FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Analýza a vylepšení testování backendové části projektu Uniqway
Student:	Andrej Ohrablo
Vedoucí:	Ing. Václav Jirovský, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2023/2024

Pokyny pro vypracování

Cílem práce je analyzovat a najít vylepšení nebo rozšíření v procesu testování kódu v monolitické Java aplikaci pro backend projektu sdílení vozidel Uniqway. Vyberte si vhodný případ vylepšení (např. doplnění stubbingu databáze nebo platební brány, refactoring integračních testů) a rozšíření (např. zavedení regresních testů a jejich automatického generování, automatizovaný proces pro sbírání statických metrik a statement nebo branch coverage). Při realizaci práce postupujte v následujících krocích:
1. Důkladně analyzujte současný testovací proces na backendu projektu Uniqway s důrazem na jeho zdokonalení a zjednodušení.
2. Na základě analýzy navrhněte možná vylepšení a rozšíření, odhadněte jejich dopad na fungování systému.
3. Z dostupných zdrojů prostudujte potřebnou problematiku a vyberte vhodné nástroje k implementaci navrhnutých vylepšení a rozšíření.
4. Na základě návrhu implementujte funkční prototyp vylepšení a rozšíření.
5. Řešení připravte na integraci do projektu.
6. Navrhněte další možná budoucí vylepšení.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 6. února 2023 v Praze.

Bakalárska práca

ANALÝZA A VYLEPŠENÍ TESTOVÁNÍ BACKENDOVÉ ČÁSTI PROJEKTU UNIQWAY

Andrej Ohrablo

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedúci: Ing. Václav Jirovský, Ph.D.
11. mája 2023

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam výpisov kódu

Chcel by som poďakovať svojmu vedúcemu Ing. Václavu Jirovskému, Ph.D. a členom tímu Uniqway za ich cenné rady a konzultácie pri tvorbe tejto práce. Tiež ďakujem mojej rodine za všetku podporu počas môjho celého štúdia.

Vyhlásenie

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona a to na dobu určitou do skončení trvání ochrany dle Smlouvy. Nakládání s předloženou prací se řídí Smlouvou o spolupráci uzavřenou v návaznosti na spolupráci mezi Českým vysokým učením technickým v Praze a společností ŠKODA AUTO a.s. a Smart City Lab s.r.o na výzkumném projektu „CarSharing pro vysokoškolské studenty", uveřejněné v registru smluv na adrese https://smlouvy.gov.cz/smlouva/5973503. Jsem vázán Smlouvou o zachování mlčenlivosti, že nezpřístupním třetí osobě důvěrné informace, které jsem při své práci na Projektu získal.

V Praze dňa 11. mája 2023 .....

Abstrakt

Táto bakalárska práca sa zaoberá analýzou testovacieho procesu na backende systému spoločnosti Uniqway a jeho možných vylepšení. Práca pokrýva analýzu backendu aplikácie a súčasný stav testovania. Sú tu rozobrané postupy testovania jednotlivých častí a používané typy testov. V návrhovej časti sú predstavené niektoré možnosti vylepšenia terajšieho procesu. Praktická časť sa následne zaoberá implementáciou prototypu vybraných vylepšení.

Klíčová slova Uniqway, testovací proces, migračné skripty, Play framework, regresné testy, Randoop, Code coverage

Abstract

This bachelor thesis deals with the analysis of Uniqway's backend system testing process and its possible improvements. The thesis covers the analysis of the application backend and the current state of testing. The testing procedures of the different parts and the types of tests used are discussed. In the design section, some options for improving the current process are presented. The practical part then deals with the prototype implementation of selected improvements.

Keywords Uniqway, testing process, migration scripts, Play framework, regression tests, Randoop, Code coverage

Zoznam skratiek

API	Application Programming Interface
CICD	Continuous Integration and Continuous Deployment
DAO	Data Access Object
DBMS	Database Management System
DTO	Data Transfer Object
E2E	End-to-End
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
JVM	Java Virtual Machine
JSON	JavaScript Object Notation
MVC	Model-View-Controller
RAM	Random Access Memory
RDBMS	Relational Database Management System
REST	Representational State Transfer

Zoznam termínov

code review	Proces kontroly kódu
constraint	Obmedzenie alebo pravidlo, ktoré musí byť dodržané
Docker kontajner	Izolované prostredie v aplikácií Docker
framework	Sada nástrojov na vývoj softwaru
geofencing	Virtuálne ohraničenie geografickej oblasti
Git	Distribuovaný systém pre správu verzií
merge request	Žiadosť o zlúčenie zmien v kóde
mock	Náhrada reálneho objektu na testovanie
open source	Softvér s verejným prístupom k zdrojovému kódu
rollback	Proces vrátenia systému do predošlého stavu
plugin	doplnkový modul aplikácie
silno previazané	Označenie pre objekty množstvom vzájomných závislostí
trigger	Procedúra, ktorá spúšťa automatizovaný proces

Úvod

Anglické slovo „bug“ pre popis toho, že systém nefunguje tak ako má vzniklo ešte pred prvým počítačom a pôvodne popisovalo mechanické poruchy[1]. Bolo bežnou súčasťou inžinierskeho slovníka už v roku 1870. Zachovalo sa niekoľko listov kde ho používal aj Thomas Edison a v takmer nezmenenom význame nás sprevádza aj dnes v digitálnom svete.

História softwarového vývoja je natoľko plná bugov, že existujú zoznamy iba tých mediálne známych. Za zmienku stojí prvá správa poslaná cez internet[2] alebo nedávny bug, kde chyba v jednom riadku kódu známeho herného programu mohla užívateľovi vymazať celý adresár [3]. Paradoxne k pocitu, že všetko digitálne funguje tak ako má a dnes sa na software sa môžeme spoľahnúť, dochádza v dobe kedy sú informačné systémy komplexnejšie ako kedykoľvek predtým.

Písanie testov sa ukázalo ako najlepšie riešenie pri dlhodobej prevencii chybovosti. Spolu s vývojom procesov pre písanie softwaru tak vznikali aj rozmanité procesy pre jeho testovanie. V poslednej dobe sa kód písaný pre testy začína stávať rovnako dôležitým ako samotný kód na ktorom software beží.

V tejto bakalárskej práci sa budem zaoberať testovaním backendu aplikácie, ktorá vám vie odomknúť auto a následne si od vás vypýta peniaze. Konkrétne ide o študentský projekt Uniqway, ktorý ponúka tzv. carsharing, teda možnosť požičať si auto napríklad pomocou mobilnej aplikácie.

V prvých kapitolách budem analyzovať terajší stav testovania kódu na backendovej časti, teda časti zodpovednej za funkčnosť samotného systému. Preskúmam rozsah a typy použitých testov, spôsob akým sa kód testuje ako aj kedy a koľko testov sa píše. Aj keď neexistuje žiadna univerzálna metóda pre hodnotenie existujúcich testov, pokúsim sa pozrieť na to, ako sa dodržiavajú zaužívané princípy ich písania.

Následne v návrhovej časti navrhnem niekoľko konkrétnych vylepšení, ktoré by sa dali v budúcnosti implementovať a v implementačnej časti potom niektoré vybrané návrhy implementujem.

Výsledok práce by mal sprehľadniť stav testovania v Uniqway a zároveň vylepšiť niektoré jeho časti. Tému som si zvolil z dôvodu že rieši problém na reálne používanej funkčnej aplikácii, v ktorej nie je tak jednoduché robiť zmeny. Zároveň sa jedná o často nedocenenú časť sofwarového inžinierstva, ktorá sa stáva stále aktuálnejšou.

Ciele práce

Cieľom tejto bakalárskej práce je zanalyzovať proces testovania kódu na backendovej časti projektu Uniqway, zistiť ktoré časti sú testované dostatočne a kde by sa dal tento proces vylepšiť, prípadne rozšíriť. Budú tu popísané využívané technológie a stručne naznačené fungovanie backendovej časti v rámci projektu Uniqway. Následne bude analyzovaný samotný proces testovania, dôjde k prieskumu spôsobu písania testov, popisu ich štruktúry a dosahu v rámci aplikačného kódu.

Dôvodom pre túto analýzu je možnosť vylepšiť testovací proces tak, aby dochádzalo k menšej chybovosti pri vývoji projektu alebo pri behu aplikácie. Rovnako v niektorých prípadoch pôjde o pokus skúsiť proces tvorby testov zjednodušiť, otestovať nové časti projektu, vylepšiť už existujúce testy, či zvýšiť počet informácií o projekte získavaných testovaním.

V návrhovej časti budú predstavené konkrétne návrhy zmien v testovacej časti, ktoré sa budú snažiť už existujúci proces vylepšiť. Z týchto návrhov bude vybraný vhodný prípad pre rozšírenie, resp. vylepšenie, pre ktorý bude zostrojený konkrétny návrh jeho implementácie. Po vybraní vylepšenia na implementáciu prebehne detailný prieskum jeho problematiky a identifikácia dostupných nástrojov na zostrojenie riešenia. Na základe týchto zistení bude vytvorený konkrétny zoznam implementačných zmien.

V praktickej časti potom bude na základe podrobného návrhu implementovaný funkčný prototyp tak, aby bol pripravený na budúce zasadenie do projektu.

- Dôkladne analyzujte súčasný testovací proces na backende projektu Uniqway s dôrazom na jeho zdokonalenie a zjednodušenie.
- Na základe analýzy navrhnite možné vylepšenia a rozšírenia, odhadnite ich dopad na fungovanie systému.
- Z dostupných zdrojov preštudujte potrebnú problematiku a vyberte vhodné nástroje k implementácii navrhnutých vylepšení a rozšírení.
- Na základe návrhu implementujte funkčný prototyp vylepšení a rozšírení.
- Riešenie pripravte na integráciu do projektu.
- Navrhnite ďalšie možné budúce vylepšenia.

Kapitola 1
Analýza

Táto kapitola sa zoberá predstavením služby Uniqway a stručným opisom toho, ako funguje jej backendová časť. Prvá sekcia približuje myšlienku tohoto študentského projektu. V dodatku prvej sekcie sú tiež naznačené špecifiká aktuálnej situácie. Ďalšie sekcie potom predstavujú technológie a softwarové postupy používané pri vývoji backendovej časti aplikácie.

1.1 Služba Uniqway

Služba Uniqway je študentský carsharingový projekt, teda projekt zameraný na zdieľanie vozidiel medzi uživateľmi tejto služby. Služba si kladie za cieľ zvýšiť mobilitu študentov a zamestnancov, ktorí nemajú prístup k autu a poskytnúť im cenovo dostupné, pohodlné a udržateľné dopravné riešenie [4].

Na vývoji a chode tejto služby so sídlom v Prahe sa podieľajú študenti z troch pražských univerzít, konkrétne z Českého vysokého učení technického, České zemědělské univerzity a Vysoké školy ekonomické. Projekt Uniqway sa od svojho vzniku v roku 2017 značne rozrástol a stále sa rozvíja, avšak zatiaľ o ňom platí že je uzavretou komunitou, čo znamená, že službu môžu využívať iba študenti a zamestnanci českých vysokých škôl[4].

1.1.1 Aktuálna situácia

Projekt Uniqway od svojho vzniku fungoval ako tzv.„free-floating carsharing“, teda služba kde je flotila áut od Uniqway rozdelená naprieč celým mestom a uživatelia si môžu autá vyzdvihnúť a vrátiť ich kdekoľvek v určenej oblasti.

O flotilu áut ako aj podporu prevádzky služby sa starala spoločnosť ŠKODA AUTO, ktorá však začiatkom roku 2023 od projektu odstúpila. Služba Uniqway tým prišla o hlavného sponzora a svoju flotilu áut. Preto pozastavila svoje fungovanie na dobu neurčitú.

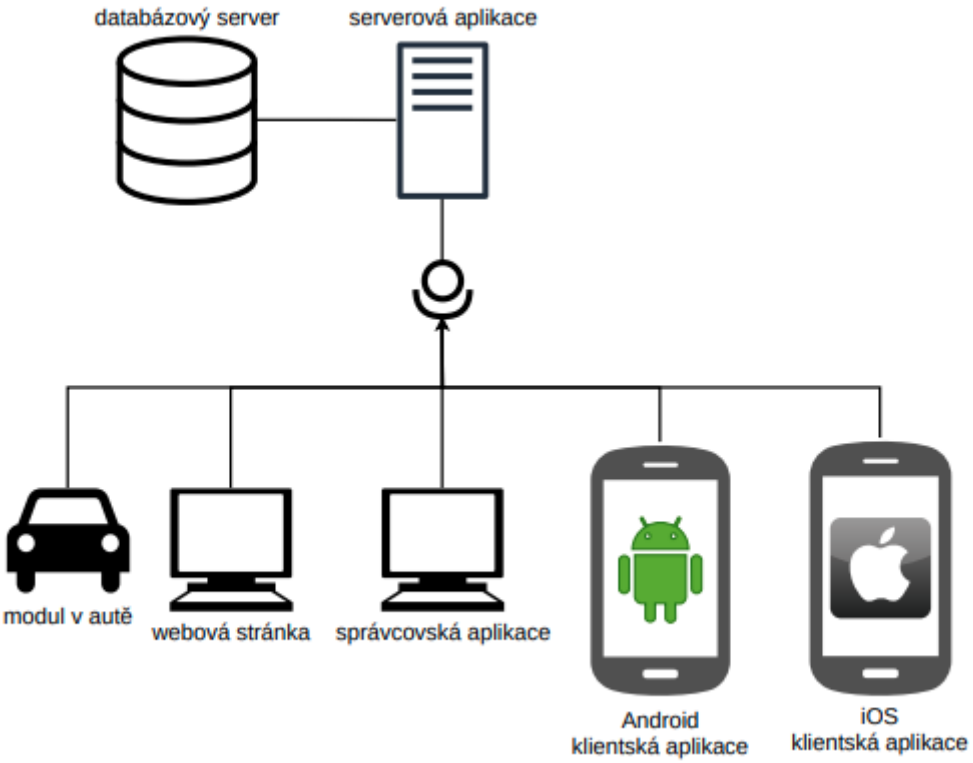
1.2 Prehľad systému

Základ systému tvorí serverová aplikácia, ktorá zároveň tvorí aj najväčšiu časť na backende systému. Tá sa stará o zber dát, všetky logické výpočty, komunikáciu cez koncové body API a integráciu aplikácii z tretích strán potrebných pre fungovanie. Napríklad zodpovedá za výpočet ceny na základe dát zaslaných z automobilov alebo integráciu platobnej brány.

Serverová aplikácia disponuje pripojením na databázový server, resp. databázu v ktorej sú uložené dáta o uživateľoch, autách a všetky ostatné informácie potrebné pre chod aplikácie.

Komunikácia medzi backendovou časťou s databázou a zvyškom systému prebieha pomocou rozhrania REST API. Na tieto koncové body API sú potom napojené mobilné uživateľské aplikácie pre Android a iOS, ktoré slúžia na všetko od registrácie až po rezerváciu a vyzdvihnutie automobilu. Ďalej má Uniqway ešte správcovskú aplikáciu pre administrátorov na správu dát a svoje webové stránky, ktoré okrem informovania o novinkách ponúkajú pre uživateľov podobnú funkčnosť ako mobilné aplikácie.

V období, keď ešte Uniqway poskytovala svoje služby, disponovala niekoľkými desiatkami áut značky ŠKODA vybavených špeciálnym modulom na komunikáciu so serverom pomocou API. Pomocou tohoto modulu mohol uživateľ napr. na diaľku auto odomknúť a zamknúť. Počas jazdy modul odosielal na server množstvo dát napr. o svojej polohe.



Obr. 1.1 Schéma systému Uniqway. Obrázok z diplomovej práce *Transformace systému z monolitické architektury do architektury mikroslužeb*, Petr Prouza, ČVUT FIT [5]

1.3 Backend

V nasledujúcej kapitole je stručne predstavené fungovanie backendovej časti aplikácie Uniqway.

Ako bolo v 1.2 spomenuté, backend tvorí akýsi centrálny článok starajúci sa o všetky potrebné výpočty, kontrolu dát a akúkoľvek inú logickú funkcionalitu potrebnú pre správny beh aplikácie.

Keďže je to zároveň aj jediný prístupový bod k databázi, stará sa aj o všetok zápis a mazanie dát. Okrem iného je tiež zodpovedný za vznik a všetky úpravy jej štruktúry. V prípade relačnej databáze to znamená správne počiatočné nastavenie stĺpcov, riadkov a všetkých kontrolných podmienok. V neposlednom rade poskytuje REST API rozhranie umožňujúce pripojenie rôznych modulov a aplikácií.

1.3.1 Monolitická architektúra

Aplikácia je v súčastnosti vyvíjaná na monolitickej architektúre. To znamená že sa skladá z jedného veľkého celku, ktorého komponenty sú spolu silno previazané a ťažko oddeliteľné. Zdrojový kód týchto komponent je uložený na jednom mieste a bežia v spoločnom prostredí[6].

To so sebou prináša niekoľko výhod ako napríklad priamočiare pridávanie nových funkcií, čo bolo pri vzniku spoločnosti veľkou výhodou.

1.3.1.1 Nevýhody monolitickej architektúry

Popularita monolitickej architektúry však v súčastnosti upadá. Môžu za to najmä nevýhody spojené s údržbou a škálovateľnosťou väčších celkov. Silná previazanosť znamená že aj malé zmeny v jednej časti kódu môžu mať za následok nutnosť upraviť množstvo susedských tried.

Znamená to aj, že silná previazanosť znemožňuje písať komplexné testy, keďže na nasimulovanie jednej triedy je potrebné potrebné vyriešiť závislosti a návratové hodnoty niekedy až z desiatok iných tried. Ako neskôr ukazuje kapitola 2.6, implementácia integračných testov v aplikácii značne zaostáva. S ohľadom na budúce zmeny v kóde sa integračné testy ukazujú aj ako ťažko udržateľné práve z dôvodu nutnosti simulácie množstva tried pre otestovanie menšieho aplikačného celku.

Okrem toho je s monolitickou architektúrou spojená aj nutnosť pri každej zmene celý projekt znova kompilovať a znova nasadiť. Pritom jej postupný rast často vytvára viazanosť na jednu konkrétnu technológiu a znemožňuje tak prenesiteľnosť aplikácie na inú platformu[6].

1.3.1.2 Prechod na modulárnu architektúru

Z dôvodov popísaných v 1.3.1.1 sa v blízkej budúcnosti očakáva dlho plánovaný prechod na modulárnu architektúru, ktorá by riešila problémy s previazanosťou.

Keďže sa tento prechod ešte neuskutočnil, nie je zrejmé presne aké technológie a postupy sa budú v novej verzii vyskytovať. Pravdepodobne sa však ostáva pri rovnakom programovacom jazyku a rovnakom frameworku ako doteraz 1.3.5.

