Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Automatizované testování webové aplikace

Dominik Sága

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Dominik Sága

Informatika

Název práce

Automatizované testování webové aplikace

Název anglicky

Test automation for web application

Cíle práce

Bakalářská práce se zabývá problematikou automatizovaného testování webové aplikace.
Cílem teoretické části práce je vymezení základních pojmů a procesů z oblasti testování software, včetně popisu problematiky automatizace testování a analýzy využívaných nástrojů.
Cílem praktické části je vytvoření testovacího scénáře a automatického testu pro webovou aplikaci v bankovním sektoru, a následné porovnání časové náročnosti s testem manuálním.

Metodika

Teoretická část bakalářské práce bude založena na studiu a analýze odborných literárních zdrojů a zkušeností autora z praxe. V praktické části bude vytvořen testovací scénář a automatizovaný test, pro který bude využit vybraný nástroj vycházející z analýzy v teoretické části práce. Výsledkem práce bude porovnání celkové časové náročnosti automatizovaného testu s manuálním testem.

Doporučený rozsah práce

30-60 stran

Klíčová slova

automatizované testování, webová aplikace, kvalita software, testovací scénář

Doporučené zdroje informací

BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽEL, Peter SVOBODA, Zdeněk GRÖSSL, Martin KOMÁREK, Ondřej MACEK a Radoslav MLYNÁŘ. Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu. Praha: Grada, 2016. Profesionál. ISBN 978-80-247-5594-6.

NAIK, Kshirasagar, Priyadarshi TRIPATHY. Software Testing and Quality Assurance: Theory and Practice, John Wiley & Sons, Inc, 2008, ISBN 978-0-471-78911-6.

PATTON, Ron. Testování softwaru. Praha: Computer Press, 2002. Programování. ISBN 80-7226-636-5.

Předběžný termín obhajoby

2022/23 LS – PEF

Vedoucí práce

Ing. Dana Vynikarová, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 31. 10. 2022

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 24. 11. 2022

doc. Ing. Tomáš Šubrt, Ph.D.

Děkan

V Praze dne 02. 02. 2023

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci „Automatizované testování webové aplikace“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 15.03.2023 _____

Poděkování

Rád bych touto cestou poděkoval Ing. Daně Vynikarové, Ph.D. za odborné vedení, konzultace a cenné rady při tvorbě této práce. Dále bych chtěl poděkovat Komerční bance, která mi umožnila v jejím prostředí realizovat praktickou část práce.

Automatizované testování webové aplikace

Abstrakt

Bakalářská práce se zabývá problematikou automatizovaného testování webových aplikací. Cílem této práce je vytvoření testovacího scénáře a k němu příslušný automatizovaný test v bankovním prostředí. V teoretické části jsou nejdříve vysvětleny zásadní pojmy z oblasti testování softwaru, jako je definice testování a jeho cíle, objasnění problematiky softwarových defektů, vysvětlení jednotlivých metod testování, popis úrovní funkčního testování, rozdělení testovací dokumentace a vymezení jednotlivých rolí v testovacím týmu. Dále se teoretická část věnuje shrnutí problematiky automatizovaného testování, kde popisuje rozdíly a výhody proti manuálnímu testování a objasnění, jaké testy jsou vhodné automatizovat, a naopak jaké testy pro automatizaci vhodné nejsou. Teoretická východiska jsou následně zakončena analýzou využívaných nástrojů pro automatizované testování webových aplikací. V praktické části je nejprve popsána aplikace a funkcionalita, na které testování probíhalo. Na základě poznatků z teoretické části práce jsou následně vytvořeny testovací scénáře, které slouží jako podklad pro provedení testů. Nejprve je popsán průběh manuálního testování, na který navazuje podrobný popis tvorby a provedení automatizovaných testů. Výstupem práce je následné porovnání celkové časové náročnosti obou testovacích metod.

Klíčová slova: automatizované testování softwaru, testovací scénář, webová aplikace, uživatelské rozhraní, bankovní sektor

Test automation for web application

Abstract

The bachelor thesis deals with the issue of automated testing of web applications. The aim of this thesis is to create a test scenario and the corresponding automated test in a banking environment. The theoretical part first explains the fundamental concepts in the field of software testing, such as the definition of testing and its objectives, clarification of software defects, explanation of different testing methods, description of functional testing levels, division of testing documentation and definition of different roles in the testing team. Furthermore, the theoretical part is devoted to a summary of the issues of automated testing, describing the differences and advantages against manual testing and clarifying which tests are suitable for automation and conversely, which tests are not suitable for automation. The theoretical part is then concluded with an analysis of the tools used for automated testing of web applications. The practical part first describes the application and functionality on which the testing was performed. Based on the findings from the theoretical part of the thesis, test scenarios are subsequently created to serve as a basis for the execution of the tests. First, the manual testing process is described, followed by a detailed description of the creation of and execution of automated tests. The output of the work is a subsequent comparison of the overall time consumption of the two testing methods.

