIntegrationTest.java** – testuje koncové body pre správu scenárov, vrátane získania náhľadu scenára, vytvorenia, aktualizácie a odstránenia, a overuje mapovanie vzťahov na jazyky a fotografie.
- **SceneControllerIntegrationTest.java** – overuje REST API pre scény: nahrávanie a načítanie binárnych dát fotografie, manipuláciu s popismi scén a správne zvládnutie požiadaviek so `null` či prázdnymi poľami.
- **TaskControllerIntegrationTest.java** – testuje koncové body pre úlohy: získanie zoznamu úloh, detailu úlohy s vnorenými scenármi a scénami, filtrovanie podľa otvorených sedení a vytváranie nových úloh.
- **TaskSessionControllerIntegrationTest.java** – overuje správanie API pre relácie úloh, vrátane spustenia session, jej ukončenia, ukladania časových značiek a merania priebehu, ako aj získania histórie sedení pre danú úlohu.
- **UserControllerIntegrationTest.java** – kontroluje REST koncové body pre používateľov: registráciu, získanie detailu profilu, aktualizáciu údajov, priradenie k skupinám a spracovanie výnimiek pri neexistujúcich ID.

Integračné testy služieb

Vo všeobecnosti integračné testy servisnej vrstvy overujú správnu interakciu medzi službami, Spring Data repozitármi a databázou, transakčnú konzistenciu operácií, validáciu vstupných dát a spracovanie výnimiek v reálnom prostredí Testcontainers. Rovnako ako pri výpise B.1 je každej triede služieb priradená trieda s integračnými testami, medzi ktorými sú stručne

- **ServiceBaseIntegrationTest.java** – centralizuje spoločnú konfiguráciu Testcontainers, Spring Test a autentifikačné nastavenia pre všetky integračné testy služieb.
- **AuthenticationServiceIntegrationTest.java** – testuje registráciu a prihlásenie používateľa, generovanie a overenie JWT tokenov, ako aj spracovanie neplatných prihlasovacích údajov.
- **AdminServiceIntegrationTest.java** – overuje operácie služby `AdminService`, vrátane vytvárania, aktualizácie a mazania administrátorských účtov, prideľovania rolí a správnej perzistencie zmien v databáze.

- **ConfirmationTokenServiceIntegrationTest.java** – testuje generovanie, ukladanie, validáciu a vypršanie potvrdovacích tokenov pre registráciu a obnovu hesla, vrátane správania pri opätovnom požadovaní a expirovaní tokenu.
- **JwtServiceIntegrationTest.java** – overuje tvorbu prístupových a refresh tokenov, dekódovanie, validáciu podpisu a spracovanie expirovaných či neplatných JWT tokenov.
- **OAuthUserServiceIntegrationTest.java** – kontroluje spracovanie úspešného OAuth prihlásenia, vytváranie alebo aktualizáciu interných profilov používateľov na základe externých údajov.
- **OAuthSuccessHandlerIntegrationTest.java** – skúma správanie handlera po úspešnej OAuth autentifikácii, vrátane správneho nasmerovania používateľa a inicializácie kontextu bezpečnosti.
- **PasswordResetTokenServiceIntegrationTest.java** – testuje životný cyklus resetovacieho tokenu vrátane jeho generovania, ukladania, overovania platnosti a vypršania.
- **FiringAttemptServiceIntegrationTest.java** – overuje zaznamenávanie neúspešných pokusov o autentifikáciu a agregáciu štatistík podľa používateľa a časového intervalu.
- **ScenarioServiceIntegrationTest.java** – overuje CRUD operácie nad entitou `Scenario`, vrátane spracovania vnorených jazykových súborov a obrázkov.
- **SceneServiceIntegrationTest.java** – testuje ukladanie a načítanie binárnych dát fotografií, modifikáciu popisných polí a správanie pri `null` či prázdnych vstupoch.
- **TaskServiceIntegrationTest.java** – kontroluje vytváranie, aktualizáciu a vymazanie úloh, vrátane správy referencií na scenáre a scény a integrity relácií úloh.
- **TaskSessionServiceIntegrationTest.java** – overuje spustenie a ukončenie relácie úlohy, zaznamenávanie časových značiek a stavov sedení v databáze.
- **UserServiceIntegrationTest.java** – testuje CRUD operácie nad používateľmi, aktualizáciu profilových údajov, priraďovanie k skupinám a spracovanie výnimiek pri neexistujúcich identifikátoroch.