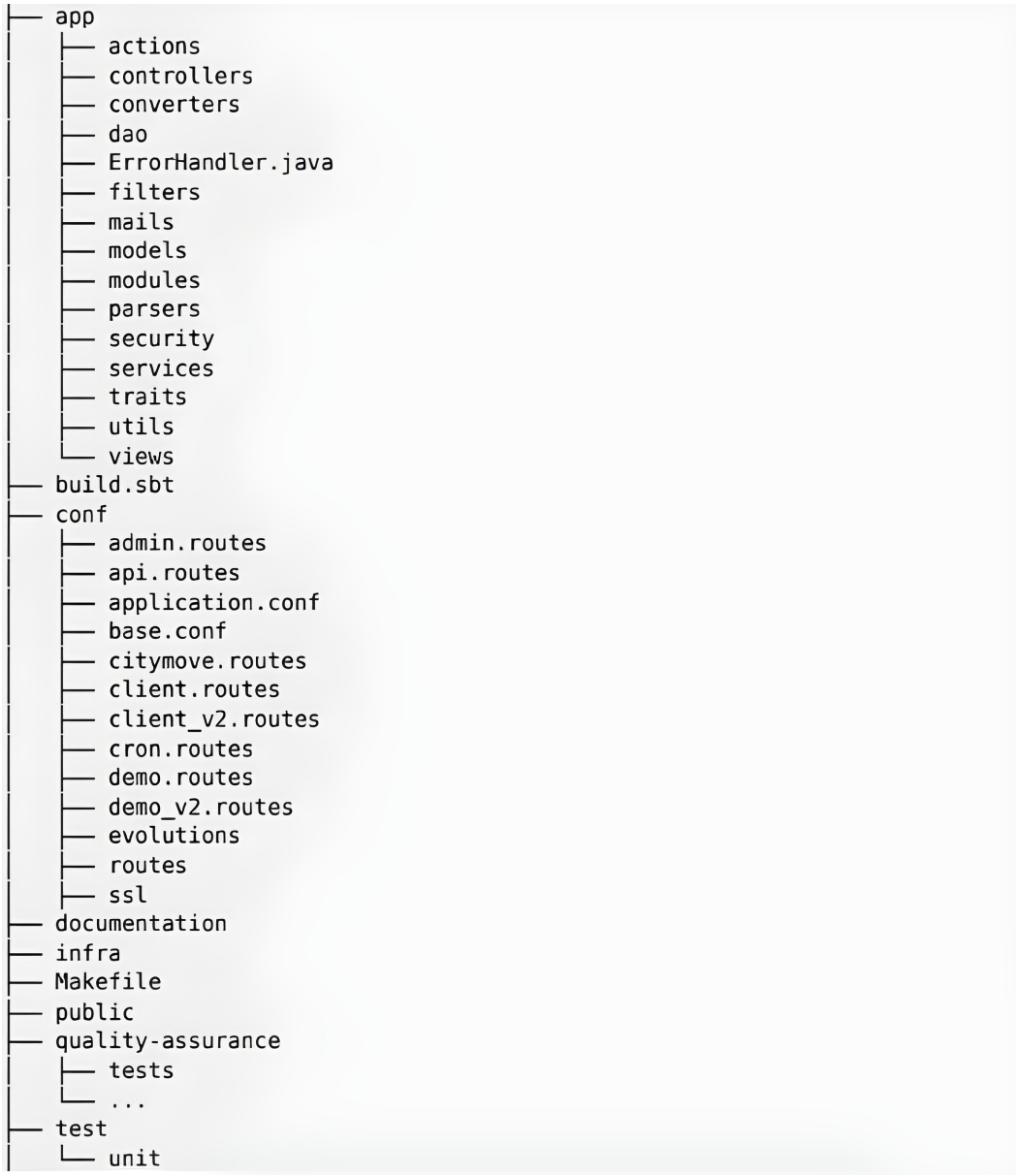
Rovnako ešte nie je známa ani forma konkrétneho rozdelenia na jednotlivé moduly. Z pohľadu testovania to znamená, že ideálnym časom pre doplnenie komplexnejších integračných testov bude práve čas prechodu na novú architektúru.

1.3.2 Umiestnenie kódu

Uniqway backend je v súčastnosti celý udržiavaný v jednom Git repozitári, kde sa nachádza všetok produkčný kód. Postupné zmeny aplikácie sú potom adresované pomocou *merge requestu*, ktorý sa po schválení cez proces code review pridá k pôvodnému kódu. Tým de facto dôjde k novej verzii celej aplikácie, ktorá je vždy označená novým číslom.

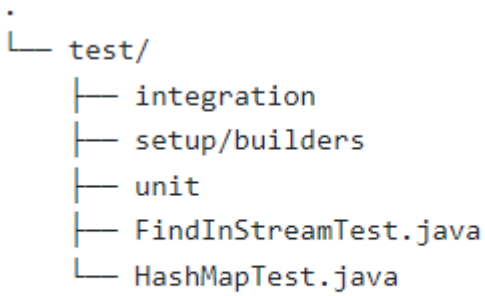
Pre lepšiu predstavu sa nižšie nachádza obrázok skrátenej adresárovej štruktúry 1.2 a kompletné zloženie adresáru *test* 1.3.

Monolitický návrh, vďaka ktorému je celý projekt uložený v jednom adresári môže vytvárať dojem, že sa jedná o pomerne jednoduchú aplikáciu. V realite však ide o pomerne komplikovaný projekt s desiatkami tisícov riadkov a viac než tisíckou súborov. Rozsah práce v čase písania tejto práce je priblížený v tabuľke 1.1.



Obr. 1.2 Skrátená adresárová štruktúra z internej dokumentácie Uniqway

- **app** - adresár s produkčným kódom
- **build.sbt** - konfiguračný súbor pre sbt
- **conf** - adresár s konfiguračnými súbormi
- **documentation** - adresár s dokumentáciou kódu
- **quality-assurance** - adresár so súbormi na kontrolu kvality, mimo iné sa tu nachádzajú Python testy pre API
- **test** - adresár, kde se nachádzajú Java testy zdrojového kódu



Obr. 1.3 Adresár *test*

1. **integration** - adresár s integračnými testami
2. **setup/builders** - adresár s pomocnými triedami na tvorbu Java objektov
3. **unit** - adresár s unit testami, tu sa nachádza drvivá väčšina napísaných testov
4. **FindInStreamTest.java** - test funkčnosti java streamov
5. **HashMapTest.java** - test funkčnosti java hašovacej mapy

Štatistiky kódu projektu		
Celkový počet súborov	1599	(100%)
Počet Scala súborov	83	(5.2%)
Počet Java súborov	1516	(94.8%)
Celkový počet riadkov	141,540	(100%)
Počet riadkov kódu	90,142	(63.7%)
Počet riadkov komentárov	15,986	(11.2%)
Počet prázdnych riadkov	35,492	(25.1%)
Počet riadkov s jednou zátvorkou	15,827	(11.2%)
Počet znakov	3,495,067	(100%)
Počet znakov kódu	3,180,391	(91.0%)
Počet znakov komentárov	314,676	(9.0%)
Priemerná dĺžka súboru	88 riadkov	

Tabuľka 1.1 Rozsah backendu Uniqway. Štatistiky získané pomocou balíčku sbt-stats[7]

1.3.3 Hlavné funkcionality

Aplikácia takého rozsahu má v sebe nesmierne množstvo funkcionalít. Tie by sa dali rozdeliť na niekoľko hlavných kategórií podľa toho akej časti systému sa venujú.

1. **Uživatelia** - Patrí sem skupina funkcií starajúcich sa o správu užívateľských kont. Príkladmi sú skupiny funkcií pre registráciu, prihlasovanie, uživateľské oprávnenia alebo správu nahraných dokumentov.

2. **Posielanie oznamov** - Funkcie starajúce sa o automatizované posielanie emailov a notifikácii. Využívajú sa napríklad na zasielanie zľavových kódov alebo vystavenie dokladu za platbu.

3. **Správa áut** - Jedná sa o veľké množstvo funkcií spojených s prevádzkou samotných automobilov. Sú to funkcie, ktoré komunikujú s hardvérovým modulom v umiestneným v aute. Príkladom sú funkcie sledujúce polohu alebo stav paliva v aute.

4. **Platba za jazdy** - Patria sem funkcie pre výpočet ceny jazdy, spracovávanie zľavových kupónov alebo integrácia platobnej brány.

5. **Správa rezervácií** - Skupina funkcií, ktorá má za účel spravovať jednotlivé uživateľské rezervácie.

6. **Obchod s odmenami** - Jedným zo spôsobov ako nalákať nových uživateľov je poskytovať im body za každú prejdenú jazdu. Tie môžu následne vymeniť za odmeny v obchode s odmenami. Tieto funkcie sa starajú o odmeňovanie uživateľov a o nákupy vo virtuálnom obchode.

7. **Ostatné** - Backend samozrejme disponuje ešte množstvom funkcií, ktoré nepatria do ani jednej zo spomenutých kategórií. Príkladmi sú funkcie na geofencing, tvorbu účteniek alebo integrácie s ďalšími aplikáciami ako je napr. citymove.

1.3.4 Java

Jednou z najskorších volieb pri písaní novej aplikácie je nutnosť vybrať si ten správny programovací jazyk. Pri backende serverovej aplikácie Uniqway táto voľba padla jazyk Java[8]. Konkrétne už dlhšiu dobu aplikácia funguje na verzií s názvom Java 11, ktorá vyšla v septembri roku 2018. V začiatkoch projektu sa teda jednalo o jednu z najnovších podporovaných verzií.

Jazyk Java, je už dlhú dobu populárnou voľbou pri písaní backendových a enterprise aplikácií. Je to vysokoúrovňový programovací jazyk, ktorý bol vyvinutý v roku 1995 firmou Sun Microsystems (následne kúpenou spoločnosťou Oracle Corporation)[8].

Java je navrhnutá tak, aby bola platformovo nezávislá, čo znamená, že programy napísané v tomto jazyku môžu byť spustené na akomkoľvek počítači alebo zariadení s nainštalovanou Java Virtual Machine (JVM), bez ohľadu na operačný systém[8].

Je objektovo orientovaná, čo znamená, že všetky prvky jazyka, ako sú triedy, objekty, metódy, premenné, atď., sú objektami. Ďalšou jeho vlastnosťou je silná typová kontrola, čo znamená, že každá premenná alebo hodnota má presne definovaný typ, ktorý sa počas behu programu nemení[8].

Výhodou popularity Javy je aj fakt, že je vyučovaná na veľkej časti škôl so zameraním na programovanie. Výnimkou nie je ani ČVUT, kde sa Java vyskytuje v predmetoch povinných pre viaceré obory, čo je pre projekt ako Uniqway, ktorý je z veľkej časti vyvíjaný študentami veľkou výhodou.

1.3.5 Play Framework

K popularite Javy sa viaže aj fakt, že obsahuje veľké množstvo knižníc a frameworkov, ktoré zjednodušujú vývoj a umožňujú programátorom rýchlejšie a efektívnejšie vytvárať aplikácie.

Jedným z nich je aj Play framework[9] (tiež iba Play alebo Java Play) slúžiaci na tvorbu webových aplikácií v jazykoch Java a Scala používaný v Uniqway. Ide o open-source projekt s majúci za cieľ zjednodušiť vývoj a poskytovať vývojárom moderné a efektívne nástroje pre tvorbu rýchlych, škálovateľných a flexibilných webových aplikácií[9].

Vďaka balíčku Akka, na ktorom je Play postavený je z hľadiska využitia výpočetných zdrojov pomerne nenáročný a poskytuje jednoduchú škálovateľnosť. Play Framework používa softwarovú architektúru Model View Controller (MVC), čo umožňuje oddeliť rôzne zložky aplikácie, ako je logika, prezentačná a dátová vrstva. Okrem toho disponuje celou škálou funkcionalít ako plná podpora REST služieb alebo hot reload umožňujúci vidieť nové zmeny za behu aplikácie[9].

Pre študentský projekt je výhodou, že Play framework je pomerne jednoduchý na použitie a na nových programátorov tak nekladie zbytočne vysoké nároky. Veľká väčšina základných využití je riadne zdokumentovaná aj s ukážkovým použitím v zdrojovom kóde. Play navyše podporuje integráciu s veľkou časťou Java ekosystému, čo umožňuje vývoj rozličných typov aplikácií využívajúcich množstvo ďalších knižníc a frameworkov. Príkladmi firiem využívajúcich Play sú napr. EA, LinkedIn, Samsung alebo Verizon[9].

1.3.6 Komunikácia v rámci služby

Implementáciu jednotlivých funkcionalít je vo väčších aplikáciach nutné uskutočniť pomocou návrhových vzorov, ktoré nám poskytujú overený návod ako spoľahlivo a jednoducho písať aplikačný kód tak aby bol v budúcnosti ľahko udržiavateľný.

Uniqway sa pre veľkú väčšinu funkcionalít drží princípov z návrhového vzoru MVC (Model View Controller), ktorý implementuje pomocou frameworku Play 1.3.5.

Ako príklad implementácie funkcionality na backende Uniqway môžeme použiť požiadavok o všetkých záznamoch jázd užívateľa - **ride** 1.4.

Celý proces začína vystaveným koncovým bodom **API**, z ktorého príde požiadavok GET vo formáte JSON. V praxi sa to dá predstaviť ako uživateľ klikajúci na tlačítko pre zobrazenie jázd.

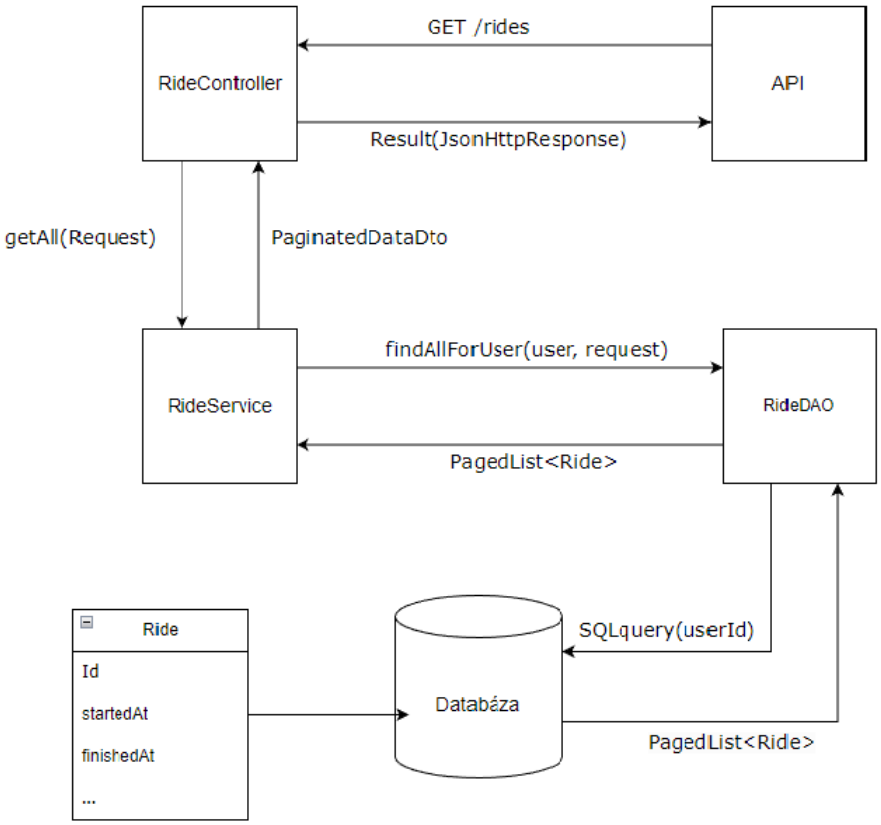
Požiadavok potom prevezme trieda nazývaná **RideController**, ktorá má za úlohu spracovávať tieto požiadavky volať príslušné funkcie na ďalšej vrstve. Tam je v tomto prípade trieda **RideService**, ktorá na základe parametrov predaných z controlleru zavolá správnu kombináciu funkcií komunikujúcich s databázou. Druhou podstatnou úlohou service tried je konverzia návratových hodnôt do formátu DTO. Ten slúži práve na komunikáciu medzi service a controller vrstvami.

O vyberanie dát z databázy sa potom stará trieda s názvom **RideDao**. Tá v tomto prípade preloží zadaný parameter - Id užívateľa, na SQL požiadavok. Výsledná odpoveď následne prebubláva cez rôzne formáty návratových hodnôt až naspäť ku koncovému bodu a užívateľovi.

1.3.6.1 Ebean

Keďže manuálne písanie SQL požiadavkov pre stovky rôznych funkcií by bolo značne nepraktické, backendový server využíva open source framework Ebean [10]. Ide o ORM framework umožňujúci mapovať relačné dáta uložené v databázach na objekty.

Jednotlivé objekty uložené v databáze tak majú v zložke models odpovedajúcu Java triedu, ktorá sa dá na tabuľku v databázi jednoducho namapovať. Toto mapovanie umožňuje frameworku Ebean vytvárať množstvo komplexných požiadavkov pre objekty uložené v databáze pomocou jednoduchej syntaxe. Príkladmi sú vkladanie, mazanie, vyhľadávanie, použitie agregačných funkcií či dokonca možnosť uskutočňovania transakcií s viacerými SQL požiadavkami naraz.



Obr. 1.4 Ilustrácia MVC na záznamoch jázd užívateľa

Výpis kódu 1.1 Ebean SQL požiadavok pre nájdenie všetkých záznamov s hodnotou userId

```
public PagedList<Ride> findAllForUser(Long userId, Request<?> request) {
        return createQueryFromRequest(request).where()
                .eq("exampletable.user.id", userId)
                .findPagedList();
    }
```

1.3.7 Databáza

Dôležitou častou backendu je aj databázový server. V terajšej konfigurácií sú dáta uložené v jednej hlavnej databázi, ktorá odpovedá na všetky odoslané požiadavky. Tá v súčastnosti funguje ako relačná databáza, využívajúca systém PostgreSQL.

1.3.7.1 PostgreSQL

PostgreSQL, alebo skrátene Postgres, je open source relačný databázový systém známy pre svoju spoľahlivosť a výkon. Medzi databázovými systémami sa aj po viac než 35 rokoch vývoja stále radí k populárnym voľbám s aktívnou komunitou[11].

PostgreSQL je ORDBMS, teda objektovo orientovaný relačný databázový systém, čo prináša oproti konkurencií niektoré výhody. Okrem ukladania dát v podobe riadkov a stĺpcov umožňuje aj ukladanie samotných objektov napríklad vo formátoch JSON alebo XML. Zaujímavou vlastnosťou je aj to, že tieto objekty podporujú dedičnosť podobne ako objekty v objektovo orien-

tovaných programovacích jazykoch. Často využívanou vlastnosťou je podpora tvorby vlastných komplexných dátových typov ako nadstavbu nad základnou SQL syntaxou[11].

Rovnako pre PostgreSQL existuje množstvo doplnkových balíčkov s vlastnými dátovými typmi. V Uniqway je toho príkladom rozšírenie PostGIS, ktoré pridáva dátové typy pre prácu s polohou a súradnicami. Vďaka tomu, že je PostgreSQL objektovo orientovaný, poskytuje aj jednoduché mapovanie jeho objektov pomocou ORM využiteľné napr. pri frameworkoch ako je Ebean 1.3.6.1.

1.3.7.2 Správa databáze cez migračné skripty

Pri veľkých a stále sa meniacich projektoch, akým je aj Uniqway, je databáza živou súčasťou projektu. Je potrebné ju aktualizovať podobne ako je potrebné aktualizovať zdrojový kód. V minulosti sa problém zmien v databázi riešil zmazaním starej verzie a nahraním novej. To je však pri komplexných projektoch, kde navyše môže viacero programátorov pracovať na rôznych častiach projektu naraz značne nepraktické. Pomerne ľahko môže dôjsť k navzájom nekompatibilným zmenám.

Jedným z potenciálnych riešení je verzovanie databáze pomocou migračných skriptov. Tieto skripty sa snažia o akési zachytenie a rozdelenie celého skriptu pre vytvorenie schémy databáze do menších častí [12]. Tie by mali odzrkadľovať postupné pridávanie zmien do databáze počas vývoja aplikácie. Z istého pohľadu sa tiež dá povedať, že sa v obmedzenej forme snažia o zachytenie histórie podobne ako sa o to snažia systémy pre správu verzií zdrojových kódov.