Keywords: automated software testing, test scenario, web application, user interface, banking sector

Obsah

1 Úvod **10**

2 Cíl práce a metodika **11**

2.1 Cíl práce 11

2.2 Metodika 11

3 Teoretická východiska **12**

3.1 Testování softwaru 12

3.1.1 Co je testování 12

3.1.2 Cíle testování 13

3.1.3 Softwarový defekt 14

3.1.3.1 Proč defekty vznikají 16

3.1.3.2 Správná terminologie 17

3.1.4 Testovací dokumentace 17

3.1.4.1 Testovací plán 18

3.1.4.2 Testovací případ 18

3.1.4.3 Testovací scénář 18

3.1.5 Testovací tým 19

3.1.5.1 Manažer testování 19

3.1.5.2 Analytik testování 19

3.1.5.3 Tester 20

3.1.6 Životní cyklus testování softwaru 20

3.1.7 Rozdělení metod testování 22

3.1.7.1 Statické a dynamické testování 22

3.1.7.2 Testování černé, bílé a šedé skříňky 23

3.1.7.3 Funkční a nefunkční testování 23

3.1.7.4 Smoke a regresní testování 24

3.1.8 Úrovně funkčního testování 24

3.1.8.1 Jednotkové testování 24

3.1.8.2 Integrační testování 25

3.1.8.3 Systémové testování 25

3.1.8.4 Akceptační testování 25

3.2 Automatizované testování 26

3.2.1 Manuální vs automatizované testování 26

3.2.2 Výhody automatizovaného testování 27

3.2.3 Vhodné testy pro automatizaci ... 28
3.2.4 Nevhodné testy pro automatizaci ... 29
3.3 Analýza využívaných nástrojů ... 30
3.3.1 Selenium ... 30
3.3.2 Katalon Studio ... 31
3.3.3 Cypress ... 32
3.3.4 TestComplete ... 33
3.3.5 Závěr analýzy ... 33
4 Vlastní práce ... 35
4.1 Testovaná aplikace ... 35
4.2 Tvorba testovacího scénáře ... 35
4.3 Manuální testování ... 38
4.4 Automatizované testování ... 38
4.4.1 Struktura projektu ... 39
4.4.2 Identifikace webových prvků ... 41
4.4.3 Tvorba pomocných metod ... 43
4.4.4 Sestavení testovacích scénářů ... 46
4.4.5 Spuštění testů ... 47
5 Výsledky měření ... 49
5.1 Časová náročnost přípravy na testování ... 49
5.2 Časová náročnost provedení testů ... 50
5.3 Srovnání výsledků ... 51
6 Závěr ... 52
7 Seznam použitých zdrojů ... 53
8 Seznam obrázků, tabulek a grafů ... 56
8.1 Seznam obrázků ... 56
8.2 Seznam tabulek ... 56
8.3 Seznam grafů ... 56
Přílohy ... 57

1 Úvod

Softwarové aplikace se staly neodmyslitelnou součástí lidského života a jejich neustálý pokrok čím dál více ovlivňuje to, jak lidská společnost funguje. Svůj podíl na tom mají i webové aplikace, které jsou díky své snadné přístupnosti velmi populární. S tím, jak dokážou být dnešní aplikace komplexní, vzniká neustále větší potřeba jejich důkladného testování.

Testování je jedním z klíčových aspektů procesu vývoje softwaru, který zajišťuje, že je aplikace spolehlivá, robustní a uživatelsky přívětivá. Manuální testování je sice účinné, ale v případě větších projektů může být časově náročné, pracné a náchylné k chybám. Z tohoto důvodu vzniklo automatizované testování, které se začalo jevit jako ideální řešení těchto problémů. Automatizace může zkrátit dobu testování, zvýšit přesnost výsledků testů a umožnit testerům soustředit se na složitější a kritičtější scénáře. Díky automatizaci je také možné rychlejší a častější vydávání nových verzí produktů na trh, což je v dnešním konkurenčním prostředí velmi důležité.

Automatizace testování webových aplikací zahrnuje použití specializovaných nástrojů a frameworků pro automatizaci provádění testovacích scénářů. Tyto nástroje simulují akce a interakce uživatelů s webovou aplikací, ověřují reakce aplikace a generují hlášení, které pomáhají sledovat průběh a identifikovat chyby.

Téma bakalářské práce si autor zvolil na základě jeho profesní praxe v oblasti testování softwaru, kde se jako tester v bankovním sektoru věnuje manuálnímu i automatizovanému testování. Autor chce čtenáři tento obor přiblížit vysvětlením nezbytné teorie a následnou demonstrací řešení reálného problému z praxe.

Teoretická část práce se zabývá vysvětlením zásadních pojmů z oblasti testování softwaru, vyjasněním problematiky automatizovaného testování a analýzou využívaných nástrojů pro automatizaci testování webových aplikací.

V praktické části práce je popsán celý proces tvorby testovacích scénářů a automatizovaných testů v bankovním prostředí. V rámci této části je také změřena a následně porovnána časová náročnost manuálního a automatizovaného testu.

2 Cíl práce a metodika

2.1 Cíl práce

Cílem teoretické části práce je vymezení zásadních pojmů a procesů z oblasti testování softwaru, včetně popisu problematiky automatizovaného testování a analýzy využívaných nástrojů, sloužících pro automatizaci testování webových aplikací.

Cílem praktické části je vytvoření testovacího scénáře a automatizovaného testu pro webovou aplikaci v bankovním sektoru a následné změření a porovnání časové náročnosti automatizovaného testu s testem manuálním.

2.2 Metodika

Teoretická část bakalářské práce bude založena na studiu a analýze odborných literárních zdrojů a zkušeností autora z praxe. V praktické části bude nejprve vytvořen testovací scénář, na jehož základě bude proveden manuální a automatizovaný test. Pro tvorbu automatizovaného testu bude využit nástroj, který bude vybrán na základě analýzy z teoretické části práce. Výsledkem práce bude porovnání celkové časové náročnosti automatizovaného testu s manuálním testem.