Integračné testy rrepozitárov

Integračné testy repozitárov overujú správnu interakciu medzi Spring Data repozitármi a databázou, vrátane CRUD operácií, transakčnej konzistencie a spracovania výnimiek. Tieto testy zabezpečujú, že všetky operácie nad entitami sú správne vykonané a že dáta sú korektne uložené a načítané z databázy. Boli pridané nasledujúce testy:

- **ConfirmationTokenRepositoryIntegrationTest.java** – overuje ukladanie a načítanie potvrdzovacích tokenov, funkčnosť metódy `findByToken()` a správne vymazanie expirovaných záznamov.
- **FiringAttemptRepositoryIntegrationTest.java** – testuje zaznamenávanie neúspešných pokusov o autentifikáciu, dotazy na agregovaný počet podľa používateľa a filtrovanie podľa časového intervalu.
- **GroupRepositoryIntegrationTest.java** – kontroluje CRUD operácie so skupinami a funkčnosť vlastnej metódy `findByName()`, vrátane spracovania prípadov s duplicitnými názvami.
- **PasswordResetTokenRepositoryIntegrationTest.java** – overuje ukladanie, načítanie a expirovanie resetovacích tokenov, ako aj vyhľadávanie pomocou `findByToken()`.
- **ScenarioRepositoryIntegrationTest.java** – testuje perzistenciu scenárov vrátane vnorených entít `LanguageFile` a `ScenarioPhoto`, CRUD operácie a vlastné dotazy na náhľady scenárov.
- **SceneRepositoryIntegrationTest.java** – overuje ukladanie a načítanie binárnych dát fotografií scén, CRUD operácie a dotazy `findByScenarioId()`.
- **TaskRepositoryIntegrationTest.java** – kontroluje ukladanie úloh, CRUD operácie, dotazy `findByScenarioId()` a filtrovanie podľa stavu otvorenosti relácií (napr. počet aktívnych `TaskSession`).
- **TaskSessionRepositoryIntegrationTest.java** – testuje perzistenciu relácií úloh, CRUD operácie a dotazy pre načítanie sedení podľa `taskId` a časových rozsahov.
- **UserRepositoryIntegrationTest.java** – overuje CRUD operácie nad používateľmi, funkčnosť metód `findByEmail()` a `findByGroupsContains()`, vrátane spracovania používateľov bez priradených skupín.

B.2 Testovacie prostredie klientskej časti v Angular

Klientska vrstva aplikácie využíva na jednotkové testy framework Jasmine v kombinácii s bežiacim prostredím Karma. Dôležitou súčasťou testovacieho prostredia je Angular TestBed, ktorý pomocou modulov a závislostí simuluje beh komponentov v izolovanom prostredí. Pri testovaní jednotlivých komponentov je potrebné overiť: správnu inicializáciu a návratové hodnoty životného cyklu (`ngOnInit()`, `ngOnDestroy()`), viazanie vstupov a výstupov, manipuláciu s Document Object Model (DOM) (napr. vykreslenie šablónnych elementov a reakcia na používateľské udalosti), korektnú injekciu a volanie služieb, spracovanie asynchrónnych volaní (prísľuby, Observables) a správanie pri chybových stavoch. Testy tak zaručujú, že každá časť používateľského rozhrania poskytuje očakávanú interaktivitu a validáciu vstupných dát.