Hlavnou ideou je teda vznik väčšieho množstva malých skriptov pre úpravu databáze, ktoré by mali byť aplikované v sekvenčnom poradí za sebou. Sľubovanými výhodami tohoto systému je jednoduchšie hľadanie chýb, lepšia čitateľnosť pri veľkých databázových skriptoch, možnosť inkrementálneho písania skriptu alebo jednoduchšie riešenie konfliktov pri nekompatibilite nových zmien.

Správa databáze v Uniqway prebieha práve pomocou vyššie spomenutých migračných skriptov, ktoré má na starosti Play framework. V ňom sa tieto migračné skripty označujú ako **evolutions** (z anglického schema evolutions) [13]. V dnešnej dobe má týchto skriptov projekt niečo vyše 130.

1.3.8 Zostavovanie aplikácie

Používanie nástroja pre automatické zostavenie aplikácie je v dnešnej dobe prakticky nevyhnutnosťou. Tento nástroj sa stará o automatizáciu kompilácie, spúšťania a testovania zdrojového kódu pomocou jednoduchých príkazov. V závislosti na nástroji môže aj generovať dokumentáciu či distribúciu projektu. Zvyčajne je jeho súčasťou konfiguračný súbor popisujúci závislosti projektu a definujúci spôsob zostavenia. Príkladmi takýchto aplikácií pre Java projekty sú napríklad Maven[14], Gradle[15] alebo sbt[16].

1.3.8.1 sbt

Uniqway pre automatizované zostavovanie aplikácií využíva sbt. Ide o open source nástroj pre zostavovanie Scala a Java aplikácií vyvíjaný od roku 2008. Oproti ostatným nástrojom sa vyznačuje jednoduchou syntaxou a veľkou flexibilitou[16].

Výhodou pre Uniqway je aj natívna kompatibilita s Play frameworkom. Najväčšou nevýhodou však je fakt, že zatiaľ čo v Scale, pre ktorú bol nástroj pôvodne vytvorený, sbt používa 93.6% developerov, v Jave sa pravdepodobne jedná iba o jednotky percent[16]. To má zatiaľ za následok menší počet podporovaných doplnkových rozšírení.

Kapitola 2
Testovanie

Táto kapitola sa zaoberá prehľadom testovania a testovacích procesov na backendovej časti projektu Uniqway. Zo začiatku sú predstavené základné medotiky testovania softwarových aplikácií a jednotlivé typy testov. Ďalej sa kapitola zaoberá konkrétnym spôsobom testovania aplikácie v Uniqway. Je tu popísaný postup písania testov, kontrola ich kvality ako aj dodržiavaný postup pri priebežnom testovaní. Rovnako je tu analyzovaný zámer napísaných testov a pokrytie zdrojového kódu cieľovej aplikácie súčasnými testami. Nakoniec sa niekoľko sekcií venuje stavu testovania častí aplikácie, ktoré nespadajú pod jadro zdrojového kódu, ako testovanie koncových bodov API a správnosť databáze.

2.1 Ciele testovania všeobecne

Testovanie kódu je v úplnom základe akt skúmania správania sa programu za rôznych podmienok. Proces testovania by sa dal popísať pomocou jeho dvoch hlavných cieľov.

Prvý cieľ je deštrukčný, ide o snahu vymyslieť taký test, na ktorý program neodpovie správne alebo dôjde k chybe fungovania programu. Druhý cieľ by sa dal popísať ako validačný. Tu ide o snahu skontrolovať, či program plní očakávané funkcie správne a robí to, čo sa od neho očakáva[17].

Testovanie sa od svojich začiatkov rozvíjalo spoločne s vývojom softwarových aplikácií a v priebehu jeho histórie boli vyvinuté rôzne postupy na to ako testovať. Najciteľnejší je asi vývoj komplexnosti samotného testovacieho procesu, ktorý v priebehu času prešiel značnou automatizáciou.

Testovací proces má v súčastnosti takmer každá softwarová firma a nie je neobvyklé najímať celé oddelenia ľudí venujúcich sa iba testovaniu. Očakáva sa, že dnes testovanie softwaru okrem hľadania chýb zvyšuje spoľahlivosť, kvalitu a znižuje náklady spojené s údržbou softwarových aplikácií.

2.2 Ciele testovania na backende Uniqway

Backend projektu Uniqway tvorí logickú časť s množstvom komplikovaných funkcií od výpočtu ceny po správu užívateľských dát. Je teda rozumné že tieto funkcie by mali tvoriť jadro testov kontrolujúcich jeho bezproblémový chod. Okrem toho je však backend zodpovedný za mnoho iného. Nižšie je súhrn vybraných kľúčových častí, ktoré z analýzy vyplývajú ako dôležité pre testovací proces.

- **Logické funkcie** - Jadro celého backendu a zároveň jeho najrozsiahlejšia časť. Mala by jej

byť venovaná značná časť testovania. Overenie správnosti týchto funkcií je základom pre funkčnosť celého backendu.

- **MVC architektúra** - Ďalšiu veľkú časť backendu predstavujú triedy implementujúce MVC architektúru ako napríklad triedy *controller* alebo *Dao*. Tieto triedy často neobsahujú množstvo komplikovaných funkcií, avšak bez ich správneho fungovania sa dá aplikácia považovať za nefunkčnú.
- **Koncové body API** - Aj keď sa backend nestará o interakciu užívateľa s koncovou aplikáciou, je zodpovedný za chod a vystavenie koncových bodov API. Má teda zmysel testovať ich funkčnosť.
- **Databáza** - Väčšina užívateľských požiadavkov priamo či nepriamo vyžaduje interakciu s dátami v databáze. Môže ísť o jednoduché prihlásenie alebo rezerváciu jedného z dostupných áut. Správnosť databáze je preto podstatná pre fungovanie backendu.
- **Priebežné testovanie** - Uniqway je priebežne a dlhodobo vyvíjaná aplikácia. Preto musí existovať spôsob ako ju priebežne testovať, overovať funkčnosť starých funkcionalít a vyhodnocovať stav otestovanosti aplikácie.

Samozrejme, že tento zoznam nie je kompletný, často je potrebné otestovať aj iné miesta, za ktoré backend zodpovedá. Napr. integráciu aplikácií tretích strán alebo správnosť konfiguračných súborov. Zoznam ale predstavuje hlavné ciele, ktorých pokrytie by sme mohli od testovacieho procesu v Uniqway očakávať.

Na tomto mieste by bolo vhodné poznamenať, čomu sa táto práca pri analýze testovacieho procesu nebude venovať. Existuje veľké množstvo testovateľných oblastí, ale mnohé z nich nespadajú do kategórie bežných testov. Prípadne je ich podrobnejšia analýza natoľko komplikovaná, že si pomaly zaslúžia vlastnú odbornú prácu.

Napríklad tu nebudú preberané záťažové testy, ktorým už dokonca z bola venovaná časť bakalárskej práce zameriavajúcej sa na optimalizáciu výkonu backendu[18]. Penetračné testy, pre ktoré podľa neoficiálnych informácií bakalárska práca vzniká. Ani testy zaoberajúce sa splnením klientských požiadavkov. Zámer naopak bude hlavne na kontrolu funkčnosti, spoľahlivosti a udržateľnosti systému.

2.3 Metodológie testovania

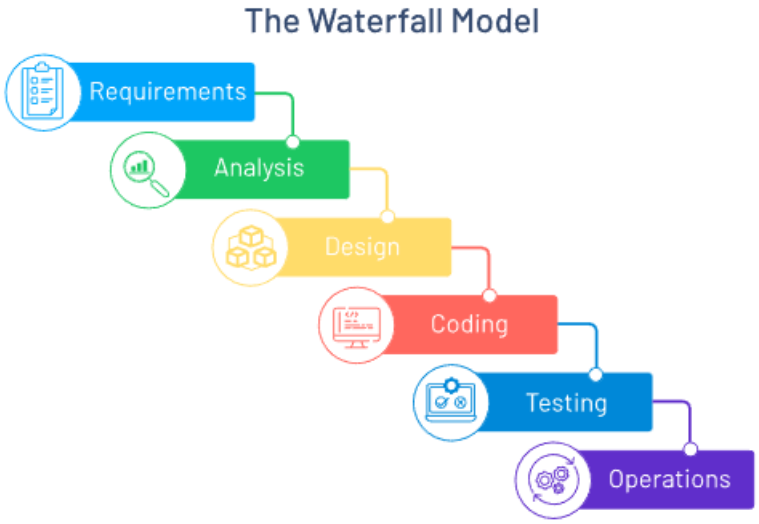
S postupom času stále prichádzajú nové metodológie pre vývoj softwaru, tie so sebou často prinášajú aj návod akým spôsobom testovať software. Medzi najpopulárnejšie sa radia waterfall a agile, ktoré prinášajú rozdielne pohľady na testovanie softwaru.

2.3.1 Waterfall

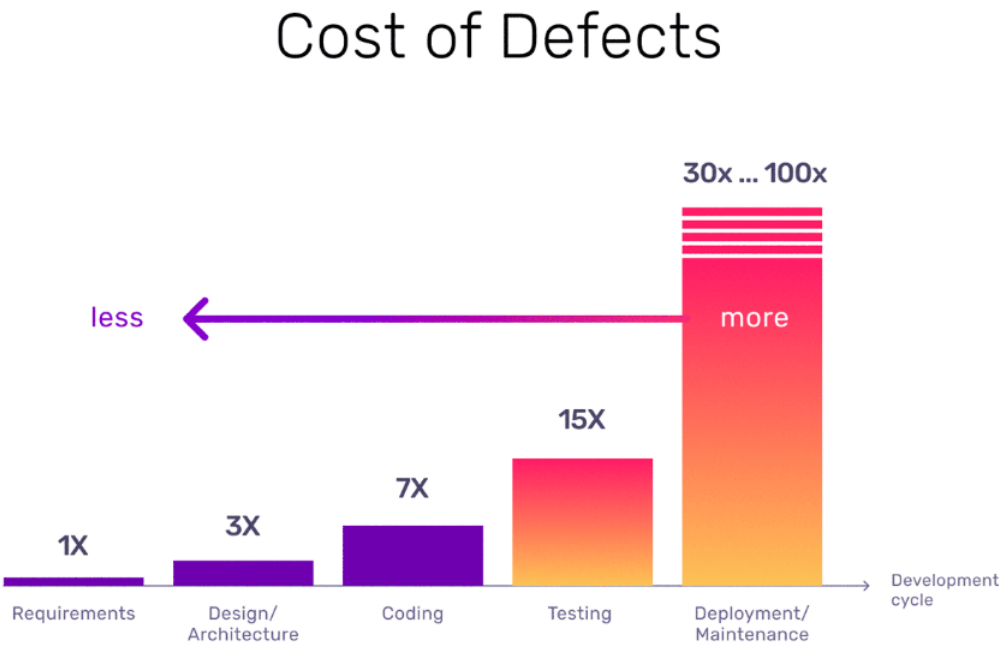
Waterfall, staršia z metodológií, rozdeľuje vývoj softwaru do oddelených fáz. Testovanie tu prebieha ako jedna z posledných častí pred vydaním aplikácie. Začína sa až potom čo je napísaný všetok aplikačný kód. [19]

To so sebou prináša výhodu písania testov pre celú aplikáciu naraz. Hlavnou nevýhodou je však znížená efektivita testovania, keďže testovacia fáza nasleduje až po tej vývojovej, testy nemôžu pomôcť s hľadaním chýb alebo kontrolou kvality v čase, kedy je samotný kód písaný. [19]

Ďalšou nevýhodou tohoto prístupu je fakt, že chyby nájdené v testovacej fáze sa môžu týkať kódu napísaného niekoľko mesiacov v minulosti, na ktorý už nadväzuje množstvo nových tried a funkcií. To pochopiteľne zvyšuje ako peňažnú tak časovú náročnosť opravy chýb. Z empirického pozorovania sa zdá, že ide dokonca o exponenciálny nárast.



Obr. 2.1 Metodológia waterfall [20]



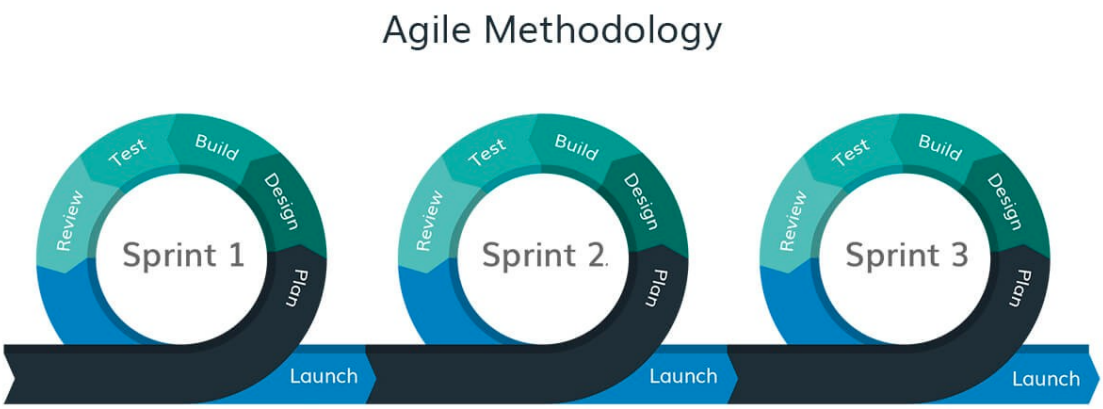
Obr. 2.2 Cena opravy chýb. [21]

2.3.2 Agile

Agile, novšia metodológia, sa snaží mnohé z problémov waterfallu eliminovať. Vývoj rozdeľuje do menších iterácií nazývaných **šprinty**, ktoré trvajú iba zopár týždňov.

Počas jedného šprintu sa pracuje na vývoji iba malej časti aplikácie. Vývojári prejdú od plánovania cez vývoj a testovanie až po zhodnotenie výsledkov šprintu. Takýchto šprintov je počas vývoja mnoho, čo pre testovanie znamená mnoho príležitostí priebežne testovať vyvíjanú aplikáciu.

Pre testovanie to teda znamená možnosť odhaliť chyby omnoho skôr a priebežne kontrolovať funkčnosť napísaného kódu [22].



Obr. 2.3 Metodológia agile [23]

2.3.3 Test-driven development

Test-driven development(TDD) ponúka ešte trochu odlišný spôsob testovania. S agile vývojom zdieľa myšlienku rozdelenia vývoju do malých iterácií, avšak v TDD sa vždy začína písaním testov.

Napísaním testu sa stanoví cieľ, ktorý by mala aplikácia splniť. Následne sa napíše minimum kódu na to, aby test prešiel. Opakovaním cyklu písania testov a následného písania kódu by mala vzniknúť aplikácia, ktorá je z definície vždy úplne otestovaná [24].

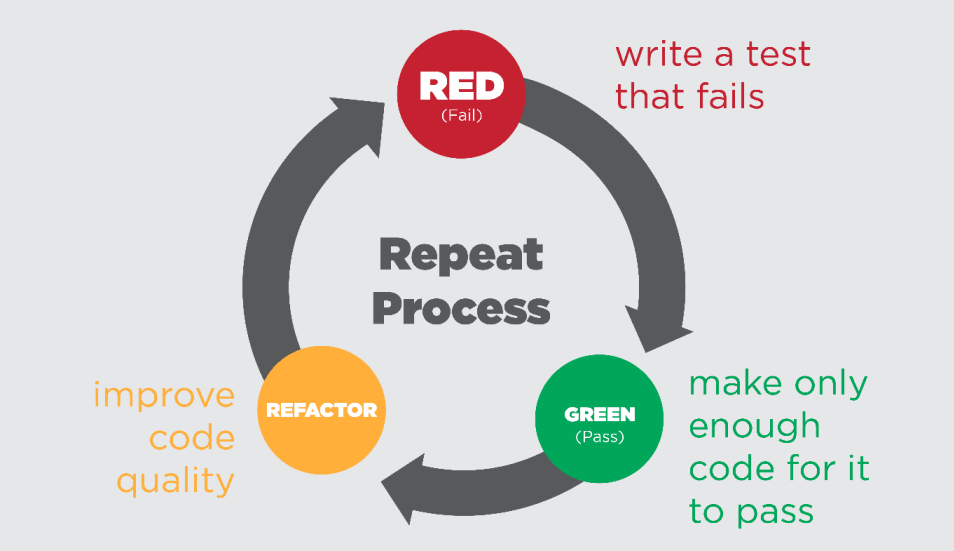
Hlavnou nevýhodou je, že napriek stúpajúcej popularite je TDD zriedka správne praktizovaný. V prieskume 300 softwarových developerov z roku 2020 odpovedalo 41% na otázku či ich tým adoptoval TDD kladne. Na otázku či píšu testy pred písaním kódu už odpovedalo kladne iba 8% [25]. Predmetom diskusie je aj to, či sa tento menej intuitívny prístup hodí aj do situácií s ťažko definovanými cieľmi.

2.4 Metodológie testovania v Uniqway

Vývoj samotného backendu prebieha v Uniqway spôsobom bližším k agilnej metodológii s častým vydávaním nových verzií aplikácie a častým prehodnocovaním krátkodobých cieľov. Nie je teda prekvapujúce, že testovací proces dodržiava podobný trend.

Konkrétny postup v praxi potom vyzerá tak, že novo vyvinutá funkcionalita, ktorá sa má stať súčasťou hlavného projektu, musí byť pred začlenením do produkčného prostredia riadne otestovaná, teda musia pre ňu existovať relevantné testy. Tým sa testovanie stáva jednou z povinných častí vývoja pri každej väčšej zmene zdrojového kódu.

Uniqway v súčasnej dobe stále nemá dostatok členov na to, aby sa niektorí z nich mohli naplno venovať testovaniu. Autorom potrebných testov je tak zpravidla ten istý vývojár, ktorý



Obr. 2.4 Test-driven development [26]

napísal kód testovanej funkcionality. V momente kedy má autor pocit, že otestoval funkcionalitu dostatočne ju odošle na kontrolu kvality.

Proces kontroly kvality prebieha formou tzv. *code review*, teda procesu kde správnu funkčnosť a kvalitu kódu musí odsúhlasiť aj iný vývojár. Zvyčajne ide o skúsenejšieho vývojára, ktorý sa nepodieľal na tvorbe danej funkcionality.

Použitie agilnej metodológie je pre rýchlo sa meniacu aplikáciu akou je Uniqway určite rozumnou voľbou. Systém akým je metodológia implementovaná sa zdá byť pre testovanie aplikácie dostatočné. Nové funkcionality sú riadne odladené a riadne otestované. V ojedinelých interných správach o výpadkoch systému sa vyskytujú odkazy na chyby v zdrojovom kóde iba zriedka.

Mierne nedostatky sa vyskytujú hlavne v tom, že sa nekladie veľký dôraz na testovanie jednotlivých funkcionalít medzi sebou. Pri menšom počte vývojárov je to však istá forma kompromisu. Súčasný systém testovania tiež nebol aplikovaný od začiatku vývoja, čo má u niektorých funkcionalít za následok nižší počet testov 2.8.

Jednou z možností ako ešte zlepšiť testy by bolo použitie TDD. Tam však nastáva problém adaptácie vývojárov študentov na menej tradičný spôsob vývoja.

2.5 Typy testov

Možností ako testovať software je mnoho a tak sa testy často delia na rôzne typy podľa rôznych kritérií.

2.5.1 Podľa spôsobu realizácie

Prvým kritériom je **spôsob realizácie** testov:

- **Manuálne testovanie** - „Ide o testovanie vykonávané samotným testerom. Môže napr. ísť o klikanie na tlačítka na webstránke alebo zadávanie hodnôt do formulárov na vyplnenie.