3 Teoretická východiska

3.1 Testování softwaru

Softwarové systémy jsou nedílnou součástí každodenního života a lidé se s nimi setkávají prakticky všude, od podnikových aplikací až po spotřebitelské produkty. Většina lidí se již pravděpodobně ve svém životě dostala do situace, kdy nějaká aplikace nepracovala tak, jak by se od ní očekávalo. Defekty v softwaru se mohou projevovat různými způsoby a mohou mít různě závažné následky v závislosti na místě jejich výskytu. Například defekt zanesený v počítačové hře představuje znatelně menší riziko než defekt, vyskytující se v lékařských nebo vojenských systémech. Mezi problémy, které mohou vzniknout při nesprávně fungujícím softwaru lze zařadit například ztrátu peněz, času a obchodní pověsti společnosti a v nejhorších případech i zranění nebo smrt. Aby bylo možné těmto problémům co nejvíce předcházet, je potřeba vyvíjený software kvalitně testovat. [1]

3.1.1 Co je testování

Pro mnoho lidí, kteří o této problematice nemají moc velké povědomí, je testování často pouze manuálním průchodu aplikací testerem, který se snaží najít nějakou funkční nebo vizuální vadu. Testování je však mnohem zajímavější a komplexnější proces, který v sobě skrývá daleko více činností a metodik, než je pouhé klikání v aplikaci. Testování se zpravidla zabývá analýzou požadavků, plánováním testování, přípravou testovacích scénářů, přípravou testovacího prostředí, provedením testů a následném vyhodnocení testování. Jako celek tyto aktivity tvoří tzv. životní cyklus testování softwaru, který bude detailněji popsán po vysvětlení potřebných pojmů. [2]

Jednotná definice o tom, co to testování vlastně je, by se hledala nejspíše stěží. Většina autorů si pod tímto pojmem představuje něco trochu odlišného a podle toho pak i vyzní jejich interpretace definice. Obecně se však dá říci, že testování je systematické zkoumání daného softwarového produktu, jehož hlavním cílem je vyhodnocovat, zda reálný produkt odpovídá předem stanoveným požadavkům a zda splňuje možné potřeby koncových uživatelů. [3]

3.1.2 Cíle testování

Podobně jako je tomu u definic testování, existuje i mnoho různých cílů, které při testování softwaru mohou vznikat. To je zapříčiněno tím, že různé projekty mají různé potřeby a od toho se i odvíjejí dané cíle. Podle jednoho ze základních pravidel uváděných ve standardu ISTQB (International Software Testing Qualifications Board), což je mezinárodní organizace pro certifikování softwarových testerů, je testování vždy závislé na kontextu. To znamená, že mobilní aplikace elektronického bankovnictví, která je vyvíjena v agilním prostředí, bude využívat jiné metodiky testování než například řídicí software pro letadla. Ačkoliv jsou zmíněné příklady dva zcela odlišné produkty, cíle jejich testování by měly vycházet ze skupiny cílů, které by se měl snažit dodržovat každý testovací tým, nehledě na kontextu produktu. [1], [4]

Do této skupiny cílů patří:

- Validovat, zda byly splněny všechny předem specifikované požadavky na produkt.
- Odhalit defekty a díky tomu snížit úroveň rizika nízké kvality softwaru.
- Předcházet vzniku nových defektů díky ohodnocení pracovních produktů, jako jsou produktové požadavky, uživatelské scénáře, návrh a kód.
- Ověřit, že je testovaný produkt kompletní a funguje tak, jak koncoví uživatelé očekávají.
- Vytvořit důvěru v danou úroveň kvality testovaného produktu.
- Poskytnout všem zúčastněným stranám dostatečné množství informací, aby mohly na jejich základě činit kvalifikovaná rozhodnutí, zejména pokud jde o úroveň kvality testovaného produktu.
- Dodržet smluvní, právní nebo regulatorní požadavky a normy a validovat, že je s nimi testovaný produkt ve shodě. [1]