B.2.1 Konfigurácia testovacieho prostredia v Angular

Konfigurácia testovacieho prostredia pre klientske testy v Angular je rozprestretá naprieč piatimi kľúčovými súbormi. V `package.json` sú definované závislosti (napr. `jasmine-core`, `karma`, `karma-chrome-launcher`, `karma-jasmine-html-reporter`) a skripty (`"test": "ng test"`), ktoré zabezpečujú spustenie testovacieho behu pomocou Angular CLI. Súbor `tsconfig.spec.json` rozširuje hlavnú konfiguračnú šablónu TypeScriptu a špecifikuje výstupný adresár `out-tsc/spec`, zoznam typov (`jasmine`, `node`) a zahrnutie `*.spec.ts` a pomocných deklarácií, vďaka čomu sa kompilujú len relevantné testovacie súbory.

V `karma.conf.js` sa konfigurujú testovací framework (Jasmine), reportéry (HTML, coverage), spúšťače prehliadačov (napr. ChromeHeadless) a ďalšie položky ako remapovanie súborov či proxy pravidlá. Súbor `src/polyfills.ts` načítava nevyhnutné polyfily (napr. `zone.js`) pre simuláciu prehliadačového prostredia v Node.js, čím sa zabezpečí kompatibilita testov so zónami Angular.

Nakoniec `src/test.ts` inicializuje Angular TestBed, registrovanie testing modulov a dynamické načítanie všetkých `*.spec.ts` súborov pred spustením testov, čo umožňuje izolované a konzistentné jednotkové testovanie komponentov a služieb.

B.2.2 Štruktúra testov klientskej časti

Prostredie je nakonfigurované tak, aby kompilátor našiel testovacie súbory s príponou `*.spec.ts`. Preto sme sa rozhodli pokryť testovacími súbormi tieto časti architektúry klienta:

- **komponenty**
- **služby**
- **typové modely**
- **súčasti scén virtuálneho prostredia**
- **doplnkové moduly pre Aframe komponenty**

V každej z týchto častí, ktorých rozloženie je uvedené vo výpise B.5, sú testy ďalej usporiadané podľa funkcií a modulov, čo zabezpečuje prehľadnosť a jednoduchú údržbu štruktúry testov.

Výpis B.4: Štruktúra súborov klientskej časti v Angular

```
LirkisEduVePn-UI
├── ...
├── coverage
├── src
│   ├── app
│   │   ├── components
│   │   │   ├── ...
│   │   │   └── tasks
│   │   │       ├── entities
│   │   │       ├── js
│   │   │       │   ├── components
│   │   │       │   ├── models
│   │   │       │   └── modules
│   │   │       ├── scenarios
│   │   │       │   ├── muzeum_extended
│   │   │       │   └── muzeum_habsbourg
│   │   │       └── scenes
│   │   │           ├── muzeum_extended
│   │   │           ├── muzeum_habsbourg
│   │   │           └── showcase_room
```

```
│   │   ├── models
│   │   ├── services
│   │   └── ...
│   ├── main.ts
│   ├── polyfills.ts
│   ├── test.ts
│   └── ...
├── angular.json
├── karma.conf.js
├── package.json
├── tsconfig.spec.json
└── ...
```

Testy komponentov

Testy komponentov sú zamerané na overenie správnej funkčnosti a interakcie jednotlivých častí používateľského rozhrania. Tieto testy zabezpečujú, že komponenty správne reagujú na vstupy, vykresľujú sa podľa očakávaní a správne manipulujú s DOM. Bolo veľa komponentov z predchádzajúcich prác a bolo pridaných ešte viac testov, takže tu sú súbory najdôležitejších prvkov s názvami testovacích súborov podľa testovaných prvkov:

- **auth-callback.component.spec.ts** – overuje spracovanie OAuth callbacku, najmä načítanie tokenu z query parametrov, volania metód služieb `utils.service`, `backend.service` a `transfer.service`, navigáciu na profilovú stránku pri úspechu a korektne ošetrenie scenára, keď token v URL chýba.