  Manuálne testovanie je dlhodobo na ústupe a postupne ho nahrádzajú automatické testy.“ [27] Stále uplatnenie si však drží na frontende, kde je ľudská spätná väzba dôležitým faktorom. Na backende je zavádzanie procedúr pre manuálne testovanie menej populárne, občasne

sa zavádza napr. pre kontrolu medzných hodnôt, vyskúšanie odozvy API alebo *exploratory testing* [28].

- **Automatizované testovanie** - Testovanie aplikácie pomocou písania skriptov v programovacích jazykoch, ktoré automaticky prechádzajú testovací scenár. Napr. spúšťajú funkcie so zadanými parametrami. Dnes ide o populárnejšiu variantu testovania, keďže automatické testy bežia oproti manuálnym niekoľkokrát rýchlejšie a je možné spúšťať opakovane.[27]

2.5.2 Podľa rozsahu

Najpoužívanejším spôsobom ako rozdeliť testy je potom ich rozdelenie podľa toho do ktorej, resp. ako veľkej časti aplikácie test zasahuje. Presná terminológia sa zdroj od zdroja zakaždým mierne líši[29], no základné rozdelenie vyzerá nasledovne:

1. **Unit testy** - Najjednoduchší test na overenie jednotky (unit) kódu. Môže ísť napr. o test jednej metódy [27].

2. **Integračné testy** - Testy na overenie toho, či viaceré komponenty programu spolu spolupracujú tak ako majú. Viac než na správnosť funkcií je tu kladený dôraz na bezproblémovú komunikáciu medzi komponentami. Oproti unit testom ide o výrazne komplexnejšie testy s vyšším stupňom abstrakcie [27].

3. **Funkčné/E2E testy** - Podobne ako integračné testy testujú viacero komponent naraz. Dôraz je tu však na to, či testovaná časť systému odpovedá užívateľským očakávaniam [27].

4. **Akceptačné testy** - Tieto testy sú vykonávané za behu celej aplikácie a realizuje ich zákazník. Ich cieľom je overiť, že produkt splňuje očakávané funkcie a kvalitu [27].

2.5.3 Testovacia pyramída

V súvislosti s týmto rozdelením sa často spomína model testovacej pyramídy. Tá hovorí o tom, že základ testovania by malo tvoriť veľké množstvo jednoduchých unit testov, ktoré majú za úlohu odchytiť väčšinu chýb pred implementáciou komplikovanejších typov testov.

Komplexné testy sú náročnejšie na tvorbu a zvyčajne trvá dlhšie aj samotný beh testu. So zvyšujúcou sa komplexitou by teda po správnosti mal počet testov ubúdať.

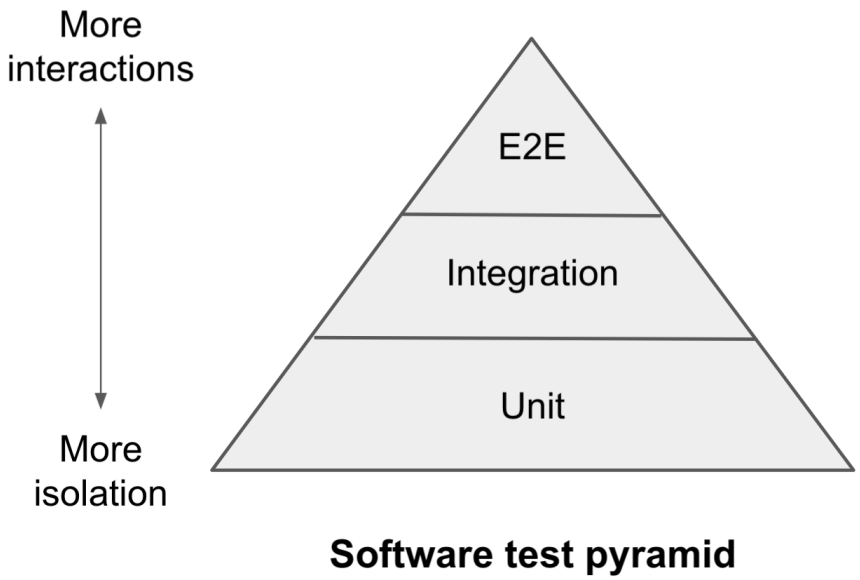
2.5.4 Testovací šesťuholník

Testovacia pyramída stále tvorí základ pre testovanie bežných softwarových aplikácií. Uniqway sa však v budúcnosti pravdepodobne bude uberať smerom k modulárnej architektúre mikroslužieb.

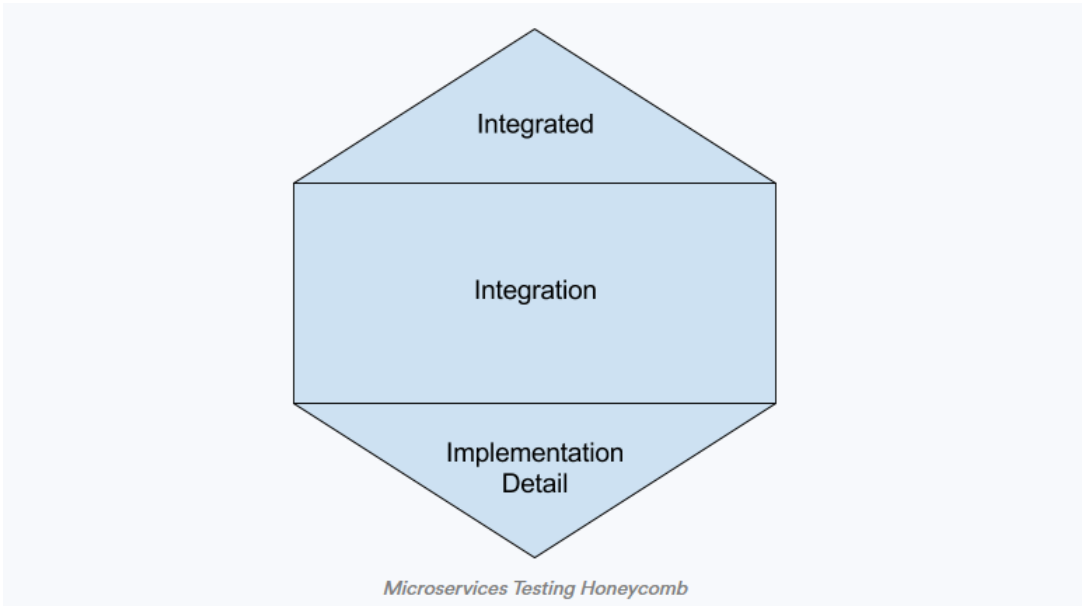
Tam však nastáva situácia, kde jedna aplikácia obsahuje veľké množstvo navzájom nezávislých komponent. Ako ukazuje príklad od firmy Spotify 2.6, k slovu sa tu viac dostáva integračné testovanie. Je to práve z dôvodu nutnosti testovania komunikácie medzi všetkými mikroslužbami.

2.5.5 Podľa funkcie

Do posledného rozdelenia patria niektoré špeciálne skupiny testov podľa ich využitia v testovacom procese.



Obr. 2.5 Testovacia pyramída [30]



Obr. 2.6 Testovací šesťuholník pre mikroslužby. [31]
Unit testy tu spadajú pod širšiu kategóriu *Implementation Detail*

- **Smoke testy** - „Využívajú sa k overeniu základnej funkčnosti aplikácie. Ide o malú sadu testov s rýchlou dobou behu, aby sa zistilo či sú hlavné časti softwaru funkčné.“ [27]

- **Regresné testy** - „Sú to testy, ktoré zisťujú či sa pri pridaní nových funkcií nenarušila funkčnosť starších častí. Ak si testovací proces môže dovoliť pravidelne spúšťať všetky svoje testy, potom všetky patria aj medzi regresné testy.“ [27]

2.6 Štruktúra testov v Uniqway

Väčšina testov zdrojového kódu sa nachádza v zložke test 1.3.2. Okrem toho ešte exituje niekoľko testov pre API, ktorým sa venuje sekcia Testovanie API 2.10.1.

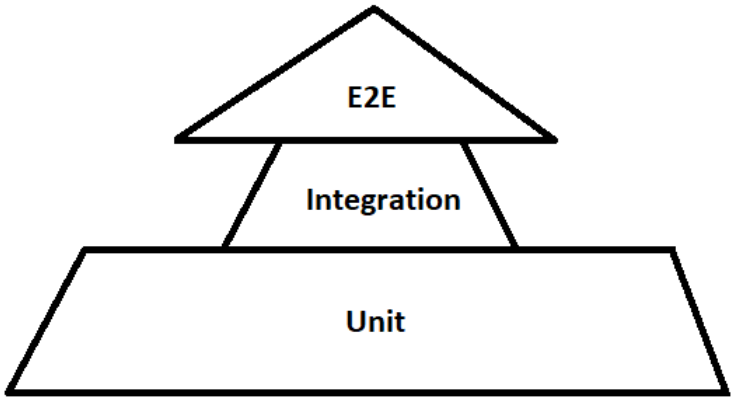
2.6.1 Typy napísaných testov

V zložke test sa v čase písania tejto práce nachádza niečo vyše 1800 testov, čo je na pomery projektu slušné číslo. Problém nastáva v momente, kedy sa bližšie zameriame na ich rozdelenie. Prevažnú časť testov tvoria unit testy, čo ak sa pozrieme na model testovacej pyramídy 2.5 nie je problém.

Vo vyšších poschodiach pyramídy je testov omnoho menej. Je tu niekoľko testov pre API, ktorým sa venuje sekcia *Testovanie API* 2.10.1.

Integračné testy, ktoré netestujú API sú však iba dva. Oba testy testujú integráciu externých balíčkov do projektu. Komplexnejšie testy, ktoré by napríklad kontrolovali komunikáciu naprieč vrstvami MVC architektúry tak pre samotný backend v podstate neexistujú, čo komplikuje aj ich analýzu. Dôvodom pre ich absenciu môže byť vyššie spomínaná absencia dedikovaného testera, vysoká previazanosť tried v aplikácií, ktorá sťažuje ich simuláciu alebo doterajšie nezavedenie lokálnej databázy pre testovanie.

E2E testovaniu sa venuje zložka quality-assurance1.2. Nachádzajú sa tu spomínané testy pre API 2.10.1 a množstvo pomocných funkcií pre manuálne testovanie pri testovaní lokálnej verzie aplikácie. Tieto testy sa však zameriavajú na užívateľský pohľad a tak nie sú plnohodnotnou náhradou integračných testov.



Obr. 2.7 Pyramída testov v Uniqway

2.7 Unit testy

Písanie unit testov je naopak zvládnuté veľmi dobre. Sú písané vo framewroku JUnit[32] navrhnutom pre testovanie v Java kódu. Testy musia odpovedať štandardom určeným v interných dokumentoch Uniqway. Každý test musí byť pomenovaný podľa triedy, ktorú testuje a umiestnený v správnom adresári. Názov metód musí navyše okrem testovanej funkcie obsahovať aj popis vstupných parametrov a očakávaného výsledku. Časťou internej dokumentácie je rovnako niekoľko doporučení pre testovanie rôznych typov funkcií. Pred začlenením do projektu sú potom testy kontrolované cez code review 2.4.

```
@Test
public void setReservationPricing_freePricing_shouldSetPrice() throws PaymentCardNotValidException {
```

Obr. 2.8 Príklad názvu unit testu v Uniqway pre triedu ReservationServiceTest

```
.
└── test/
    └── unit/
        ├── actions
        ├── annotations
        ├── car
        ├── mails
        ├── models/
        │   ├── main
        │   ├── user
        │   └── ...
        ├── services/
        │   ├── admin
        │   ├── client
        │   └── ...
        └── utils
```

Obr. 2.9 Skrátená adresárová štruktúra unit testov

Jediným problémom je to, že aj napriek ich počtu nepokrývajú všetky časti projektu. Táto situácia je pravdepodobne spôsobená dvomi príčinami.

Prvou je fakt, že terajší systém testovania v začiatkoch projektu neexistoval a tak mnoho tried žiadne testy nemá.

Druhou je veľmi veľký rozsah projektu, kvôli ktorému je časovo nepraktické testovať všetky jednoduché a priamočiare metódy. Bližšia analýza tohto problému je v sekcií *code coverage* 2.8.

2.8 Code coverage

Code coverage alebo pokrytie kódu testami je jedna z mála objektívnych metrík na meranie úspešnosti testovacieho procesu. V princípe odpovedá na otázku, koľko percent aplikačného kódu je pokrytého nejakým testom.

Toto číslo má mnoho využití a občas je ho možné vidieť na popredných miestach rôznych aplikácií[33]. Stále ale ide len o orientačnú metriku. V Uniqway má code coverage najväčšiu výpovednú hodnotu pre unit testy, keďže tie majú za cieľ otestovať funkčnosť každej funkcie v

systéme.

Na výpočet code coverage v Uniqway bola použitá open source knižnica JaCoCo vyvinutá pre Java aplikácie. Údaje o pokrytí získava analyzovaním Java bytecode po tom, čo prebehli všetky testy [34]. Z týchto informácií následne generuje výslednú správu so štatistikami pre daný projekt. Štatistiky nižšie boli získané cez integrácie JaCoCo s vývojovým prostredím IntelliJ IDEA[35] a s nástrojom pre zostavovanie aplikácií sbt cez plugin sbt-jacoco[36].

Jacoco Coverage Report

Element	Missed Lines	Total Lines	Cov.	Missed Branches	Cov.	Missed
models.main.reward		166	36%		100%	59
models.main.ride		59	47%		100%	14
services.client.versions		37	78%		100%	3
services.admin.tire		335	97%		100%	7
services.global.versions		54	94%		100%	1
services.admin.review		51	96%		100%	1

Obr. 2.10 Príklad JaCoCo code coverage reportu

```
72.
73.     public LocalDate getMaxExpirationDate() {
74.         return maxExpirationDate;
75.     }
76.
77.
78.     public void setMaxExpirationDate(LocalDate maxExpirationDate) {
79.         this.maxExpirationDate = maxExpirationDate;
80.     }
81.
82.
83.     public int getAmount() {
84.         return amount;
85.     }
86.
87.
88.     public void setAmount(int amount) {
89.         this.amount = amount;
90.     }
91.
```

Obr. 2.11 Príklad vyznačenia pokrytého kódu v JaCoCo reporte

2.8.1 Metriky pokrytia

Metrík na počítanie percent pokrytia code coverage je niekoľko. JaCoCo ponúka **Branch**, **Line**, **Method** a **Class coverage**. **Branch** sa zameriava na pokrytie všetkých vetiev v kóde (príkazy if a switch). **Line** na počet pokrytých riadkov. **Method** na počet metód a **Class** na počet tried[34].

Všetky metriky používajú na výpočet pokrytia podobný vzorec a výsledky reprezentujú percentuálne. Ten je názorne predstavený pre Line coverage 2.1.

$$\text{Line coverage} = \frac{\text{Počet riadkov pokrytých testami}}{\text{Celkový počet riadkov}} \times 100 \tag{2.1}$$

Najdôležitejšou tabuľkou bez zabiehania do prílišných implementačných detailov o zdrojovom kóde je tabuľka súhrnných štatistík pre jednotlivé priečinky implementujúce backendovú časť Uniqway.

Element	Class, %	Method, %	Line, %	Branch, %
parsers	50% (1/2)	100% (1/1)	100% (2/2)	100% (0/0)
traits	100% (3/3)	78% (11/14)	78% (11/14)	0% (0/4)
utils	56% (58/102)	41% (132/321)	50% (349/685)	61% (56/91)
actions	50% (6/12)	40% (22/54)	38% (57/147)	27% (10/36)
models	47% (301/637)	34% (2334/6723)	37% (4192/11300)	43% (189/435)
services	33% (156/460)	26% (1006/3837)	30% (3781/12397)	32% (641/1962)
mails	37% (6/16)	22% (14/63)	24% (39/161)	66% (4/6)
dao	5% (9/157)	2% (10/365)	1% (21/1152)	0% (0/57)
controllers	2% (11/503)	0% (7/2065)	0% (26/4925)	0% (0/670)
security	16% (1/6)	3% (1/31)	0% (1/110)	0% (0/28)
converters	0% (0/2)	0% (0/2)	0% (0/29)	0% (0/2)
modules	0% (0/7)	0% (0/6)	0% (0/20)	100% (0/0)
filters	0% (0/2)	0% (0/6)	0% (0/10)	100% (0/0)

Obr. 2.12 Súhrnné štatistiky pokrytia testami

Doporučenou hodnotou pre code coverage, najčastejšie meranou cez Line coverage býva približne 60-80% [37][38][39]. Menej percent zvyčajne poukazuje na viaceré netestované časti a pri snahe dosiahnuť 100% klesá návratnosť času investovaného do písania menej podstatných testov[39].

Doporučenú hodnotu dosahujú iba 2 priečinky z 13. Je však nutné dodať, že výsledky mierne skresľuje veľké množstvo krátkych neotestovaných funkcií, pre ktoré by testy vyzerali triviálne. Manuálne písanie týchto testov by v tomto prípade nemuselo byť najvhodnejším využitím času programátora. Aj napriek tomu tu však existuje niekoľko priečinkov s nízkymi jednotkami percent, ktoré by si zaslúžili pridanie aspoň regresných testov na spätnú kontrolu ich funkčnosti.

2.9 Priebežné testovanie

Testovací proces sa okrem písania testov musí starať aj o ich pravidelné spúšťanie. V Uniqway je spúšťanie testov povinnou časťou CI/CD pipeline v súbore .gitlab-ci.yml projektového Git repozitáru. Testy sa teda vždy spúšťajú počas procesu nasadzovania novej verzie aplikácie.

Tu sa po zostavení aplikácie v Docker kontajneri spustí sbt skript **test**. Ten spustí všetky testy v zložke test. Obdobný proces funguje aj pri spúšťaní API testov, avšak za pomoci iného skriptu. Podmienkou pre úspešné nasadenie novej verzie aplikácie je potom 100% priechodnosť všetkých spustených testov.

2.9.1 Lokálne testovanie

Lokálne testovanie benefituje zo spomínaného sbt skriptu *test*, ktorý je možné spustiť aj z terminálu. Testovanie konkrétnych tried je možné pomocou sbt príkazu *testOnly* umožňujúceho spustiť zadaný zoznam testov.

Jednoduchá povaha testov znamená, že beh viac než 1800 testov trvá na priemernom počítači približne iba jednu minútu. Ich výstup však môže byť kvôli vysokému počtu ťažko čitateľný. Do budúcnosti tak môže byť vhodné vytvoriť skripty testujúce iba určitú skupinu.

2.10 Ostatné testovateľné komponenty

Táto sekcia sa zaoberá časťami backendu, ktorých úlohou je spravovať fungovanie komponent operujúcich mimo samotnej serverovej aplikácie. Konkrétne je tu preskúmané testovanie API a databázy.

2.10.1 Testovanie API

Funkčnosť API je testovaná trochu separátne od ostatných testov. Testy sa nachádzajú v zložke *quality-assurance* a nie *test*. Na rozdiel od ostatných testov sú tiež písane v jazyku Python.

Ide o kolekciu približne 100 testov dopodrobna kontrolujúcich funkčnosť koncových bodov API tým, že na ne posielajú rôzne požiadavky a kontrolujú správnosť http odpovedí, textových odpovedí alebo správneho nastavenia hodnôt objektov. Príkladom je napríklad overenie admin práv alebo zobrazenie počtu litrov benzínu v nádrži auta po zaslaní údajov z modulu auta.

API v Uniqway je teda pomerne dobre otestovaná. Navyše tu okrem funkčných testov existujú aj záťažové testy. Ich bližšiemu testovaniu a testovaniu koncových bodov API sa venuje v časti svojej bakalárskej práce o optimalizácií výkonu backendu Hoang Nam Tran[18].