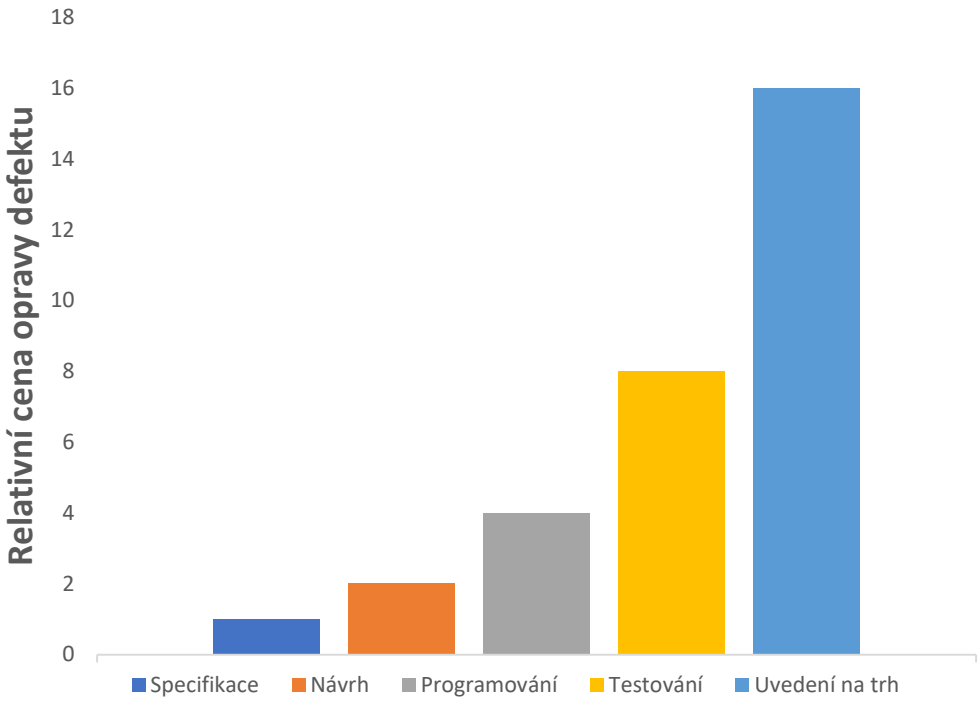
3.1.3 Softwarový defekt

Ačkoliv je představa softwaru, který je hned vyvinut bez jakýchkoliv defektů velmi lákavá, v drtivé většině případů je stále nereálná. Softwarové defekty jsou nevyhnutelnou součástí procesu vývoje softwaru a vyskytují se ve všech jeho fázích, od úplného počátku až po nasazení a využívání softwaru v produkčním prostředí. Jejich výskyt může mít velmi negativní dopad na funkčnost, výkon a celkovou kvalitu daného produktu. Při testování lze narazit na defekty různých závažností, od malých a skoro nevýznamných defektů až po kritické defekty, které způsobují selhání systému. Aby bylo možné určit, co se dá považovat za defekt, a co naopak považovat za defekt nelze, je zapotřebí znát specifikaci testovaného produktu. [2]

O defektu pak lze hovořit pokud:

- Testovaný software nevykonává něco, co by podle produktové specifikace vykonávat měl.
- Testovaný software vykonává něco, co by podle produktové specifikace vykonávat neměl.
- Testovaný software vykonává něco, o čem se produktová specifikace nezmiňuje.
- Testovaný software vykonává něco, o čem se produktová specifikace nezmiňuje, ale měla by se o tom zmiňovat.
- Testovaný software není dobře srozumitelný, těžko se s ním pracuje, je pomalý nebo jej (podle názoru testera) nebude koncový uživatel považovat za správný. [2]

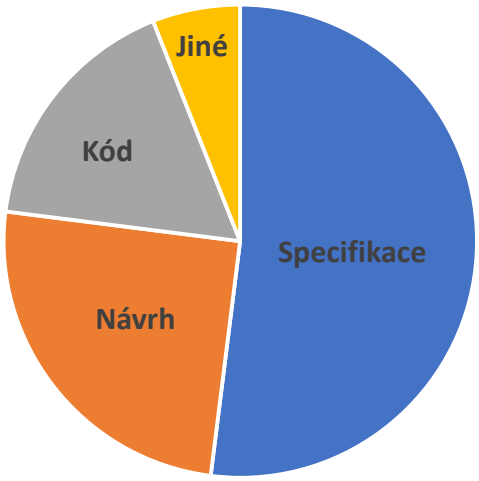
U softwarových defektů obecně platí, že náklady na jejich opravu rostou exponenciálně s fází procesu, ve které byly nalezeny. Pokud je defekt objeven již ve fázi specifikace požadavků, budou vynaložené náklady na jeho opravu mnohonásobně nižší, než pokud je objeven až v produkčním prostředí. Důležitou myšlenkou, kterou je potřeba mít na paměti je ta, že u většiny defektů jsou náklady na jejich opravu ve finále mnohem větší, než by byla pořádná investice do jejich předcházení. Příkladem může být zbytečně velký tlak na začátek samotného programování, když ještě není důkladně prodiskutována specifikace a návrh produktu. Na grafu níže lze vidět relativní cenu opravy defektu v závislosti na fázi procesu, ve které byl nalezen. [5]



Graf 1 - Náklady na opravu defektu [2]

3.1.3.1 Proč defekty vznikají

Příčin vzniku defektů je celá řada. Člověk je nedokonalé a chybující stvoření a lidská složka hraje při vzniku nových defektů velmi významnou roli. Mnoho lidí se může mylně domnívat, že nejčastější příčinou vzniku defektů je pochybení programátorů při psaní kódu, avšak ten hlavní problém se vyskytuje již daleko dříve, a to u specifikace produktu. Jak je ukázáno na grafu níže, tak více než polovina defektů vzniká z důvodu specifikace, druhým nejčastějším problémem je návrh, až poté jsou defekty zapříčiněné programovým kódem a zbylou část tvoří ostatní problémy. [2]



Graf 2 - Příčiny vzniku defektů [2]

To že je produktová specifikace nejčastějším důvodem vzniku defektů má hned několik odůvodnění. V některých případech ji jednoduše nikdo nenapíše, v jiných může být moc obecná, nebo se neustále mění a tím zapříčiňuje nejasnosti. V případě návrhu se pak většinou jedná o velice podobné problémy. Návrh se často uspěchá a dostatečně se nezkontroluje. To může vést k tomu, že některé důležité vlastnosti a charakteristiky mohou v návrhu zcela chybět. Další příčinou vzniku defektů je programový kód. Programátoři mohou být například v časové tísni a z tohoto důvodu být více náchylní k dělání chyb, systém může být moc komplikovaný na naprogramování, nebo jednoduše programátoři nemají potřebné znalosti a dovednosti pro jeho realizaci podle návrhu. Do poslední kategorie příčin vzniku defektů pak patří všechny ostatní případy. [2]

Důležitým faktem, který je potřeba si zde zmínit je ten, že velké množství defektů vzniklých v určité fázi může být následkem defektů, které byly do procesu zaneseny již v předchozích fázích. Jednoduchým příkladem je defekt v programovém kódu, který byl zanesen kvůli špatnému návrhu nebo nedostatečné specifikaci. [2]

3.1.3.2 Správná terminologie

V praxi se lze setkat s různou terminologií pro nesprávně pracující software a často jsou jednotlivé pojmy používány špatně. Pokud chce být člověk přesný a předcházet tak zbytečným nedorozuměním, je potřeba v určitých situacích používat ty správné výrazy. Existují tři hlavní pojmy, které by se měly rozlišovat, jelikož každý z nich popisuje jiný problém. Těmito pojmy jsou chyba, defekt a selhání. [4]