- **forgot-password.component.spec.ts** – testuje validácie e-mailového poľa, odoslania požiadavky na backend cez `backend.service`, zobrazenia úspešnej alebo chybovej správy cez `MatSnackBar` a zabránenia opakovaným požiadavkám počas cooldown obdobia.

- **reset-password.component.spec.ts** – overuje získanie resetovacieho tokenu z `ActivatedRoute`, validáciu hesla a potvrdenia, volanie `resetPassword` s korektnými parametrami, navigáciu späť na prihlasovaciu stránku a zobrazenie potvrdzujúcej správy, ako aj spracovanie chýb pri neplatnom tokenu či hesle.

- **auth.guard.spec.ts** – overuje či bol správne zísťený autentifikačný stav používateľa pomocou `auth.service` a povolí alebo zamietne prístup k chrá-

neným adresám, vrátane presmerovania na prihlasovaciu stránku pri neautorizovanom prístupe.

- **auth.interceptor.spec.ts** – kontroluje, či pri odosielaní HTTP požiadaviek vkladá do hlavičky `Authorization` správny JWT token, a zároveň zabezpečuje správne spracovanie odpovedí s chybovými stavmi (napr. 401 Unauthorized).
- **role.guard.spec.ts** – kontroluje, či na základe údajov z `auth.service` správne vyhodnocuje požadované roly pre danú routu, povoľuje prístup používateľom s prislúchajúcou rolou a v opačnom prípade presmeruje alebo zobrazí chybové hlásenie.
- **create-quiz.component.spec.ts** – overuje inicializáciu komponentu `CreateQuizComponent`, vrátane nastavenia reaktívneho formulára s príslušnými validátormi (napr. povinné polia, minimálna dĺžka), volanie metódy `quizService.createQuiz()` pri úspešnom odoslaní formulára, navigáciu na zoznam kvízov po úspechu a správne zobrazenie chybových hlásení pri zlyhaní služby.
- **create-scene.component.spec.ts** – testuje najmä proces nahrávania a náhľadu fotografie cez `FileReader`, viazanie vstupov pre mapy scén, formovú validáciu názvu a popisu a správanie pri chybových odpovediach servera.
- **create-task.component.spec.ts** – kontroluje nastavenie a validáciu formulára v `CreateTaskComponent`, výber súvisiacich entít, ako scenár a scéna, cez `MatSelect`, vymazanie formulára po úspechu a zobrazenie upozornení cez `MatSnackBar` v prípade neúspechu.
- **dashboard.component.spec.ts** – overuje načítanie súhrnných štatistík (počet scenárov, scen, úloh, aktívnych sedení) prostredníctvom `dashboardService.getOverview()`, spracovanie Observable dát, vykreslenie výsledkov v šablóne a zobrazenie indikačnej animácie počas načítavania alebo chybového stavu.
- **confirmation-dialog.component.spec.ts** – testuje správné zobrazenie nadpisu, správy a tlačidiel „Potvrdiť“/„Zrušiť“, injekciu `MatDialogRef` a odovzdanie výsledku `true` alebo `false` späť volajúcemu komponentu pri kliknutí.
- **group-sessions.component.spec.ts** – testuje načítanie relácií úloh pre danú skupinu, spracovanie paginácie a filtrovania (pomocou `MatPaginatorStub`),

zobrazenie tabuľky sedení a správanie pri prázdnych dátach či chybách služby.

- **group-settings.component.spec.ts** – kontroluje inicializácie reaktívneho formulára s poľami pre názov a popis skupiny, validátorov (povinné polia, maximálna dĺžka), odoslanie zmien, správne zobrazenie úspešnej alebo chybovej notifikácie a vymazanie formulára po úspechu.

- **groups-dashboard.component.spec.ts** – overuje, či načíta prehľad všetkých skupín a ich štatistiky, správne spracuje Observable reťazce, vykreslí grafické prvky (počet skupín, aktívne session) a zobrazenie indikátora načítavania počas asynchrónnej operácie.