2.10.2 Databáza

U projektov s väčšou databázou postupne prichádza potreba kontrolovať jej správnosť. Na tomto projekte sa dá testovanie databázy rozdeliť na 2 kategórie. Prvou je testovanie SQL migračných skriptov používaných na tvorbu databázovej schémy. Migračné skripty sú predstavené v sekcií *Správa databáze cez migračné skripty* 1.3.7.2 a neskôr detailne v sekcií *Migračné skripty* 3.2.2. Druhou je testovanie databázovej schémy samotnej, teda overovanie správnosti mapovania tabuliek na objekty, testovanie *triggers*, *constraints* a ďalších relačných závislostí.

V Uniqway momentálne automatizované testovanie databáze neprebieha. Správnosť sa teda kontroluje manuálne. Tvorba viac ako 130 migračných skriptov tak prebehla viacmenej metódou pokus-omyl, kedy správnosť bolo potrebné overiť pri behu aplikácie. V terajšom stave migračné skripty navyše obsahujú zopár preklepov. Nie je ich teda možné použiť na zostavenie novej prázdnej databáze a ani na úplný rollback už zostavenej databáze do pôvodného stavu. Súčasným riešením je používať už zaužívanú databázu, písať migračné skripty tak aby na nej fungovali a snažiť sa o čo najmenej rollbackov.

Výpis kódu 2.1 Príklad migračného skriptu

```
# --- !Ups

CREATE TABLE not_registered_emails
(
  id    bigserial primary key,
  email varchar(64) not null unique
);

alter table car_model rename to car_models;

# --- !Downs

drop table if exists not_registered_emails;

alter table car_models rename to car_model;
```

Testovanie samotných dát na tom nie je omnoho lepšie. Problémy s migračnými skriptami napomáhajú skutočnosti, že sa v terajších testoch nikde nevyužíva dummy databáza a nie je teda

kde testovať napr. vloženie alebo upravovanie dát, prípadne správnosť nastavených obmedzení databáze.

Kapitola 3
Návrh

Táto kapitola sa venuje navrhovaným zmenám v testovacom procese backendu Uniqway. Na začiatku je predstavených niekoľko konkrétnych návrhov zmien na základe postrehov z analytickej časti. Ďalej sú tu rozobrané dve konkrétne zmeny, pre ktoré je predstavený detailnejší návrh ich prevedenia tak, aby boli kompatibilné so zvyškom systému. Ako názov sekcií napovedá, ide o zavedenie testovania tvorby databázového schématu a o automatické generovanie regresných testov. V oboch prípadoch sa jedná najmä o presné stanovenie si cieľov a rozbor použiteľných technológií.

3.1 Všeobecné možnosti vylepšení

V tejto sekcií je predstavených niekoľko oblastí na možné vylepšenia testovacieho procesu v Uniqway. Vylepšenia sa prevažne týkajú pridania nových funkcionalít alebo rozšírenia rozsahu testov. Je to hlavne z dôvodu, že súčasný testovací proces by sa dal považovať za dobre nastavený. Metodika testovania odpovedá spôsobu vývoja, počet novo vznikajúcich unit testov je dostatočný a aplikácia je priebežne testovaná pri každej aktualizácií produkčného kódu. Najväčším nedostatkom je práve neprítomnosť rozličných typov testov a pomerne veľké množstvo staršieho neotestovaného kódu.

- **Integračné testy** - Kontrole systému by výrazne prospel vznik viacerých komplexných testov, ktoré by boli schopné kontrolovať zložitejšie operácie, pre ktoré je nutná spolupráca viacerých funkcií, komponent alebo databáze. Príkladom takého testu je napríklad test uloženia DTO užívateľa a auta do databáze a následné vytvorenie rezervácie jazdy. V súčasnej dobe je týchto testov iba minimum. Možnosť testovať však sťažuje veľká previazanosť aplikácie a nevyužívanie simulácie reálnej databázy v testoch. Napriek veľkému potenciálu tak ide o ťažko dosiahnuteľný cieľ. Terajšiu situáciu ešte komplikuje fakt, že do najbližšieho produkčného spustenia aplikácie sa očakávajú výrazné, ešte neupresnené zmeny v oblasti jej modularizácie. Dopisovať testy pre dnešnú architektúru tak stráca zmysel. Naopak zaviesť testy v novom systéme hneď od začiatku by mohlo byť prínosné.

  Odhad dopadu na systém:

Schopnosť kontrolovania kódu	Schopnosť odhalenia chýb	Náročnosť implementácie	Náročnosť údržby
vysoká	vysoká	vysoká	stredná

- **Funkčné testy** - Funkčné testy testujú aplikáciu už z pohľadu užívateľa a nie sú teda doménou len backendu. Výhodou vytvorenia nových funkčných testov pre backend je, že za istých okolností môžu nahradiť funkciu chýbajúcich integračných testov bez nutnosti meniť architektúru. Mohli by, síce s väčšou mierou abstrakcie ako u integračných testov, testovať komunikáciu komponent, prípadne databázy, cez rôzne uživateľské scenáre. Najväčším

problémom je opäť súčasná situácia, kedy nie je jasná budúca podoba aplikácie, a pochopiteľne teda ani čo je v nej potrebné testovať.

Odhad dopadu na systém:

Schopnosť kontrolovania kódu	Schopnosť odhalenia chýb	Náročnosť implementácie	Náročnosť údržby
stredná	stredná	stredná	stredná

- **Databázová schéma** - Testovanie databáze pri tom ako komplikovaná je schéma v Uniqway umožňuje šetrenie množstva času a problémov. Viditeľné je to najmä v prípade migračných skriptov, kde sa chyba môže nenápadne šíriť a tvar vytvorenej databáze nemusí odpovedať našim predstavám. V horších prípadoch nemusia byť migračné skripty ani aplikovateľné. Spoznať že je tu niečo zle je bez nástroja na testovanie náročné. Rovnako tak je potrebné overiť správnosť operácií s dátami, k čomu je opäť potrebný nástroj, ktorý by simuloval databázu aj s aplikovanými migračnými skriptami.

  Odhad dopadu na systém:

Schopnosť kontrolovania kódu	Schopnosť odhalenia chýb	Náročnosť implementácie	Náročnosť údržby
stredná	stredná	stredná	stredná

- **Automaticky generované regresné testy** - Pomerne veľká časť zdrojového kódu nie je pokrytá žiadnymi testami. Ideálnym riešením by samozrejme bolo tieto testy vytvoriť. Z časových dôvodov je však zaujímavou možnosťou vygenerovať automatizované regresné testy pomocou jedného z programov na to určených. To poskytne aspoň základné overenie funkčnosti týchto funkcií.

  Odhad dopadu na systém:

Schopnosť kontrolovania kódu	Schopnosť odhalenia chýb	Náročnosť implementácie	Náročnosť údržby
stredná	nízka	nízka	nízka

- **Statické metriky** - Najpopulárejnejšej statickej metrike, code coverage, bola venovaná sekcia 2.8. Tam sa však jednalo iba o jednorázový zber dát. V terajšom stave sa statické metriky o zdrojovom kóde počas priebežného testovania nezbierajú. Pravidelné zisťovanie údajov o komplexnosti kódu, udržateľnosti, code coverage alebo napríklad iba jednoduchý prehľad o vývoji celkového počtu riadkov pomocou programov ako SonarQube alebo JaCoCo integrovaných do CI/CD pipeline by sprehľadnilo zisťovanie stavu v akom sa projekt nachádza.

  Odhad dopadu na systém:

Schopnosť kontrolovania kódu	Schopnosť odhalenia chýb	Náročnosť implementácie	Náročnosť údržby
-	-	nízka	nízka

- **sbt alias pre skupiny testov** - Spúšťanie testov cez nástroj sbt teraz umožňuje buď spustenie všetkých 1800 testov, alebo nutnosti názov testu špecifikovať. Nastáva tak príležitosť zadefinovať niekoľko nových príkazov na spustenie skupiny podobných testov.

  Odhad dopadu na systém:

Schopnosť kontrolovania kódu	Schopnosť odhalenia chýb	Náročnosť implementácie	Náročnosť údržby
-	-	nízka	nízka

Po diskusii s backend teamom v Uniqway a autorovom zhodnotení realizovateľnosti jednotlivých zmien bolo rozhodnuté implementovať testovanie tvorby databázovej schémy a vyskúšať aplikovateľnosť automaticky generovaných regresných testov.

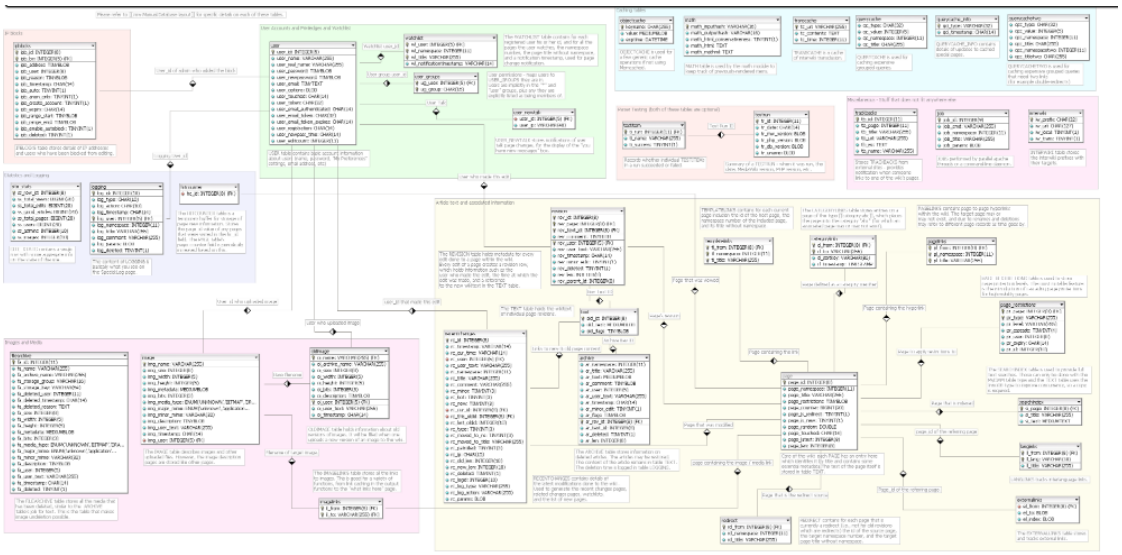
3.2 Návrh testovania tvorby databázovej schémy

Táto sekcia začína detailným popisom problému testovania migračných skriptov pre tvorbu databázovej schémy a testovania databázovej schémy samotnej s využitím SQL príkazov v testova-

com prostredí JUnit. Následne sa zaoberá stanovením konkrétnych cieľov a výberu technológií potrebných pre implementáciu vhodného riešenia.

3.2.1 Popis problému

Pojem *schéma databáze* hovorí o tom ako je databáza štrukturálne poskladaná. V relačnej databáze zachytáva formu jednotlivých tabuliek, vzťahy medzi tabuľkami, *constraints*, *triggers* a množstvo ďalších prvkov tvoriacich jej kompletný popis. V prípade Uniqway je celá databáza postavená pomocou migračných skriptov. S trochou nadsádzky by sa o týchto skriptoch tvoriacich schéma databáze dalo hovoriť ako o jej zdrojovom kóde. Vizuálnym príkladom toho ako komplikované môže schéma byť je starší príklad zo systému stojacim za internetovou encyklopédiou Wikipédia.



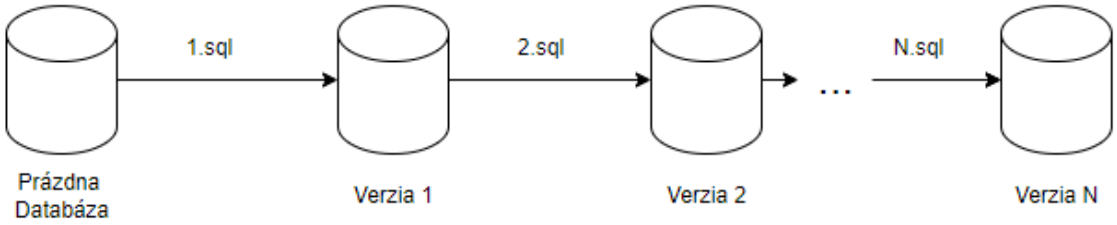
Obr. 3.1 Príklad databázovej schémy Wikipédie v roku 2007 [40]

3.2.2 Migračné skripty

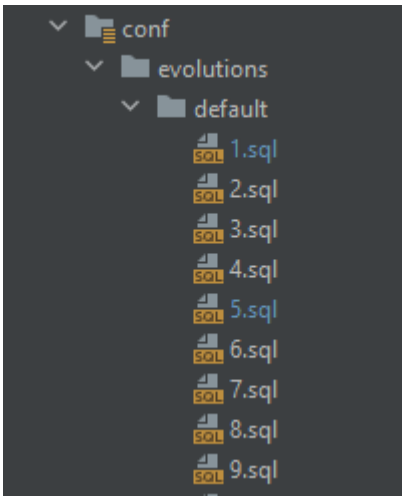
Jedným zo spôsobov ako si držať prehľad v tomto kóde je ho rozdeľovať na menšie skripty. V takom prípade je každý zo skriptov *migračný*. Teda v sebe obsahuje príkazy, ktoré upravujú formu databázy, resp. ju migrujú z jednej verzie do druhej. Postupným aplikovaním všetkých migračných skriptov potom vzniká celková schéma databáze.

Play framework nazýva tieto migračné skripty **evolutions** a každá evolution má za názov svoje poradové číslo. Začína sa súborom *1.sql*, za ktorým nasledujú *2.sql*, *3.sql* atď. Príklad takého skriptu je v sekcií 2.10.2. Play framework vyžaduje aby sa skripty pre danú databázu nachádzali v priečinku **conf/evolutions/„meno databázy“**. Základná databáza, používaná aj v Uniqway je pomenovaná **default**. Evolutions sa teda nachádzajú v priečinku **conf/evolutions/default**.

Bežné aplikovanie evolutions prebieha automaticky. Najprv je potrebné nakonfigurovať pripojenie databáze k aplikácií v súbore **application.conf**. Potom je do toho istého súboru potrebné pridať riadok **play.evolutions.autoApply=true**. Po spustení aplikácie s nakonfigurovaným pripojením k databázi sa evolutions aplikujú automaticky [13].



Obr. 3.2 Verzovanie databáze v Play



Obr. 3.3 Adresár evolutions.

Každá evolution sa skladá z časti **Up** a časti **Down**. Up časť má za úlohu aplikáciu žiadaných zmien a posun databáze do novej verzie. Down zase robí presný opak, anuluje všetky zmeny v Up časti a databázu vráti do stavu ako pred aplikovaním evolution.

Inconsistent state, alebo nekonzistentný stav je pojem, ktorým Play označuje chybný stav databáze, ku ktorému došlo chybou pri aplikovaní evolutions. V praxi Play framework aplikuje evolutions skripty tak, že ich rozdelí na jednotlivé SQL príkazy a tie po jednom odosiela databázovému serveru. Vo veľkej väčšine prípadov tak ide o stav, kedy nebolo možné SQL príkaz úspešne vykonať. V tomto prípade sa proces aplikovania evolutions zastaví a databázu je potrebné manuálne dostať do správneho stavu [13].

Chyby v tvorbe schémy databázy sa pochopiteľne prejavujú pri prechode z jednej verzie databáze na druhú. V prípade písania najnovšieho Up skriptu pre aktívnu databázu sa ešte dá očakávať, že skript bude po zopár úpravách fungovať, no jeho manuálne testovanie môže aj tak byť zbytočne komplikované.

Dôležitou poznámkou je aj to, že pri používanej databáze sa často prechádza z iba verzie N-1 na verziu N. V čase medzi aplikáciou starého a nového skriptu je pritom databáza aktívne používaná. Nie je teda zaručené, že skript bude fungovať aj pri zostavení novej databáze. Príkladom je zopár skriptov v Uniqway, kde sa pri aplikovaní Up skriptu počítalo s manuálne vopred uloženými dátami. Aplikovanie Down skriptov je zvyčajne odskúšané až v momente, kedy je ich potrebné aplikovať v produkcii. Chýba teda možnosť jednoducho skúsiť konkrétne evolutions aplikovať.

Výpis kódu 3.1 Evolution s nesprávnym Down skriptom.

```
# --- !Ups

ALTER TABLE users ADD COLUMN note varchar(600);


# --- !Downs

ALTER TABLE IF EXISTS users DROP COLUMN note varchar(600);

--Correct line should be:
--ALTER TABLE IF EXISTS users DROP COLUMN note;
```

S testovaním tvorby schémy prichádza aj možnosť testovania schémy samotnej. Dokonca pri rôznych verziách databáze. Testovanie napr. toho, či mapovanie dát odpovedá Java objektom v kóde alebo či obmedzenia a trigerrs fungujú správne zatiaľ nebolo systematicky aplikované. Riešenie by tak mohlo podporovať aj možnosť posielania SQL príkazov a tým automatizovane testovať nielen priechodnosť skriptov ale aj ich korektnosť.

3.2.3 Požiadavky na riešenie

Pred výberom konkrétneho spôsobu implementácie je potrebné si zadefinovať požiadavky na cieľové riešenie. Jednotlivé body sú vybrané tak, aby odrážali postrehy zo sekcie 3.2.2. Medzi požiadavky patria:

- **Riešenie musí fungovať offline a bez nutnosti zapnutia celej backendovej aplikácie.** Možnosť testovať migračné skripty bez spustenia samotnej aplikácie je jedným z hlavných bodov. Riešenie, ktoré by vyžadovalo bežiacu aplikáciu by stávajúci proces veľmi nezjednodušilo.
- **Riešenie musí jednoducho umožniť výmenu pripojenej databáze.** Pre kompatibilitu riešenia ako s budúcimi verziami, tak s rôznymi užívateľmi alebo Docker kontajnermi, je nevyhnutné umožniť jednoduchú výmenu databáze zadaním niekoľkých základných parametrov.
- **Riešenie nesmie byť viazané iba k jednému typu databáze.** Riešenie musí počítať s možnou budúcou zmenou DBMS a prechodom na iný typ databáze.
- **Riešenie musí byť schopné spustiť všetky uložené migračné skripty.** V kontexte frameworku Play ide o všetky skripty uložené v adresári **conf/evolutions**
- **Riešenie musí byť schopné spustiť iba časť migračných skriptov a dostať tak databázu do požadovanej verzie.** Play framework ponúka iba automatickú aplikáciu všetkých evolúcií. Požadované riešenie by toto malo rozšíriť o možnosť aplikovania iba konkrétneho počtu evolúcií zadaného testerom.
- **Riešenie musí byť schopné spustiť ktorýkoľvek samostatný skript.** V množstve testov môže byť prínosné aplikovať iba jednu konkrétnu evolúciu. Riešenie by malo túto možnosť poskytnúť.
- **Riešenie musí byť schopné v plnej miere aplikovať Up aj Down časti skriptov.** Opäť, jeden z hlavných bodov je plná podpora Up aj Down častí evolučných skriptov.
- **Riešenie musí implementovať základné testy pre otestovanie správnosti migračných skriptov.** Ide najmä o testy toho, či sa databáza aplikovaním rôznych kombinácií evolúcií nedostane do tzv. *Inconsistent state*.