Člověk může udělat nějakou chybu a zapříčinit tak vznik defektu. Pokud je potřeba defekt opravit, je zapotřebí napravit chybu. Spuštění defektu v kódu pak může mít v některých případech za následek selhání, které se obvykle projevuje zhavarováním celého systému, nebo alespoň některé z jeho částí. [4]

3.1.4 Testovací dokumentace

Podobně jako je tomu i v jiných procesech, tak i v případě testování je dokumentace jeden z jeho základních pilířů, na kterém celý proces stojí. Její správná tvorba a následné využívání by mělo být zcela běžnou aktivitou v každém testovacím týmu. Jedná se o sadu dokumentů, které společně tvoří podrobný přehled o celém procesu testování na daném produktu. Detailně popisuje stanovené cíle testování, využívanou strategii testování, technické požadavky, které jsou pro provedení testů nutné, konkrétní činnosti, které testovací tým v procesu testování vykonává, výsledky testů a mnoho dalšího. Testovací dokumentace slouží jako cenná reference nejen pro testovací tým, ale také pro ostatní zainteresované strany, jako je například vývojový tým, projektový manažer nebo zákazník. Hraje také zásadní roli při udržování integrity a sledovatelnosti procesu testování. Každý ze sady dokumentů slouží k vymezení jiné části procesu testování. Mezi hlavní typy testovací dokumentace pak lze zařadit testovací plán, testovací případ a testovací scénář. [6]

3.1.4.1 Testovací plán

Testovací plán je hlavním dokumentem celého procesu testování. Lze ho chápat jako takového průvodce testováním na daném projektu, jelikož v něm jsou stanoveny veškeré podmínky, za kterých se bude celý proces probíhat. Detailně popisuje cíle testování, strategii testování, možná rizika, prioritizaci, přehled plánovaných testů, potřebné technické zdroje a potřebné lidské zdroje. Je v něm přesně stanoveno, kdo a jakým způsobem se na testování bude podílet. Vzhledem k jeho účelu a důležitosti je tvořen ještě před samotným počátkem testování a je potřeba, aby ho sestavoval opravdu zkušený člověk, jelikož se jedná o prerekvizitu pro hladký průběh procesu testování. Jeho počáteční podoba není většinou ta finální a plán je v průběhu testování aktualizován. [6]

3.1.4.2 Testovací případ

Testovací případ je typ dokumentace, který popisuje, jak se má určitá funkcionalita testovaného objektu ověřovat. Slouží jako výchozí bod pro provedení testů, jelikož obsahuje podrobný popis kroků, které je potřeba pro správnou kontrolu provést. Obvykle se skládá z prerekvizit pro provedení testu, druhu a formátu vstupních dat, podrobných navigačních kroků, očekávaných výsledků a reálných výsledků. Pokud reálný výsledek neodpovídá tomu očekávanému, jedná se obvykle o nalezený defekt. [6]

3.1.4.3 Testovací scénář

Testovací scénář je soubor několika testovacích případů, které jsou uspořádány do logicky navazujících kroků, a jejichž účelem je otestovat konkrétní funkčnost jako celek. Scénář by měl obsahovat popis oblasti k otestování, jednotlivé testovací případy seřazené v navazujících krocích a metriky hodnocení, podle kterých je možné určit, zda bylo testování úspěšné. Jejich primárním cílem je zlepšit orientaci v testovacích případech a vytvořit tak přehlednější souhrn oblastí, které jsou pokryty testy Ačkoliv se na jejich tvorbu neklade u menších projektů takový důraz, větší projekty se bez nich neobejdou. [7]

3.1.5 Testovací tým

Dobře fungující testovací tým je dalším z pomyslných pilířů pro úspěšné testování. Jeho velikost se liší v závislosti na potřebách daného projektu. Zatímco u menších týmů se jednotlivé role při testování velmi prolínají, u větších týmů jsou role často přímo rozdělené. Tak jako je tomu i u týmů v jiných oblastech, i v případě testování je zapotřebí, aby byli na jednotlivých pozicích ti správní lidé, kteří budou své pracovní povinnosti vykonávat profesionálně. Je také potřeba, aby měl celý tým jednotný cíl, kterým je dodávat kvalitně otestovaný software bez chyb. Základním rozdělením pozic v testovacím týmu je manažer testování, analytik testování a tester. [8]

3.1.5.1 Manažer testování

Manažer testování je role, která je zodpovědná za řízení celého procesu testování na daném projektu. Před zahájením testování musí posbírat veškeré informace od analytiků a naplánovat testování tak, aby bylo co nejefektivnější. Součástí jeho práce je i neustálá komunikace s projektovým manažerem, případně vedoucím vývoje a vedoucím analýzy. Po naplánování testování sleduje, jak celý proces probíhá, průběžně ho vyhodnocuje a informuje o jeho výsledcích projektového manažera a ostatní zainteresované strany. Manažer testování je odpovědný za průběžnou aktualizaci testovacího plánu na základě nových poznatků nebo změn. Mezi hlavní schopnosti, vlastnosti a dovednosti, které by měl manažer testování mít, patří dobrá znalost procesu vývoje softwaru, perfektní znalost testovacích metodik, schopnost plánovat a řídit, schopnost dobře komunikovat a umění motivovat ostatní členy testovacího týmu. [8], [9]