- **mat-paginator.stub.ts** – poskytuje simulovanú implementáciu `MatPaginator` pre testy komponentov s tabuľkovou pagináciou, vrátane vlastností `pageSize`, `pageIndex`, `length` a EventEmitter-u `page`, čím umožňuje overiť logiku reakcie komponentu na zmeny stránkovania bez závislosti na reálnom Angular Material module.

- **task-history.component.spec.ts** – overuje, či správne načíta históriu relácií úlohy, zobrazuje tabuľku s dátami (čas spustenia, ukončenia, trvanie), spracováva pagináciu a reaguje na prázdne výsledky či chyby služby.

- **task-history-extended.component.spec.ts** – testuje kombinovanie dát relácií s podrobnými informáciami o používateľoch a scenároch, validitu transformácií na rozšírené DTO pre zobrazenie komplexných riadkov a manipuláciu s asynchrónnymi Observable tokmi.

- **task-history-statistics.component.spec.ts** – kontroluje výpočet agregovaných štatistík (priemerné, maximálne a minimálne časy relácií), načítania dát cez `taskService.getTaskSessionStats()` a korektné vykreslenie výsledkov do grafických prvkov alebo tabuľky.

- **user-profile.component.spec.ts** – overuje, či pri inicializácii zavolá `userService.getProfile()`, korektne naplní polia formulára s údajmi používateľa, spracuje ich zmenu a pri uložení zavolá `updateProfile()` s očakávaným zaťažením, vrátane zobrazenia potvrdenia alebo chybového hlásenia cez `MatSnackBar`.

- **user-settings.component.spec.ts** – testuje volanie `updateSettings()` pri zmene a správanie v prípade zlyhania volania.

- **users-dashboard.component.spec.ts** – kontroluje, či načíta súhrnné údaje o všetkých používateľoch, spracuje Observable toky, vykreslí tabuľku s metrikami (počet aktívnych/neaktívnych používateľov) a zobrazuje indikačný spinner počas načítania alebo chybové hlásenie pri zlyhaní služby.

Do tejto kategórie patria aj komponenty obsiahnuté vo virtuálnych scénach:

- **`scene.component.spec.ts`** – testuje inicializáciu a vytvorenie `scene.component`, overuje správne vykreslenie 3D scény a reakciu na užívateľské udalosti.
- **`muzeum_habsbourg.component.spec.ts`** – overuje správne nastavenie vstupných dát komponentu múzea, zobrazenie exponátov a navigačné funkcie v rámci histórie Habsburgovcov.
- **`showcase.component.spec.ts`** – testuje načítanie a spracovanie vstupných položiek do ukážkovej scény, ich správne zobrazenie a emisiu udalostí používateľských interakcií.

Testy služieb

Vo vrstve klienta Angular slúžia tri hlavné služby na správu autentifikácie, komunikácie s API a prenosu dát medzi komponentmi. Testy týchto služieb overujú, či sa JWT tokeny správne ukladávajú, či HTTP volania smerujú na očakávané koncové body a či sa cez observables emituje správny stav aplikácie.

- **utils.service.spec.ts** – overuje sa ukladanie a odstraňovanie JWT tokenu v lokálne úložisko, správne rozpoznanie prihláseného stavu vrátane vypršania platnosti tokenu a správanie metód `getDecodedAccessToken` a `tokenExpired` pri rôznych vstupoch .
- **backend.service.spec.ts** – testuje sa odosielanie HTTP požiadaviek pre registráciu, prihlásenie, prepisovanie hesla, načítanie a aktualizáciu profilu, pričom sa kontrolujú URL, metódy (GET/POST/PATCH), hlavičky (Content-Type, Authorization) a správne spracovanie odpovedí vrátane logovania URL pri prihlásení.
- **transfer.service.spec.ts** – overuje sa publikovanie hodnôt cez jednotlivé subjekty a `Observable` polia (`loginStatus`, `profileStatus`, `roleStatus`, `updatedUserStatus`, `createdGroupStatus`, `updatedGroupStatus`) pri volaní metód `changeStatus`, `changeProfile`, `changeUpdatedGroup()`