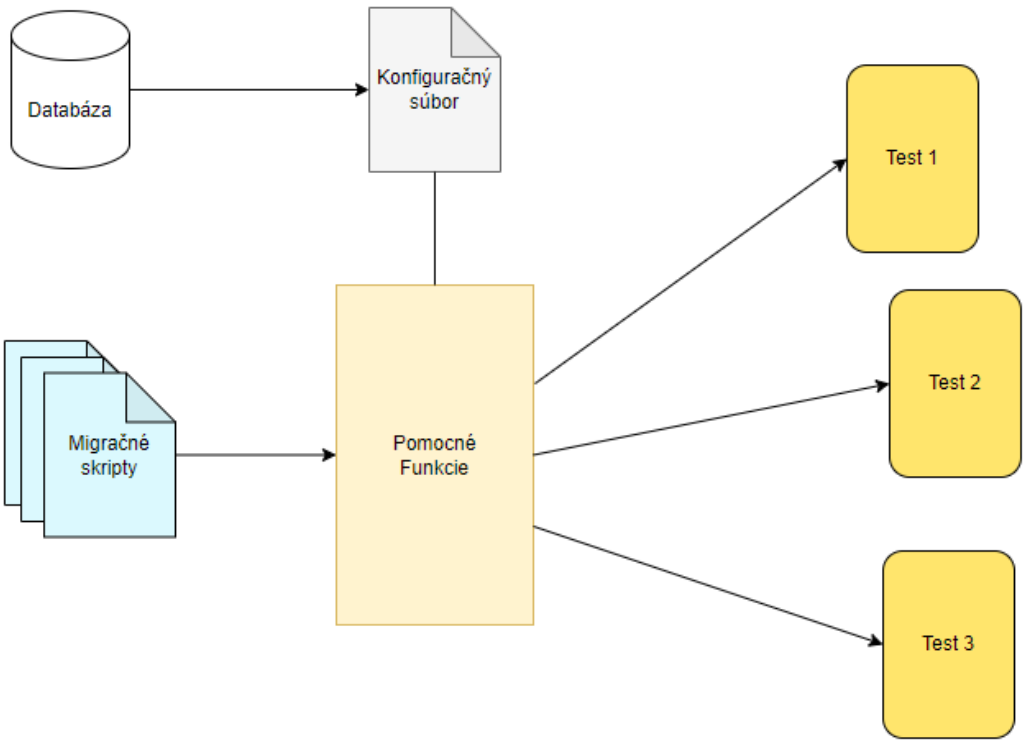
- **Riešenie by malo byť schopné podporovať odosielanie SQL príkazov do testovanej databáze počas behu testov.** Veľkou výhodou bude, ak riešenie bude jednoducho umožňovať zasielanie SQL príkazov na otestovanie pripojenej databáze.

  Súčasný návrh riešenia počíta s využitím programovacieho jazyku Java a frameworku Play.

3.2.4 Návrh usporiadania

Konečným cieľom je vznik testov pre migračné skripty a databázu. Riešením by tak mala byť trieda obsahujúca pomocné funkcie umožňujúce písanie týchto testov. Usporiadanie by mohlo vyzerať nasledovne 3.4.

Konfiguračný súbor - Súbor s obsahujúci informácie o databáze. Konkrétne *driver*, *url*, *username* a *password*.
Pomocné funkcie - Trieda implementujúca funkcie pre splnenie požiadavkov pre aplikovanie migračných skriptov z 3.2.3.
Test - Súbor implementujúci konkrétny test, ktorý dedí z triedy pomocných funkcií.



Obr. 3.4 Návrh usporiadania

3.2.5 Dostupné technológie

Technológie schopné testovať migračné skripty by sa dali rozdeliť na dve kategórie. V prvej kategórií sú funkcie poskytované samotným Play frameworkom. V druhej sú verejne dostupné open source nástroje pre správu databáz.

3.2.5.1 Play Evolutions

Play framework v sebe obsahuje podporu pre manuálne aplikovanie evolutions. Ide o zopár funkcií, ktoré sa spolu s iným obsahom zmestia do jednej stránky Play dokumentácie[41]. Tie sa z počiatku ukazovali ako nedostatočné pre implementáciu požadovaného riešenia. Po zdĺhavom skúšaní a vhodnému prispôsobeniu niektorých vlastností v procese načítania evolutions sa však zdajú byť postačujúce. Podstatnú časť tu zohráva funkcia **forDefault** [42], ktorá je schopná pri dodaní SQL skriptov vygenerovať vlastnú evolution.

Výpis kódu 3.2 Príklad funkcie forDeafult

```
Evolutions.forDefault(
    new Evolution(
        1,
        "create table car_name (id int not null,
                modelname varchar(50));",
        "drop table car_name;"));
```

Veľkou výhodou tohoto prístupu je jednoduchá adaptácia riešenia do súčasného systému bez nutnosti pridávať nové závislosti. To rovnako uľahčí prácu vývojárom, ktorí už systém Uniqway poznajú.

3.2.5.2 Play Evolutions databáza

Rozdiel v technológiách pri Play frameworku nastáva až pri zvolení typu databáze. Riešenie by síce nemalo požadovať konkrétny typ databáze, no na jeho implementáciu by bolo vhodné zvoliť predpokladaný spôsob použitia. Play umožňuje buď využitie in-memory databáze alebo pripojenie k databázovému serveru.

In-memory databáza je narozdiel od bežných databáz uložená v pamäti samotného počítača, zvyčajne v RAM. To umožňuje omnoho rýchlejšie zostavenie a násobne rýchlejšie odpovede vďaka vyššej rýchlosti RAM oproti tradičným diskom. V praxi má niekoľko nevýhod spojených s nižšou spoľahlivosťou a vyššími nákladmi na prevádzku. Pre testovanie však ide o ideálnu voľbu, keďže na beh testov sa nie je potrebné pripájať k databázovému serveru a teoreticky by tak nebolo nutné nastavovať akékoľvek pripojenie. Testy na in-memory databáze by navyše bežali rýchlejšie.

Play poskytuje in-memory databázu založenú na technológií H2[43]. Na jej použitie stačí do zastavovacieho sbt súboru pridať jeden riadok s H2 závislosťou **libraryDependencies += "com.h2database"% "h2"% "1.4.192"**[44].

Neriešiteľný problém však nastáva pri práci s Uniqway databázou písanou v PostgreSQL. H2 síce poskytuje mód emulujúci syntax PostgreSQL pomocou príkazu **MODE=PostgreSQL**[44], avšak nie v plnej podobe.

Pri skúsení spustenia evolutions došlo k niekoľkým chýbam už pri prvom skripte. Po upravení jedného skriptu sa spravidla vyskytla nová chyba hneď v nasledujúcom. Na využitie in-memory databáze by tak museli byť prepísané napr. viaceré dátové typy, ktoré PostgreSQL umožňuje používať do ich jednoduchších ekvivalentov. To by následne znamenalo aj nutnú úpravu zdrojového kódu.

Uniqway navyše používa PostgreSQL rozšírenie PostGIS s dátovými typmi určenými pre záznam polohy. Ekvivalent tohoto rozšírenia síce existuje aj pre H2 databázu, avšak podpora rôznych rozšírení nie je obvyklá. Pri vývoji aplikácie by sa tak zbytočne muselo počítať s podporou SQL kódu dvoma rôznymi technológiami zároveň.

Druhou variantou je klasické pripojenie k **PostgreSQL databázovému serveru**. Výhodou je práve vytvorenie rovnakého prostredia aké je využívané v produkcii. Evolutions v tomto prípade fungujú bezproblémovo. Výhodou je, že PostgreSQL patrí k rýchlejším databázovým systémom a aplikácia všetkých evolutions trvá iba nízke jednotky sekúnd. Výkonnostne tak aj

táto varianta odpovedá požiadavkom riešenia a je vhodná na terajšie využitie. Nevýhodou je zatiaľ nutnosť existencie databázového pripojenia na spustenie testov.

3.2.5.3 Flyway

Flyway je open-source nástroj pre spravovanie migrácií databáz. Podporuje veľké množstvo DBMS vrátane PostgreSQL, MySQL, H2, SQLite alebo Oracle, SQL Server [45].

Spôsobom fungovania sa veľmi nelíši od evolutions v Play frameworku. Rovnako automaticky aplikuje sadu súborov uložených v priečinku na pripojenú databázu. Rozdiely sú v spôsobe ukladania súborov. Flyway je v tomto ohľade voľnejší. Umožňuje načítanie migračných skriptov z akéhokoľvek adresára, stačí aby bol definovaný v parametri **locations**.

Ďalší rozdiel je v rozdelení a pomenovávaní súborov. Každý súbor sa musí začínať prefixom určujúcim jeho funkciu. V základnom nastavení je to **V** pre migrácie aplikujúce nové zmeny, **U** pre migrácie vracajúce databázu do predošlého stavu a oproti Play evolutions ponúka aj možnosť **R** pre migrácie, ktoré je možné spúšťať opakovane. Za prefixami *V* a *U* vždy nasleduje číslo poradia verzie migračného skriptu, separátor v tvare __ a vlastný názov. Príkladom validného názvu migračného skriptu je napríklad **V3__Modify_fuel_tank.sql**[46].

Obr. 3.5 Príklad umiestnenia migračných skriptov vo Flyway [46]

Oproti evolutions tak rozdeľuje Up a Down časti skriptu do dvoch odlišných súborov a umožňuje každý skript výstižne pomenovať. Pre využitie Flyway by tak bolo potrebné previesť všetky existujúce skripty do odpovedajúceho stavu. Vzhľadom na podobný charakter oboch nástrojov by však nebolo potrebné meniť SQL príkazy a šlo by iba o jednoduché prekopírovanie a rozdelenie Up a Down častí do vlastných súborov.

Veľkou výhodou Flyway je zabudovaná podpora funkcií vhodných pre testovanie. Menovite ide o funkcie **Cherry Pick** [47] a **Target** [48].

Cherry Pick umožňuje aplikovať konkrétny zoznam skriptov v poradí v akom sú zadané. Napríklad príkaz *flyway -cherryPick=3,2,1* by aplikoval skript *3* potom skript *2* a nakoniec skript *1*.

Target umožňuje zadať číslo konkrétneho migračného skriptu a postupne spustiť všetky skripty od toho s najnižším číslom až po zadaný skript, čím dostane databázu do verzie odpovedajúcej aplikácií migračného skriptu so zadaným číslom. Napríklad príkaz *flyway -target=10* by dostal databázu do verzie odpovedajúcej aplikácií všetkých skriptov až po ten s číslom *10*.

Nevýhodou použitia Flyway je pochopiteľne nutnosť integrácie novej aplikácie do projektu a presun všetkých migračných skriptov. Väčším problémom je však neúplná integrácia s frame-

workom Play. Podpora existuje iba v podobe neoficiálneho pluginu **flyway-play** [49], ktorého autorom je Toshiyuki Takahashi. Ten síce podporuje väčšinu potrebných funkcionalít ale chýba mu podpora Down skriptov.

3.2.5.4 Liquibase

Liquibase, podobne ako Flyway je open-source nástroj pre správu migrácií databáz a rovnako podporuje veľkú väčšinu populárnych DBMS [50]. Jeho prvá verzia vyšla v roku 2006 a odvtedy sa presadil ako spoľahlivý a časom overený systém s veľkou mierou prispôsobivosti pre rozmanité typy projektov.

Správa migračných skriptov prebieha na rozdiel od Flyway a Play mierne odlišne. Miesto ich načítania z adresára sa migračné skripty spravujú cez súbor **databaseChangeLog** . Tento súbor obsahuje informácie o zmenách v blokoch zvaných **ChangeSets**. Každý blok obsahuje id, meno autora a buď cestu k súboru s migračným skriptom alebo samotný migračný skript. Liquibase podporuje okrem skriptov v SQL aj formáty XML, YAML alebo JSON. [51]

Výpis kódu 3.3 Príklad XML súboru databaseChangeLog

```
<?xml version="1.0" encoding="UTF-8"?>
<databaseChangeLog
    <preConditions>
        <runningAs  username="uniqway"/>
    </preConditions>

    <changeSet  id="1"  author="exampledev">
        <createTable  tableName="car_name">
            <column  name="id"  type="int"  autoIncrement="true">
                <constraints  primaryKey="true"  nullable="false"/>
            </column>
            <column  name="modelname"  type="varchar(50)"/>
        </createTable>
    </changeSet>
</databaseChangeLog>
```

Tak ako Flyway, aj Liquibase ponúka niekoľko zabudovaných funkcií vhodných pre testovanie. Ekvivalentom funkcie Target je tu funkcia *update-count* [52], ktorá aplikuje zadaný počet blokov ChangeSets. Príklad použitia update-count: *liquibase update-count –count=5 –changelog-file=some-changelog.xml*.

Podobne funguje ešte funkcia *update-to-tag* [53], ktorá aplikuje bloky ChangeSets až do momentu kedy nenarazí na blok s voliteľným parametrom **tag** s odpovedajúcim názvom.

Príklad použitia update-to-tag: *liquibase update-to-tag –tag=addFuelTank –changelog-file=some-changelog.xml*.

Ekvivalentom Cherry Pick je tu funkcia *update-one-changeset* [54], ktorá po zadaní id, mena autora a názvu súboru aplikuje požadovaný ChangeSet. Príklad použitia update-one-changeset: *liquibase update-one-changeset –changelog-file=some-changelog.xml –changeset-id=2 –changeset-author=somedeveloper –changeset-path=some-changelog.xml –force*.

Pre integráciu s frameworkom Play existuje modul **Play Liquibase Module** [55]. Ten ešte stále podporuje stávajúcu verziu Play frameworku, avšak už nebol viac než 3 roky aktualizovaný.

3.2.5.5 Zhrnutie

Flyway a Liquibase ponúkajú väčšie množstvo zabudovaných funkcionalít pre zjednodušenie procesu testovania. Avšak ich otázna podpora Play frameworku a mierne rozdiely v spôsobe ukladania skriptov, ktoré by znamenali buď nutnosť prechodu celého systému evolutions do iného nástroja alebo potrebu súbežne udržiavať dva rozdielne systémy naraz sú značnými nevýhodami.

Naproti tomu priamočiare riešenie s využitím funkcií poskytnutých Play frameworkom nevyžaduje žiadne zmeny pôvodného systému a implementácia nových funkcionalít sa zdá byť uskutočniteľná. Z týchto dôvodov sa Play framework ukazuje ako najlepšia voľba.

3.3 Návrh automatizovaného generovania regresných testov

Táto sekcia sa zaoberá návrhom implementácie automatizovanej tvorby regresných testov. Začína rýchlym úvodom do oblasti programov pre generovanie testov a stanovením základných cieľov implementácie. Potom nasleduje porovnanie dostupných programov vzhľadom k ich využiteľnosti v Uniqway a popis zasadenia vybraného programu do súčasnej aplikácie.

3.3.1 Popis problému

Z analýzy v sekcií *Code coverage* 2.8 vyplýva, že pre veľkú časť zdrojového kódu nie je napísaný žiadny test. Či už bol dôvodom obmedzený čas programátora alebo iné nastavenie bývalého testovacieho procesu, vznik testov pre tieto triedy a metódy by bol bezpochyby prospešný. Spätné dopisovanie testov je však jedno z najmenej efektívnych využití softwarového vývojára, najmä v menšom tíme akým je ten v Uniqway.

Riešenie tohto problému prisľubujú programy na automatické generovanie testov. Tie pomocou starostlivej analýzy zdrojového kódu a využitia selekcie rôznych komplexných algoritmov dokážu automaticky vygenerovať spustiteľné testy.

Konkrétne použité techniky pre tvorbu testov sa líšia a sú stálym predmetom akademického výskumu. Populárnymi technikami sú napr. *fuzzing*, *mutation testing*, *model-based testing*, *randomizované vyhľadávacie alogritmy*, *genetické algoritmy* a v poslednej dobe sa stáva čoraz viac populárnym využitie umelej inteligencie.

Líšia sa aj z pohľadu očakávaného výsledku. Mnohé programy si kladú za cieľ nájsť čo najviac chýb, zhodiť program alebo otestovať medzné hodnoty. Pre túto sekciu nás budú zaujímať programy schopné pre zadané metódy vygenerovať testy, ktoré spätne testujú ich funkčnosť a po ich spustení na terajšom kóde vždy prechádzajú.

Keďže ide o automaticky pracujúce programy, ich zavedenie do aplikácie nebýva príliš komplikované. Často ide o napísanie zopár skriptov na nakonfigurovanie komunikácie súboru s programom a testovanej aplikácie, prípadne o nastavenie iného rozhrania, ktoré daný program ponúka.

Ťažší je v tomto prípade vhodný výber programu. Nové programy často nadväzujú na výskum novej techniky na akademickej pôde. Nebýva tak neobvyklé, že po pár rokoch fungovania prestanú byť podporované keď pôvodní autori stratia záujem o ďalší výskum alebo sa nadväzujúci produkt komerčne neudrží. Napríklad ako v prípadoch programov JCrasher [56] alebo AgitarOne [57], ktoré generovali unit testy pre kód v Jave. Výber technológií sa tak sústreďuje iba na aktívne podporované programy.

Hlavným cieľom je tak zvýšiť pokrytie kódu automaticky generovanými testami a s postupom času overiť ich efektivitu. Generovanie regresných testov by v prípade, že sa ukáže ako vhodný doplnok mohlo byť používané aj pri písaní testov pre budúci zdrojový kód.

3.3.2 Požiadavky na riešenie

- **Riešenie musí poskytovať jednoduchý proces tvorby regresných testov.** V závislosti od vybranej technológie by malo ísť iba o zopár kliknutí alebo zadanie jednoduchého príkazu do terminálu.
- **Všetky výsledné regresné testy musia byť spustiteľné a úspešne prechádzať.** Podmienky sa pochopiteľne týkajú iba verzie kódu, pre ktorý boli vytvorené.

- **Riešenie musí byť schopné vytvárať testy opakovane.** Musí byť zabezpečená možnosť vytvoriť nové testy po každej zmene zdrojového kódu.
- **Riešenie musí byť schopné vygenerovať testy pre všetky metódy v aplikácií.**
- **Riešenie musí byť schopné vygenerovať testy pre konkrétne špecifikované metódy.**

3.3.3 Dostupné technológie

V tejto sekcií sa nachádza porovnanie programov pre automatické generovanie regresných testov pre Javu a zhodnotenie ich využiteľnosti v Uniqway. Analyzovaní budú stáli účastníci súťaží SBST Tool Competition [58] a Java Unit Testing Tool Competition [59] porovnávajúcej nástroje na generovanie Java unit testov. Konkrétne Randoop a EvoSuite. Ako posledné je ešte porovnanie nového komerčného nástroja Diffblue Cover.

Ukážky vygenerovaných testov sa vzťahujú k metóde v **crownsIntoPennies** 3.3.3.

Výpis kódu 3.4 Metóda crownsIntoPennies

```
public static Long crownsIntoPennies(Double crowns) {
    if (crowns == null) return null;
    return crownsIntoPennies(crowns.doubleValue());
}

 /**
 * Turns crowns into pennies.
 * E.g. 200 CZK = 20000 pennies
 *
 * @param crowns crowns
 * @return pennies
 */
public static long crownsIntoPennies(double crowns) {
    return (long) Precision.round(crowns * 100, 0);
}
```

3.3.4 Randoop

Randoop je open-source program pre generovanie unit testov. Okrem Javy existuje ešte aj verzia Randoopu s podporou pre platformu .NET. Testy generované pre Javu sú vytvorené vo frameworku JUnit.

„Pre tvorbu unit testov Randoop používa algoritmus generovania náhodných testov so spätnou väzbou. Táto technika pseudonáhodne, avšak sofistikovane generuje sekvencie volania metód resp. konštruktorov. Tieto sekvencie Randoop následne spustí a na základe odpovedí programu vygeneruje odpovedajúce testy. Randoop generuje regresné testy a testy odhaľujúce chyby v programe.“ [60]

Užívateľské rozhranie je poskytované cez príkazový riadok. Pre tvorbu testov existuje príkaz **gentests**. Na spustenie je mu potrebné dodať cestu k .jar súboru Randoopu, .jar súborom so všetkými závislosťami a .class súborom, ktoré majú byť testované.