3.1.5.2 Analytik testování

Při plánování pomáhá analytik testování manažerovi testování například se specifikací typů testů, metodik nebo využívaných nástrojů. Ve chvíli, kdy je plánování dokončeno, zabývá se především přípravou na testování. Vychází ze všech dostupných podkladů, které si musí důkladně prostudovat. Po nastudování veškerých dostupných zdrojů navazuje návrhem testovacích případů a přípravou testovacích scénářů. Cílem analytika testování je co nejvíce pochopit, jak má software správně pracovat a jeho funkčnosti co nejkvalitněji pokrýt testovacími případy. Po dokončení scénářů předává vytvořenou dokumentaci testerům, jako podklad pro testování. S testery je neustále v kontaktu a reaguje na jejich případné dotazy. Neustále komunikuje s manažerem testování a informuje ho

o procesu testovací analýzy. Mezi hlavní schopnosti, vlastnosti a dovednosti, které by měl analytik testování mít, patří analytické myšlení, znalost obvyklých defektů, znalost metodik testování, pečlivost a cit pro detail. [8], [9]

3.1.5.3 Tester

Tester se dostává do procesu až ve fázi implementace testů. Před samotným testováním pomáhá připravit testovací data a kontroluje, zda je prostředí připravené na testy. Následně je zodpovědný za samotné provedení testovacích případů a zaznamenání jejich výsledků. Pokud při testování narazí na nějaký defekt, má za úkol ho nahlásit a po opravě testování opakovat pro kontrolu, že se v softwaru již defekt nevyskytuje, a že oprava nezavlekla do systému jiný defekt. V průběhu celého procesu informuje manažera testování o průběhu a výsledcích testů. Celý tento proces může vykonávat manuálně, nebo s využitím automatizačních nástrojů. Mezi hlavní schopnosti, vlastnosti a dovednosti, které by měl tester mít, patří schopnost rozpoznat problémy a následně je řešit, pečlivost, vytrvalost, zvídavost, dobrá znalost testovaného softwaru a v případě testerů zaměřených na automatizaci ještě ovládat skriptovací dovednosti. [8], [9]

3.1.6 Životní cyklus testování softwaru

Jak již bylo zmíněno v úvodní kapitole o testování softwaru, testování není jednotvárná činnost, při které jde pouze o průchody aplikací podle testovacích scénářů. Jedná se o přesně definovaný sled činností, které musí v rámci celého procesu testovací tým provádět. Tyto činnosti jsou jako celek nazývány jako životní cyklus testování softwaru. Mezi hlavní fáze tohoto cyklu patří analýza požadavků na testování, plánování testování, tvorba testovacích případů a scénářů, příprava testovacího prostředí, provedení testů a závěrečné ukončení a vyhodnocení testování. Každá z těchto fází je přesně strukturovaná a má vydefinované vstupy, se kterými začíná a výstupy, se kterými končí. [10], [11]

První v pořadí zmíněných činností je analýza požadavků. V této fázi studuje manažer testování společně s analytiky testování požadavky na budoucí ověřování produktu. Pokud při analýze narazí na nějakou nejasnost nebo defekt, okamžitě to komunikují s ostatními zainteresovanými stranami. Jejich hlavním úkolem je tedy v této fázi perfektně porozumět zadaným požadavkům na testování. Na vstupu jsou veškeré dokumenty týkající se požadavků na testy a na výstupu pak vytvořená určitá matice sledovatelnosti požadavků. [10], [11]

Na analýzu požadavků přímo navazuje další aktivita, kterou je plánování testování. V této fázi je hlavním představitelem manažer testování, který s případnou výpomocí analytiků testování tvoří testovací plán. Tato činnost je velmi důležitá a je potřeba na ní klást dostatečný důraz, jelikož špatně navržená strategie testování může celý proces značně narušit. Na vstupu je matice sledovatelnost požadavků z předchozí fáze a na výstupu plán testování. [10], [11]

Další fází v pořadí je tvorba podrobných testovacích případů a scénářů. Této činnosti se věnují převážně analytici testování. Při tvorbě testovacích případů zaznamenávají požadavky na testovací data, s jejichž přípravou případně pomáhají testeři. Na vstupu této fáze je schválený plán testování a na výstupu pak připravené testovací případy a scénáře. [10], [11]

Příprava testovacího prostředí je velmi důležitá aktivita, jelikož rozhoduje o podmínkách, ve kterých se bude testovaný objekt validovat. Tato fáze je speciální v tom, že je časově nezávislá na těch předešlých a často probíhá současně s tvorbou testovacích případů a scénářů. Prostředí většinou vytváří vývojový tým, avšak je nezbytné, aby před samotným testováním testeři ověřili, že je prostředí funkční a vytváří tak vhodné podmínky pro nadcházející testování. Na vstupu je systémový návrh a architektura systému, na výstupu pak připravené prostředí pro testování. [10], [11]

Ve chvíli, kdy je vše připravené, začíná fáze samotného provedení testů. Při této činnosti testeři provádějí testy podle příslušných testovacích případů. Pokud zde naleznou nějaký defekt, vrací ho vývojovému týmu k opravě. Po opravě testeři ověřují opakováním testu, že se již defekt v systému nevyskytuje. Tento proces se může opakovat i několikrát. Na vstupu do této fáze jsou veškeré výstupy z předešlých fází a na výstupu pak jsou zdokumentované výsledky provedených testů. [10], [11]

Závěrečnou fází životního cyklu testování softwaru je ukončení testování. Hlavním cílem této činnosti je zajistit, aby byly všechny činnosti související s testováním dokončeny a testovaný software byl připraven k předání. Dále zde testovací tým společně vyhodnocuje celý proces a získané zkušenosti, na jejichž základě může být vylepšen budoucí proces testování. Na vstupu této fáze je tedy dokončené testování a na výstupu zpětná vazba, předání znalostí a zkušeností a zpráva o dokončeném testování. [10], [11]

3.1.7 Rozdělení metod testování

V oblasti softwarového testování existuje celá řada různých metod, jak lze k testování přistupovat. Každá metoda má své klady a zápory a jejich využití je závislé na kontextu ověřovaného objektu. Pro optimální výsledky je vhodné mezi sebou různé metody kombinovat.