Testy typových modelov

Hoci modely zohrávajú skôr definičnú úlohu a majú málo funkcií, testy typových modelov overujú správnu inicializáciu a správanie jednotlivých typov, ako aj ich vzájomné prepojenie. Tieto testy zabezpečujú, že všetky typy sú správne definované a fungujú podľa očakávaní. V tejto časti sa nachádzajú nasledujúce testy typových modelov:

- **group.model.spec.ts**
- **group-tasks.model.spec.ts**
- **login-request.model.spec.ts**
- **profile-update.model.spec.ts**
- **registration-request.model.spec.ts**
- **scenario-preview.model.spec.ts**
- **scene-preview.model.spec.ts**
- **task-creation.model.spec.ts**
- **task-files.model.spec.ts**
- **task-names.model.spec.ts**
- **task-preview.model.spec.ts**
- **task-request.model.spec.ts**
- **task-session-finish-request.model.spec.ts**
- **task-session-info.model.spec.ts**
- **user-profile.model.spec.ts**

B.3 Testovanie komponentov virtuálneho prostredia

Medzi časti virtuálneho prostredia možno rozdeliť na všeobecné moduly komponentov A-Frame a jednotlivé komponenty scény vrátane ich skriptov (viditeľné vo výpise B.5).

Výpis B.5: Štruktúra súborov klientskej časti v Angular

```
LirkisEduVePn-UI
├── ...
├── coverage
├── src
│   ├── app
│   │   ├── components
│   │   │   ├── ...
│   │   │   └── tasks
│   │   │       ├── entities
│   │   │       ├── js
│   │   │       │   ├── components
│   │   │       │   ├── models
│   │   │       │   └── modules
│   │   │       ├── scenarios
│   │   │       │   ├── muzeum_extended
│   │   │       │   └── muzeum_habsbourg
│   │   │       └── scenes
│   │   │           ├── muzeum_extended
│   │   │           ├── muzeum_habsbourg
│   │   │           └── showcase_room
│   │   └── ...
│   └── ...
├── angular.json
├── karma.conf.js
├── package.json
├── tsconfig.spec.json
└── ...
```

Moduly implementujú základné funkcie pre prácu s Petriho sieťami a zaznamenávanie udalostí v A-Frame komponentoch. Modul `cpnLoader.mjs` poskytuje metódy na konverziu XML dokumentov CPN do JS objektov vrátane prevodu XML na JSON a extrakcie miest, prechodov a hrán so správnym rozpoznaním orientácií a hmotností.

- `cpnLoader.spec.ts` – overuje správnu funkciu metódy `coerceArray` pri rôznych vstupoch a kompletný rozbor miest, prechodov a hrán Petriho sieťe, vrátane obojstranných hrán a čistenia reťazcov pred konverziou na čísla.

- `petriNet.spec.ts` – testuje konštruktor triedy `PetriNet`, vyhľadávanie miest a prechodov, získavanie označení, kontrolu povolenosti prechodov a vyhľadávanie vstupných aj výstupných hrán pre daný prechod.
- `petriNetLoader.spec.ts` – overuje, že funkcia `loadXMLDoc` správne konvertuje jednoduchý PNML XML reťazec na objekt so zohľadnením miest, prechodov a hrán s ich zdrojmi, cieľmi a hmotnosťami.
- `serverLogger.spec.ts` – simuluje konštruktor Parse.Object, testuje, či `createParseSession` a `createParseFiringAttempt` nastavujú správne vlastnosti, volajú `save()` a zaznamenávajú úspech či chybu na konzolu.
- `userActivityLogger.spec.ts` – kontroluje, či klientske API moduly `createSession`, `createFiringAttemt`, `endSession` a `getFiredTransitionsFromSession` odosielajú HTTP požiadavky s očakávanými metódami, hlavičkami a telami, správne vracajú JSON alebo vyhadzujú chyby, a či sa pri ukončení relácie odstraňuje položka z `localStorage`.