Randoop sa vyznačuje tým, že oproti konkurencií generuje násobne väčší počet testov. Často aj stovky pre jedinú triedu. Testy sú si však veľmi podobné a tak ich počet zriedka zvyšuje efektivitu testovania.

Pri skúšaní generovania testov v Uniqway bol Randoop schopný po dlhšom nastavovaní konfigurácie schopný vygenerovať testy pre celý projekt.

Výpis kódu 3.5 Príklad vygenerovaného testu pre metódu crownsIntoPennies

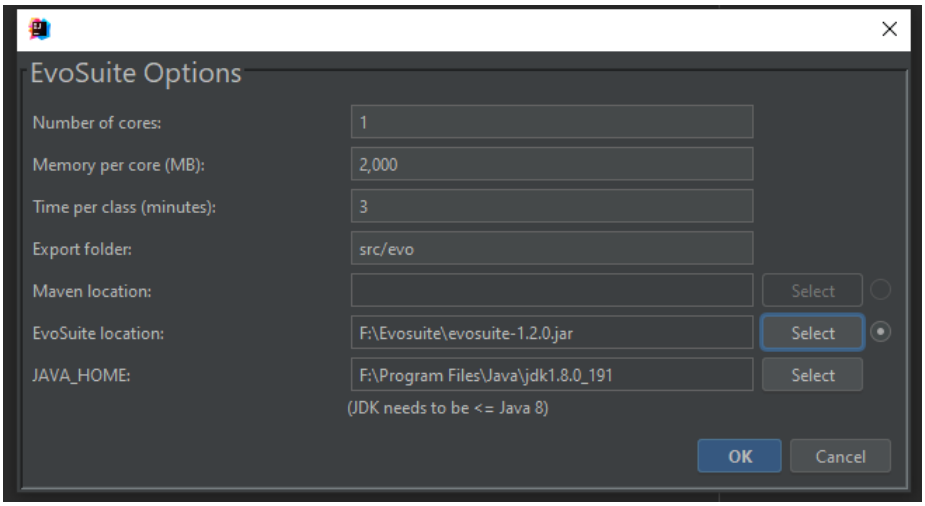
```
@Test
 public void test09634() throws Throwable {
     if (debug)
         System.out.format("%n%s%n", "RegressionTest19.test09634");
     long long1 = utils.helpers.CurrencyFormatter.
         crownsIntoPennies((double) 406L);
     org.junit.Assert.assertTrue("'" + long1 + "' !=
         '" + 40600L + "'", long1 == 40600L);
 }
```

3.3.5 EvoSuite

EvoSuite je rovnako ako Randoop open-source nástroj na generovanie unit testov a taktiež generuje testy vo frameworku JUnit [61].

„Na vytvorenie príslušného testu používa genetický algoritmus založený na evolučnom prístupe.“ [62]

Ponúka užívateľské rozhranie vo variantách príkazového riadku alebo pluginov do IDE. V príkazovom riadku sa podobne ako pri Randoope generujú funkciou **generateTests** so zadanou cestou k .jar súborom EvoSuite, závislostí a cestou k triedam, pre ktoré majú byť vytvorené testy. V prípade použitia pluginu je proces funkčne rovnaký ale používateľ môže jednotlivé cesty nastavovať v jednoduchom grafickom rozhraní.



Obr. 3.6 EvoSuite plugin pre Intellij

Na rozdiel od Randoopu, EvoSuite sadu vytvorených testov pred ukončením minimalizuje a podobné testy zlúči do jedného. Výstup tak generuje výrazne menej testov a je prehľadnejší.

Zásadnou nevýhodou sa však ukázala byť oficiálna podpora Javy iba po verziu Java 9, pričom Uniqway používa Javu 11. To sa prejavilo tým, že EvoSuite nedokázal po množstve neúspešných pokusov pre aplikáciu vygenerovať žiadne testy ani z príkazového riadku, ani z grafického pluginu.

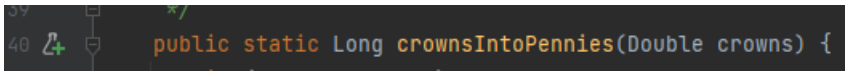
3.3.6 Diffblue Cover

Diffblue Cover je komerčný nástroj založený na práci výskumnej skupiny zaoberajúcej sa umelou inteligenciou z Oxfordskej univerzity. Prvá verzia vyšla v roku 2020, teda stále ide o pomerne nový nástroj. Rovnako generuje unit testy pre Javu vo frameworku JUnit [63].

Testy sú generované s využitím umelej inteligencie. Keďže ale ide o komerčnú firmu, presný algoritmus na ich tvorbu nie je známy [63]. Výhodou vytvorených testov je však to, že sú napísané tak aby boli pre vývojárov ľahšie porozumiteľné [64].

Diffblue Cover ponúka dve možnosti generovania unit testov. Prvou je plugin do Intellij Idea, ktorý pridáva ku každej metóde ikonku. Po kliknutí na ikonku sa pre danú metódu automaticky vygenerujú testy.

Pre vygenerovanie testov pre celý projekt naraz ponúka Diffblue Cover rozhranie cez príkazový riadok. Podobne ako pri vyššie spomínaných programoch sa v príkazovom riadku funkciou **create** so zadanou cestou k .jar súborom závislostí a cestou k triedam, pre ktoré majú byť vytvorené testy.



Obr. 3.7 Ikonka na generovanie testov v Diffblue Cover

Výpis kódu 3.6 Príklad vygenerovaného testu pre metódu crownsIntoPennies

```
/**
 * Method under test:
 * {@link CurrencyFormatter#crownsIntoPennies(double)}
 */
@Test
void testCrownsIntoPennies() {
    assertEquals(1000L, CurrencyFormatter.crownsIntoPennies(10.0d));
    assertEquals(0L, CurrencyFormatter.crownsIntoPennies
        (Double.NaN));

    assertEquals(1000L, CurrencyFormatter.crownsIntoPennies
        ((Double) 10.0d).longValue());

    assertNull(CurrencyFormatter.crownsIntoPennies((Double) null));
    assertEquals(0L, CurrencyFormatter.crownsIntoPennies
        ((Double) Double.NaN).longValue());
}
```

Bohužiaľ, voľná verzia je dostupná len pre individuálnych developerov a povoľuje iba 100 testov na deň. V malých projektoch, je toto množstvo dostačujúce, ale pri väčších je potrebné kúpiť licenciu, ktorá povoľuje 500 testov na užívateľa na deň. Táto verzia však stojí $56,000 na rok. V prípade, že firma má záujem o viac testov, je nutné dohodnúť sa na personalizovanej ponuke.

3.3.7 Ostatné programy

Okrem spomínaných programov existuje na generovanie Java unit testov množstvo iných alternatív. Z tých verejne dostupných však väčšinou ide menšie projekty ako napr. TestMe [65] alebo Machinet [66], založený na modeli umelej inteligencie GPT4. Tie pri skúšaní na projekte Uniqway nedokázali generovať dostatočné množstvo testov alebo zaručiť ich správnosť.

3.3.8 Zhrnutie

Po vyskúšaní dostupných technológií sa Randoop ukázal ako jediný voľne dostupný nástroj vhodný na automatizované generovanie regresných testov.

Kapitola 4

Implementácia prototypu a zhodnotenie výsledkov

Táto kapitola sa zaoberá implementáciou prototypu oboch rozšírení podľa analýzy a návrhov z predchádzajúcich kapitol. Na začiatku sa nachádza popis použitých vývojových technológií, potom nasleduje popis implementácie najprv k rozšíreniu týkajúceho sa databázovej schémy a následne regresných testov. Na konci každej časti nasleduje zhodnotenie terajšej účinnosti vykonanej zmeny. Posledná sekcia sa venuje vytvorenej dokumentácií pre obe rozšírenia.

4.1 Technológie použité pre vývoj

Pre vývoj softwaru dnes existuje veľké množstvo dostupných nástrojov. Nasledujúca sekcia popisuje použité nástroje, ktoré ešte neboli predstavené v sekcií o analýze systému Uniqway 1.3.

4.1.1 Intellij IDEA

Pre samotný vývoj bolo zvolené prostredie Intellij IDEA od spoločnosti JetBrains s.r.o. Prostredie sa zameriava na vývoj softwaru v jazykoch Java, Kotlin, Scala, Groovy a iných jazykoch založených na JVM. Podobne ako iné IDE poskytuje funkcionality pre zjednodušenie vývoja ako napr. statickú analýzu kódu, automatické dopĺňanie slov alebo prostredie pre debugging. Veľkou výhodou Intellij IDEA je však široký ekosystém pluginov alebo dokonca priamych integrácií pre rôzne ďalšie nástroje či už od spoločnosti JetBrains alebo softwaru tretích strán [35].

4.1.2 JUnit

JUnit je populárny open-source framework pre písanie unit testov v Jave. Vývojárom umožňuje jednoduchým spôsobom vytvárať množstvo testov na kontrolu správnosti softwaru. Pri písaní testov sa vyznačuje využívaním anotácií [32].

Anotácia *@Test* je povinným označením každej testovacej metódy. Pre zjednodušenie práce poskytuje napr. anotácie *@BeforeEach* a *AfterEach* na časti kódu vykonávané pred, resp. po každom teste. Pre zložitejšie prípady existujú aj anotácie ako *@BeforeAll*, *@AfterAll*, *@RepeatedTest*, *@Parameters*, *@RunWith* a mnoho ďalších [67].

JUnit patrí do rodiny frameworkov xUnit založených na podobných princípoch. Patria sem aj frameworky ako *PyUnit*, *NUnit* a *CppUnit*, ktoré jednotlivo testujú programy v jazykoch Python, .NET a C++ [68].

4.1.3 psql

Psql alebo PostgreSQL interactive terminal je interaktívny terminál pre prácu s PostgreSQL databázou. Umožňuje vytvárať nové databázy, pripájať sa k nim a posielať SQL príkazy. Ide o rozšírenie, ktoré je typicky inštalované spolu so systémom PostgreSQL bližšie popísanom v sekcií [69].

4.1.4 pgAdmin

Obdobou programu psql s grafickým rozhraním je program pgAdmin. Ide o jeden z najpopulárnejších open-source programov na správu PostgreSQL databáze [70]. Funkcionálne má podobné vlastnosti ako psql, vďaka GUI je však orientácia v databáze rýchlejšia a prehľadnejšia.

4.2 Postup implementácie pri migračných skriptoch

Pre spustenie migračných skriptov bolo najprv potrebné navrhnúť vhodné rozhranie na pripojenie k lokálnej databázi. Toto rozhranie bolo následne rozšírené tak aby podporovalo výmenu rôznych typov databáze cez jednoduché nastavenie konfiguračného súboru.

Po pripojení databáze bolo vytvorených niekoľko funkcií pre prácu s evolutions tak aby pokrývali požiadavky na riešenie zo sekcie 3.2.3. Prevažne ide o adaptáciu funkcií a dátových typov poskytovaných frameworkom Play, keďže tým bude možné najvierohodnejšie simulovať skutočné chovanie evolutions za behu aplikácie. Týchto funkcií je síce veľmi limitovaný počet, ale pri tvorbe prototypu boli dostačujúce na splnenie požiadavkov.

Po vytvorení funkcií boli vytvorené unit testy na kontrolu ako celého behu evolutions tak kontrolu niektorých vybraných skriptov.

Na konci sekcie je ešte poznámka o chybách nájdených a opravených počas implementácie.

4.2.1 Databázové pripojenie

Na vytvorenie lokálnej databáze v štýle Uniqway je najprv potrebné nainštalovať si systém PostgreSQL. Do databáze je potom potrebné nainštalovať rozšírenie podľa manuálu v internej dokumentácií. Výsledkom je bežiaci PostgreSQL server.

Na pripojenie sa k bežiacej databázi bola vytvorená abstraktná trieda *DatabaseProvider*, ktorú je možné implementovať pomocou tzv. *Factory* návrhového vzoru pre rôzne typy databázových pripojení.

Konkrétna implementácia, napr. pre PostgreSQL potom za pomoci knižnice Typesafe Config [71] načíta z konfiguračného súboru parametre zadané užívateľom a vytvorí databázový objekt kompatibilný s Play frameworkom.

Výpis kódu 4.1 Konfiguračný súbor pre PostgreSQL

```
db {
    type = "postgresql"
    driver = org.postgresql.Driver
    url = "jdbc:postgresql://localhost:5432/uniqway_test"
    username = myusername
    password = "mypassword"
}
```

4.2.2 Pomocné funkcie

Funkcie pre načítanie a spúšťanie evolutions sú implementované ako Java metódy v triede *EvolutionsHelpers*. Metódy využívajú funkcie poskytnuté Play frameworkom v balíčku Evolutions [72]. Na základe požiadavkov zo sekcie 3.2.3 boli implementované funkcie pre načítanie evolutions z adresára *conf/evolutions* a pre aplikovanie jedného, niekoľkých alebo všetkých Up a Down skriptov.

4.2.3 Vytvorené testy

Pre vyskúšanie fungovania pomocných funkcií a základného otestovania behu migračných skriptov bolo vytvorených niekoľko testov vo frameworku JUnit, ktoré spúšťajú evolutions jednotlivo a hromadne.

Pri spúšťaní týchto testov bolo odhalených zopár chýb v evolution skriptoch, ktoré znemožňovali ich plynulý chod. Tieto chyby boli následne v evolutions opravené a boli na ne vytvorené príslušné testy.

4.3 Postup implementácie pri regresných testoch

Po vyskúšaní a vybraní funkčného nástroja v sekcií 3.3.3 sa implementačná časť zaoberá prevažne jeho zasadením do projektu Uniqway.

Pred spustením Randoopu je nutné zozbierať všetky závislosti a zoznam tried potrebných na jeho spustenie. Tomu sa venujú prvé dve časti tejto sekcie. V ďalšej časti nasleduje popis procesu automatizovaného generovania testov a výberu správnych parametrov. Posledná časť je venovaná prehľadu pokrytia pred a po vygenerovaní automatizovaných testov.

4.3.1 Pridanie závislostí

Randoop na svoje fungovanie potrebuje tri typy závislostí. Prvým je samotný .jar súbor Randoopu, ktorý je možné stiahnuť z jeho webovej stránky [60] a triviálne vložiť do projektu.

4.3.1.1 Externé závislosti

Druhým typom sú všetky externé závislosti potrebné na spustenie testovaného kódu. V prípade projektu Uniqway to predstavuje viac než 300 súborov. Keďže projekt je kompilovaný cez sbt, v konfiguračnom súbore sa nachádzajú iba mená hlavných závislostí. Tie si následne sťahujú ďalšie závislosti pre ich fungovanie, čím vzniká rozsiahly strom vzájomných náväzností.

Zoznam týchto závislostí je získaný pomocou integrovanej funkcionality **artifacts** v Intellij IDEA [73]. Jej výstupom je adresár so všetkými .jar súbormi, na ktorých projekt závisí. Presný popis tohoto procesu je pre generovanie budúcich verzií popísaný v priloženej dokumentácií.

Alternatívnym spôsobom ako získať všetky závislosti je vytvorenie tzv. *Fat/Uber JAR*, teda .jar súboru projektu aj so závislosťami. Typicky sa toho dosahuje spustením príslušného príkazu v zostavovacom nástroji aplikácie. Pre sbt existuje pre túto potrebu plugin sbt-assembly [74].

Po početných pokusoch o vygenerovanie Fat JAR projektu Uniqway pomocou tohto pluginu sa však nepodarilo vytvoriť uspokojivý súbor. Plugin opakovane narážal na problém v konfliktoch duplicitných závislostí. Tie vyplývajú zo spôsobu akým sú závislosti pridávané do sbt. Úprava konfiguračného sbt súboru by znamenala reorganizáciu pridávania všetkých závislostí do projektu, čo by mohlo mať množstvo neočakávaných následkov pre metódy v zdrojovom kóde. V dočasnej dobe teda Randoop nie je pre takto vytvorený Fat JAR schopný vytvoriť testy.

4.3.1.2 Triedy projektu

Tretím typom sú skompilované triedy, pre ktoré majú byť vytvorené testy. Tie môžu byť vytvorené pomocou príkazu **package** v nástroji sbt, ktorý vytvorí .jar súbor celého projektu. Ďalšou možnosťou je kompilácia celej aplikácie a nastavenie cesty. Skompilované triedy sa pri použití nástroja sbt typicky nachádzajú v priečinku **target/scala-¡verzia¿/classes/**. Ešte ďalšou možnosťou je vygenerovať tieto súbory vo funkcionalite *artifacts*, ktorá bola použitá pre tvorbu zoznamu závislostí.

Pre vytvorenie zoznamu tých tried, pre ktoré má zmysel vytvárať testy bol napísaný shell skript získavajúci plne kvalifikované mená všetkých tried v adresári *app*. Tie sú Randoopu predané v parametri *–classlist* ako textový súbor.

4.3.1.3 Generovanie testov

Pre generovanie testov boli vytvorené dva shell skripty na tvorbu testov pre celý projekt a pre zadané triedy. Tie najprv skompilujú celý projekt a následne predajú Randoopu cestu ku všetkým potrebným závislostiam. Skripty potom spustia Randoop s vhodne nastavenými parametrami. Vytvorené testy sa potom nachádzajú v priečinku *RandoopGeneratedTests*.

4.3.2 Zmeny v code coverage

Pre zistenie vhodného časového limitu poskytnutého Randoopu na generovanie testov bol prevedený pokus na troch rôznych časových intervaloch, 1 minúta, 5 minút a 10 minút. Tabuľka 4.1 ukazuje počet testov vygenerovaných Randoopom v danom časovom limite.

Čas	Počet vygenerovaných testov	Počet prechádzajúcich testov
1 minúta	2434	2434
5 minút	11074	11074
10 minút	20058	20058

Tabuľka 4.1 Počet vygenerovaných testov

Na porovnanie pokrytia boli vybrané niektoré priečinky zo sekcie Code coverage 2.8. Tabuľky 4.2, 4.3 a 4.4 ukazujú nové percento pokrytia a porovnávajú rozdiely s pokrytím zo sekcie 2.8.

Pokrytie 1 minúta								
Priečinok	Class	Rozdiel	Method	Rozdiel	Line	Rozdiel	Branch	Rozdiel
actions	83%	+33	46%	+6	45%	+7	30%	+3
controllers	19%	+17	4%	+4	7%	+7	0%	+0
dao	51%	+46	22%	+20	14%	+13	0%	+0
mails	56%	+19	28%	+6	31%	+7	66%	+0
models	79%	+32	49%	+15	52%	+15	44%	+1
security	50%	+34	6%	+3	5%	+5	0%	+0
services	68%	+35	36%	+10	37%	+7	33%	+1
utils	74%	+18	56%	+15	62%	+12	62%	+1

Tabuľka 4.2 Pokrytie 1 minúta

Ako sa dalo očakávať, percento pokrytia a počet testov stúpa spolu s prideleným časom. Celkovo sa ukazuje, že generovanie testov po dobu 5 minút prináša najlepší kompromis pokrytia vzhľadom k počtu vygenerovaných testov. Ich reálna využiteľnosť sa však ukáže až s odstupom času pri pokračujúcom vývoji.