Základní rozdělení metod testování je:

- Podle nutnosti spuštění testovaného softwaru na statické a dynamické testování.
- Podle znalosti zdrojového kódu testovaného softwaru na testování černé, bílé a šedé skřínky.
- Podle toho, zda jsou testovány funkcionality systému na funkční a nefunkční testování.
- Podle rozsahu a účelu testování na smoke a regresní testování.

3.1.7.1 Statické a dynamické testování

Statické testování je metoda, při které se ověřovaný objekt pro jeho validaci nespouští. Ve většině případů se jedná o ověření správnosti a kompletnosti veškeré dokumentace jako je produktová specifikace, návrh softwaru, testovací plán nebo testovací scénáře. Do statického testování také patří statická revize zdrojového kódu. Ta je prováděna ve většině případů strojem nebo samotným programátorem. Hlavní využití má statické testování především v raných fázích vývojového cyklu, jelikož pomáhá identifikovat defekty v dokumentaci ještě před počátkem programování, což znamená nižší náklady na opravu. Využívání této metody však má své místo během celého procesu testování. [12]

Dynamické testování je tedy opakem statického testování. Při této metodě je zapotřebí, aby byl testovaný objekt spuštěn a ověřování pak probíhá za jeho běhu. To znamená, že je potřeba, aby již existoval nějaký spustitelný a testovatelný prototyp produktu. Dynamické testování tedy vstupuje do procesu později a jeho hlavním cílem je ověření konkrétních funkčních a nefunkčních aspektů podle testovacích scénářů. [12]

3.1.7.2 Testování černé, bílé a šedé skříňky

Při metodě testování černé skříňky nemají testeři přístup ke zdrojovému kódu testovaného produktu. Z toho plyne i její název, který má evokovat fakt, že objekt je pro testery jakási neprůhledná skříňka, do které nevidí. Při využívání této metody mají testeři díky dokumentaci informace o tom, jak se má produkt chovat, ale již je nezajímá, proč se tak chová. Testování touto metodou je nejvíce podobné chování z pohledu koncového uživatele. [13]

Testování bílé skříňky je opakem metody testování černé skříňky, tudíž při jejím využívání testeři mají přístup ke zdrojovému kódu testovaného softwaru. Někdy se jí také říká metoda průhledné skříňky, což má v tomto případě evokovat fakt, že je testerům známo, co a proč se děje uvnitř. Při testování touto metodou je možné na základě kódu odhadnout, jak bude produkt reagovat na určité vstupy. To může znatelně usnadnit odhad rizika, zda se v softwaru nachází defekt. Tato metoda umožňuje testovat celý systém do hloubky, avšak pro její provedení je již nezbytná znalost programování. [13]

Poslední metodou testování v této kategorii je metoda šedé skříňky, která kombinuje obě předchozí metody. U šedé skříňky testeři obvykle ověřují produkt z pohledů černé skříňky, avšak pro doplnění testů nahlíží i do zdrojového kódu. [13]

3.1.7.3 Funkční a nefunkční testování

Pokud se hovoří o funkčním testování, znamená to, že se jedná o ověřování funkcionalit testovaného softwaru. Tato metoda se zaměřuje především na kontrolu z pohledu skutečného využívání softwaru koncovým zákazníkem. Veškeré funkcionality se ověřují zadáním vybraného vstupu a porovnáním reálného výstupu s tím očekávaným. [14]

Nefunkční testování je pak opakem toho funkčního. Při této metodě se ověřují kvalitativní atributy testovaného produktu. Mezi tyto atributy patří například testy výkonnosti, bezpečnosti, použitelnosti nebo spolehlivosti. Ačkoliv nefunkční testování neověřuje přímo nějaké funkcionality softwaru, jeho provedení je stejně podstatné, jako je funkční testování. [14]

3.1.7.4 Smoke a regresní testování

Smoke testování je metoda, při které se provádí sada testů, která má za úkol ověřit, zda jsou základní a kritické funkce testovaného objektu v dostatečně funkčním stavu. Jeho cílem není otestovat objekt do hloubky, ale pouze identifikovat případné překážky, které by mohly bránit dalšímu testování. Většinou se smoke testování provádí přímo po nasazení verze do testovacího prostředí a před provedením rozsáhlejšího testování. Pokud smoke testy odhalí nějaký problém, je potřeba zvážit, zda má význam v testování na nasazené verzi vůbec pokračovat, nebo je vhodnější počkat na opravu. [15]

Regresní testování slouží k opětovnému ověření, zda nově přidané změny, ať už přidání nových funkcionalit, nebo úprava těch stávajících, negativně neovlivňují dosavadní systém. Regresní testy jsou velmi rozsáhlé a podrobně testují celý systém. Je žádoucí, aby se prováděly po každé implementované změně, včetně těch zdánlivě nevýznamných, jelikož i ty mohly v testovaném objektu zapříčinit nějaký problém. [15]

3.1.8 Úrovně funkčního testování

Jak již bylo zmíněno v předešlé podkapitole, funkční testování ověřuje funkcionality testovaného softwaru. Za tímto účelem je testování prováděno na různých úrovních, z nichž každá má jinak velký rozsah a každá se zaměřuje na jiné aspekty. Mezi hlavní úrovně testování patří jednotkové testování, integrační testování, systémové testování a akceptační testování.