Pokrytie 5 minút								
Priečinok	Class	Rozdiel	Method	Rozdiel	Line	Rozdiel	Branch	Rozdiel
actions	91%	+41	62%	+22	55%	+17	38%	+11
controllers	29%	+27	9%	+9	12%	+12	0%	+0
dao	85%	+80	37%	+35	24%	+23	0%	+0
mails	100%	+63	60%	+38	56%	+32	66%	+0
models	97%	+50	82%	+48	83%	+46	50%	+7
security	83%	+67	38%	+35	29%	+29	0%	+0
services	94%	+61	66%	+40	54%	+24	36%	+4
utils	84%	+28	71%	+30	73%	+23	62%	+1

Tabuľka 4.3 Pokrytie 5 minút

Pokrytie 10 minút								
Priečinok	Class	Rozdiel	Method	Rozdiel	Line	Rozdiel	Branch	Rozdiel
actions	100%	+50	68%	+28	59%	+21	44%	+17
controllers	31%	+29	10%	+10	13%	+13	0%	+0
dao	98%	+93	42%	+40	27%	+26	0%	+0
mails	100%	+63	71%	+49	66%	+42	66%	+0
models	98%	+51	91%	+57	91%	+54	50%	+7
security	83%	+67	54%	+51	41%	+41	0%	+0
services	96%	+63	72%	+46	58%	+28	37%	+5
utils	87%	+31	77%	+36	77%	+27	63%	+2

Tabuľka 4.4 Pokrytie 10 minút

4.4 Dokumentácia

K obom implementovaným prototypom bola napísaná dokumentácia popisujúca ich správne použitie v podobe *Readme* textového súboru. V prípade evolutions ide najmä o popis postupu pripojenia databáze a písania nových testov. Pri regresných testoch je naznačený správny postup pre ich generovanie a vkladanie závislostí.

Kapitola 5
Návrhy do budúcnosti

Backendová časť systému Uniqway tvorí rozsiahly systém rozvetvený na množstvo rozdielnych komponent a existuje v ňom prakticky neobmedzený priestor pre pridávanie nových vylepšení. Niektoré návrhy zozbierané na základe analýzy už boli predstavené v sekcií *Všeobecné možnosti vylepšení* 3.1. Ďalším priestorom by mohlo byť rozšírenie implementovaných prototypov.

5.1 Rozšírenie testovania migračných skriptov

V oblasti testovania migračných skriptov je možné rozšíriť terajšie riešenie o množstvo nových testov. V aplikácií sa nachádza niečo vyše 130 migračných skriptov, každý s Up a Down časťou, pričom terajší prototyp testuje iba ich malú podmnožinu. Okrem nich sa s kontrolou a opravou terajších skriptov otvára príležitosť na väčšie využitie databáze pri integračných testoch.

5.2 Rozšírenie automatizovaného testovania

V oblasti automatického generovania testov zase množstvo nástrojov ponúka možnosti ako okrem regresných testov aj automaticky odhaliť chyby v zdrojov kóde. Jednou z variant je rozšírenie Randoopu, ktorý ponúka možnosť vytvárať *error-revealing tests* pre tvorbu zlyhávajúcich testov.

Kapitola 6

Záver

Cieľom tejto bakalárskej práce bolo analyzovať proces testovania na bakcendovej časti systému Uniqway a navrhnúť vhodné vylepšenia a rozšírenia. Posledným cieľom bolo implementovať funkčný prototyp tohoto riešenia vhodný pre zasadenie do projektu.

V priebehu analýzy bol najprv preskúmaný systém Uniqway. Dôraz bol kladený na zoznámenie sa s konkrétnymi technológiami využívanými v projekte. Za spomenutie stojí framework Play využívajúci MVC architektúru, na ktorej stojí celá aplikácia a PostgreSQL databáza verzovaná pomocou migračných skriptov.

Ďalej bol v kontexte aplikácie analyzovaný celý testovací proces. Boli vysvetlené použité metodológie testovania a postupy dodržiavané pri ich písaní. Zaujímavé zistenia priniesol rozbor štruktúry a testov, ktorý ukázal prevahu unit testov a nedostatky pri integračnom testovaní. Rozbor pokrytia testami odhalil aj nedostatky testovania v určitých častiach aplikácie. Ako najzásadnejší nedostatok sa však ukázala absencia testovania databáze. Napriek tomu sa súčasný stav testovacieho procesu dá považovať za dobrý a nové časti zdrojového kódu bývajú raidne otestované.

V návrhovej časti bolo predstavených niekoľko možností na vylepšenie. Pre vylepšenia týkajúce sa testovania tvorby databázovej schémy a regresných testov bol predstavený aj detailný opis problému a stanovené ciele pre ich implementáciu.

Následne bol v implementačnej časti implementovaný funkčný prototyp oboch vylepšení. Testovanie databázovej schémy počas implementácie vyriešilo problém nefunkčných migračných skriptov a umožnilo budúce využitie databáze pri písaní testov. Automatizácia regresných testov mala za následok zvýšenie pokrytia zdrojového kódu testami. Ich reálna využiteľnosť sa ale ukáže až s postupom času.

Celkovo došlo k podrobnej analýze testovacieho procesu backendovej časti Uniqway a implementácií dvoch nových vylepšení, čo je v súlade s cieľmi tejto bakalárskej práce.

Bibliografia

1. COMPUTERWORLD STAFF. *Moth in the Machine: Debugging the origins of 'bug'* [online]. Computerworld, 2011-09 [cit. 2023-04-24]. Dostupné z : https://www.computerworld.com/article/2515435/moth-in-the-machine--debugging-the-origins-of--bug-.html.
2. MOSEMAN, Andrew. *The internet's first message was sent 45 Years Ago Today* [online]. Popular Mechanics, 2014-10 [cit. 2023-04-24]. Dostupné z : https://www.popularmechanics.com/culture/web/a13020/the-internet-first-message-45-years-ago-17366635/.
3. PAUL, Ian. *Scary steam for linux bug erases all the personal files on your PC* [online]. PCWorld, 2015-01 [cit. 2023-04-24]. Dostupné z : https://www.pcworld.com/article/431317/scary-steam-for-linux-bug-erases-all-the-personal-files-on-your-pc.html.
4. *Rozjeď se s námi!* [online]. První český univerzitní carsharing Uniqway, 2022-10 [cit. 2023-05-09]. Dostupné z : http://web.archive.org/web/20221006042829/https://www.uniqway.cz/.
5. PROUZA, Petr. *Transformace systému z monolitické architektury do architektury mikroslužeb* [online]. 2021-05. [cit. 2023-05-09]. Dostupné z : https://dspace.cvut.cz/handle/10467/94555.
6. PONCE, Francisco; MÁRQUEZ, Gastón; ASTUDILLO, Hernán. Migrating from monolithic architecture to microservices: A Rapid Review. In: *2019 38th International Conference of the Chilean Computer Science Society (SCCC)*. 2019, s. 1–7. Dostupné z DOI: 10.1109/SCCC49216.2019.8966423.
7. SELLA, Orr. *Orrsella/SBT-stats: An SBT plugin for source code statistics* [online]. GitHub, 2017-09 [cit. 2023-05-09]. Dostupné z : https://github.com/orrsella/sbt-stats.
8. *What is Java technology and why do I need it?* [online]. Java.com, 2022 [cit. 2023-05-09]. Dostupné z : https://www.java.com/en/download/help/whatis_java.html.
9. *The high velocity web framework for Java and scala* [online]. Play Framework - Build Modern ; Scalable Web Apps with Java a Scala, 2021 [cit. 2023-05-09]. Dostupné z : https://www.playframework.com/.
10. *Ebean Orm* [online]. Ebean ORM for Java; Kotlin, 2023 [cit. 2023-05-09]. Dostupné z : https://ebean.io/.
11. *PostgreSQL: The World's Most Advanced Open Source Relational Database*. PostgreSQL, 2023-02. Dostupné tiež z: https://www.postgresql.org/.

12. HICKFORD, Jonathan. *Using migration scripts in database deployments* [online]. Simple Talk, 2021-08 [cit. 2023-05-09]. Dostupné z : https://www.red-gate.com/simple-talk/databases/sql-server/database-administration-sql-server/using-migration-scripts-in-database-deployments/.
13. *Managing database evolutions* [online]. Evolutions - 2.8.x, 2023 [cit. 2023-05-09]. Dostupné z : https://www.playframework.com/documentation/2.8.x/Evolutions.
14. *What is Maven?* [online]. Maven, 2023-04 [cit. 2023-05-09]. Dostupné z : https://maven.apache.org/what-is-maven.html.
15. *Gradle Build Tool* [online]. Gradle, 2023 [cit. 2023-05-09]. Dostupné z : https://gradle.org/.
16. *The Interactive Build Tool* [online]. sbt, 2023 [cit. 2023-05-09]. Dostupné z : https://www.scala-sbt.org/.
17. ULLAH, Sami. *A brief history of software testing* [online]. Salsa Digital, 2019-12 [cit. 2023-05-09]. Dostupné z : https://salsa.digital/insights/a-brief-history-of-software-testing.
18. TRAN, Hoang Nam. *Optimalizace výkonu backendu služby pro sdílení vozidel Uniqway.* 2021. Dostupné tiež z: https://dspace.cvut.cz/handle/10467/102244.
19. KANER, Cem; BACH, James. Waterfall lifecycles pit reliability against time. In: *Lessons learned in software testing.* Wiley, 2001, 156‘–157.
20. ACTITIME. *How to use The waterfall method in any project: ActiTIME Guide* [online]. actiTIME, 2021-09 [cit. 2023-05-09]. Dostupné z : https://www.actitime.com/project-management/waterfall-model.
21. CSER, Tamas. *The cost of finding bugs later in the SDLC* [online]. Functionize Inc., 2023-01 [cit. 2023-05-09]. Dostupné z : https://www.functionize.com/blog/the-cost-of-finding-bugs-later-in-the-sdlc.
22. GREGORY, Janet; CRISPIN, Lisa. *More Agile Testing: Learning Journeys for the whole team.* Addison-Wesley, 2015.
23. ODEYEMI, Dipo. *Overcoming the challenges of Agile Software Development* [online]. LinkedIn, 2020-02 [cit. 2023-05-09]. Dostupné z : https://www.linkedin.com/pulse/overcoming-challenges-agile-software-development-dipo-odeyemi.
24. BECK, Kent. Section I: Money Example. In: *Test-driven development by example.* Addison-Wesley, 2015.
25. DIFFBLUE. *2020 devops and testing report* [online]. Diffblue, 2020-05 [cit. 2023-05-09]. Dostupné z : https://www.diffblue.com/DevOps/research_papers/2020-devops-and-testing-report/.
26. SAĞLAM, İsmail Alper. *TDD is not about testing but the design: ICTERRA Information and Communication Technologies* [online]. ICterra, 2017-01 [cit. 2023-05-09]. Dostupné z : https://www.icterra.com/tdd-is-not-about-testing-but-the-design/.
27. KITNER, Radek. *Typy testování software* [online]. Druhy testů (Typy testování software), 2021 [cit. 2023-05-09]. Dostupné z : https://kitner.cz/testovani_softwaru/typy-testovani-software-trideni-testu/.
28. PARMAR, Deepak. *Exploratory testing.* Atlassian. Dostupné tiež z: https://www.atlassian.com/continuous-delivery/software-testing/exploratory-testing.
29. FOWLER, Martin. *On the diverse and fantastical shapes of testing* [online]. martinfowler.com, 2021-06 [cit. 2023-05-09]. Dostupné z : https://martinfowler.com/articles/2021-test-shapes.html.

30. MAHAJAN, Ratul. *The networking test pyramid* [online]. Intentionet, 2021-06 [cit. 2023-05-09]. Dostupné z : https://www.intentionet.com/blog/the-networking-test-pyramid/.

31. SCHAFFER, André. *Testing of microservices* [online]. Spotify Engineering, 2018-01 [cit. 2023-05-09]. Dostupné z : https://engineering.atspotify.com/2018/01/testing-of-microservices/.

32. *The 5th major version of the programmer-friendly testing framework for Java and the JVM*. JUnit 5, 2023. Dostupné tiež z: https://junit.org/junit5/.

33. *Mockito Framework Site* [online]. Mockito framework site [cit. 2023-05-09]. Dostupné z : https://site.mockito.org/.

34. KG; CONTRIBUTORS. *Coverage counters*. JaCoCo, 2023. Dostupné tiež z: https://www.jacoco.org/jacoco/trunk/doc/counters.html.

35. S.R.O., JetBrains. *Code coverage: Intellij idea*. IntelliJ IDEA Help, 2023-04. Dostupné tiež z: https://www.jetbrains.com/help/idea/code-coverage.html.

36. HOFER, Joachim; STRINGER, Michael [online]. sbt-jacoco, 2023 [cit. 2023-05-09]. Dostupné z : https://www.scala-sbt.org/sbt-jacoco/index.html.

37. IVANKOVIC, Marko; PETROVIC, Goran; JUST, René; FRASER, Gordon. Code coverage at Google. In: *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2019, s. 955–963.

38. ARGUELLES, Carlos; IVANKOVIĆ, Marko; BENDER, Adam. *Code coverage best practices* [online]. Google Testing Blog, 2020-08 [cit. 2023-05-09]. Dostupné z : https://testing.googleblog.com/2020/08/code-coverage-best-practices.html.

39. HU, Tom. *The case against 100% code coverage* [online]. Codecov, 2022-08 [cit. 2023-05-09]. Dostupné z : https://about.codecov.io/blog/the-case-against-100-code-coverage/.

40. COMMONS, Wikimedia. *File:MediaWiki 1.10 database schema.png — Wikimedia Commons, the free media repository*. 2007. Dostupné tiež z: https://commons.wikimedia.org/w/index.php?title=File:MediaWiki_1.10_database_schema.png&oldid=468579571. [Online; accessed 30-April-2023].

41. *Testing with databases* [online]. Java Testing With Databases - 2.8.x, 2023 [cit. 2023-05-09]. Dostupné z : https://www.playframework.com/documentation/2.8.x/JavaTestingWithDatabases.

42. *Class Evolutions* [online]. Evolutions (play 2.8.19) [cit. 2023-05-09]. Dostupné z : https://www.playframework.com/documentation/2.8.x/api/java/play/db/evolutions/Evolutions.html.

43. *H2 Database Engine*. H2 database engine. Dostupné tiež z: https://www.h2database.com/html/main.html.

44. *H2 database*. Developing-with-the- H2- Database - 2.8.x. Dostupné tiež z: https://www.playframework.com/documentation/2.8.x/Developing-with-the-H2-Database.

45. *Documentation - flyway by Redgate, Database Migrations Made Easy.* Documentation. Dostupné tiež z: https://flywaydb.org/documentation/.

46. *Flyway by Redgate, Database Migrations Made Easy.* Migrations. Dostupné tiež z: https://flywaydb.org/documentation/concepts/migrations.

47. *Cherry Pick*. Flyway by Redgate, Database Migrations Made Easy. Dostupné tiež z: https://flywaydb.org/documentation/configuration/parameters/cherryPick.

48. *Target*. Database migrations made easy. Target - flyway by Redgate. Dostupné tiež z: https://flywaydb.org/documentation/configuration/parameters/target.
49. TAKAHASHI, Toshiyuki. *flyway-play*. GitHub. Dostupné tiež z: https://github.com/flyway/flyway-play.
50. *The LIQUIBASE Community: The Database DevOps Community*. Liquibase.org, 2022. Dostupné tiež z: https://www.liquibase.org/.
51. Changelog. Dostupné tiež z: https://docs.liquibase.com/concepts/changelogs/home.html.
52. Update-count. Dostupné tiež z: https://docs.liquibase.com/commands/update/update-count.html.
53. Update-to-tag. Dostupné tiež z: https://docs.liquibase.com/commands/update/update-to-tag.html.
54. [online]. Update-one-changeset [cit. 2023-05-09]. Dostupné z : https://docs.liquibase.com/commands/update/update-one-changeset.html.
55. *Ticketfly/Play-liquibase: Play liquibase module*. GitHub. Dostupné tiež z: https://github.com/Ticketfly/play-liquibase.
56. CSALLNER, Christoph; SMARAGDAKIS, Yannis. *Welcome to jcrasher*. JCrasher. Dostupné tiež z: https://ranger.uta.edu/~csallner/jcrasher/.
57. *Automated JUnit Generation - 80% Code Coverage, or Better* [online]. Automated JUnit Generation — Agitar Technologies: Putting Java to the Test [cit. 2023-05-09]. Dostupné z : http://www.agitar.com/solutions/products/automated_junit_generation.html.
58. GAMBI, Alessio; JAHANGIROVA, Gunel; RICCIO, Vincenzo; ZAMPETTI, Fiorella. SBST Tool Competition 2022. In: *2022 IEEE/ACM 15th International Workshop on Search-Based Software Testing (SBST)*. 2022, s. 25–32. Dostupné z DOI: 10.1145/3526072.3527538.
59. DEVROEY, Xavier; PANICHELLA, Sebastiano; GAMBI, Alessio. Java Unit Testing Tool Competition: Eighth Round. In: Seoul, Republic of Korea: Association for Computing Machinery, 2020, s. 545–548. ICSEW'20. ISBN 9781450379632. Dostupné z DOI: 10.1145/3387940.3392265.
60. *Randoop Automatic unit test generation for Java*. Randoop. Dostupné tiež z: https://randoop.github.io/randoop/.
61. *About*. EvoSuite. Dostupné tiež z: https://www.evosuite.org/evosuite/.
62. *Evosuite - automated generation of JUnit Test Suites for java classes*. GitHub. Dostupné tiež z: https://github.com/EvoSuite/evosuite.
63. *Diffblue cover key features*. Diffblue. Dostupné tiež z: https://www.diffblue.com/cover-key-features.
64. *Diffblue's automated, human-readable Java unit tests are now free for commercial use*. Diffblue, 2021-03. Dostupné tiež z: https://www.diffblue.com/blog/testing/java/ai/diffblues-automated-human-readable-java-unit-tests-are-now-free-for-commercial-use/.
65. *Quick-start*. WeirdDev, 2018-08. Dostupné tiež z: https://weirddev.com/testme/.
66. *AI assistant for developers*. Machinet, 2023. Dostupné tiež z: https://www.machinet.net/.
67. BRANNEN, Sam; BECHTOLD, Stefan; LINK, Johannes; MERDES, Matthias; PHILIPP, Marc; RANCOURT, Juliette de; STEIN, Christian. *JUnit 5 User Guide*. JUnit 5 user guide, 2023-04. Dostupné tiež z: https://junit.org/junit5/docs/current/user-guide/#writing-tests-annotations.

68. *Xunit testing frameworks*. InterSystems, 2023. Dostupné tiež z: https://docs.intersystems.com/irislatest/csp/docbook/DocBook.UI.Page.cls?KEY=TUNT_TESTINGFRAMEWORKS.

69. *PSQL*. PostgreSQL Documentation. Dostupné tiež z: https://www.postgresql.org/docs/current/app-psql.html.

70. *PgAdmin*. pgAdmin. Dostupné tiež z: https://www.pgadmin.org/.

71. PENNINGTON, Havoc. *Typesafe config*. Typesafe Config. Dostupné tiež z: https://lightbend.github.io/config/.

72. *Class Evolutions*. Evolutions (play 2.8.19). Dostupné tiež z: https://www.playframework.com/documentation/2.8.x/api/java/play/db/evolutions/Evolutions.html.

73. *Artifacts: Intellij idea*. IntelliJ IDEA Help, 2022-03. Dostupné tiež z: https://www.jetbrains.com/help/idea/working-with-artifacts.html#build_artifacts.

74. SBT. *SBT-assembly* [online]. GitHub [cit. 2023-05-09]. Dostupné z : https://github.com/sbt/sbt-assembly.

Obsah priloženého média

- `readme.txt` ..... stručný popis obsahu média
- `src`
  - `Migračné skripty` ..... zdrojové kódy implementácie testovania migračných skriptov
  - `Regresné testy` ..... zdrojové kódy implementácie generovania regresných testov
  - `thesis` ..... zdrojová forma práce vo formáte LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce vo formáte PDF