3.1.8.1 Jednotkové testování

Jednotkové testování je první úrovní funkčního testování softwaru. Jak je již z názvu patrné, na této úrovni se testování zabývá ověřováním funkčnosti jednotek. Není přesně definováno, co všechno se pod pojmem jednotka skrývá, avšak ve většině případů se jednotkou rozumí nějaká samostatně testovatelná část zdrojového kódu. Jednotkové testování je hlavním představitelem testování metodou bílé skříňky, jelikož je celé ověřování založeno na práci s kódem. Mezi nejčastější jednotky patří například jednotlivé komponenty, třídy, funkce, procedury nebo datové struktury. Ve většině případů je prováděno vývojáři, kteří danou jednotku naprogramovali, jelikož jsou s jejím chováním nejvíce obeznámeni a pokud nejsou testy v pořádku, mohou si defekt hned opravit. Po úspěšném dotestování vývojářů na této úrovni se již software předává k ověřování funkčnosti testerům. [16]

3.1.8.2 Integrační testování

Další úrovní, která přímo navazuje na jednotkové testování, je úroveň integračního testování. Tato úroveň je již v rukách testerů, kteří zde ověřují, jak fungují naprogramované komponenty a jednotky společně. Existují čtyři různé přístupy, jak toto testování lze provádět, kterými jsou přístup Velkého třesku, přístup shora-dolů, přístup zdola-nahoru a posledním je hybridní přístup. [17]

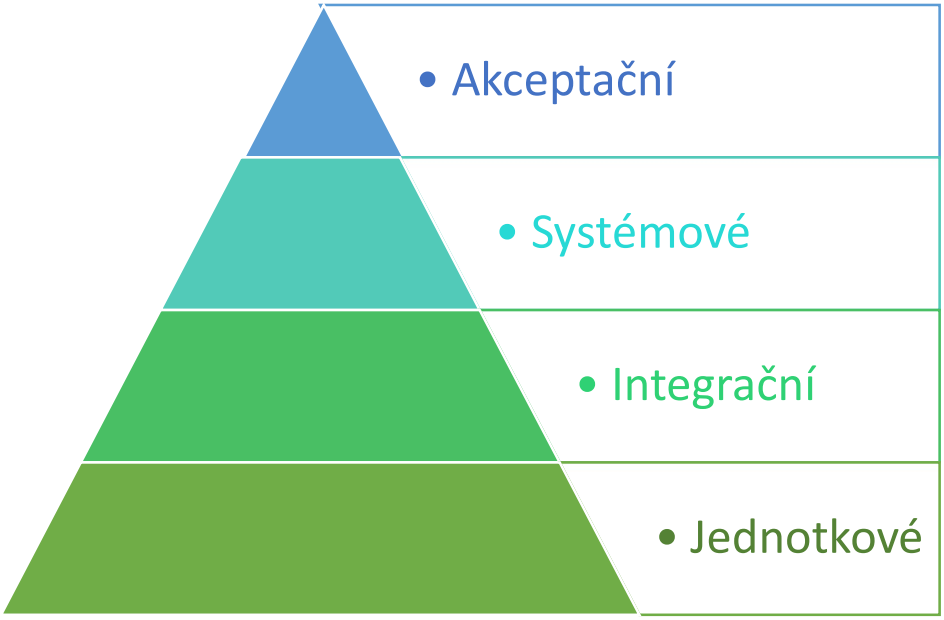
V případě přístupu k testování formou Velkého třesku se všechny jednotky zkombinují do jednoho celku a testují se všechny společně. Pokud se zvolí přístup shora-dolů, testují se nejprve jednotky na nejvyšší úrovni a postupně se prostupuje k těm na nižších úrovních. Přístup zdola-nahoru je přesným opakem toho předešlého, tudíž se začíná dole a postupně se stoupá. Posledním přístupem je pak hybridní, kterému se někdy také říká sendvič, jelikož kombinuje přístupy shora-dolů a zdola-nahoru. [17]

3.1.8.3 Systémové testování

V pořadí třetí úrovní funkčního testování softwaru je systémové testování, které taktéž provádějí testeři. Na této úrovni se již ověřuje testovaný software jako celek a validuje se, že je ve shodě se specifikovanými požadavky. Většinou se zde již využívá pouze testování metodou černé skříňky a testeři se snaží co nejvíce přiblížit potřebám koncových uživatelů. Jedná se o poslední úroveň před tím, než se předá výsledný produkt ke kontrole zákazníkovi, a tudíž je extrémně důležitá. [16]

3.1.8.4 Akceptační testování

Poslední úrovní funkčního testování je akceptační testování. Na této úrovni si již často produkt ověřuje sám zákazník s možnou výpomocí testerů. Hlavním cílem akceptačního testování je ověřit, zda produkt splňuje veškerá akceptační kritéria, což jsou předem smluvně stanovené požadavky na produkt od zákazníka. Pokud akceptační testy nejsou v pořádku, vrací se produkt zpět do vývojového procesu k úpravě. Ve chvíli, kdy jsou testy v pořádku, předává se hotový produkt zákazníkovi. [16]



Obrázek 1 - Úrovně funkčního testování [18]

3.2 Automatizované testování

Automatizace v testování softwaru se zavádí ze stejného důvodu, jako tomu je i v ostatních odvětvích. Hlavní motivací, proč se společnosti pouštějí do automatizace je časová a finanční úspora. Aby bylo možné tuto problematiku správně pochopit, je potřeba si vysvětlit, co to automatizované testování vůbec je, jaké jsou jeho výhody, kdy má smysl automatizaci zavést a co by se automatizovat určitě nemělo.

3.2.1 Manuální vs automatizované testování

Manuální a automatizované testování jsou dvě zcela odlišné metodiky, jak lze k provádění testů přistupovat. V obou případech se jedná o ověřování funkcionalit podle testovacích případů, avšak hlavní rozdíl je v tom, jak je validace prováděna.

V případě manuálního testování jsou testy prováděny ručně, nejčastěji testery a bez využití automatizačních nástrojů. Naopak při automatizovaném testování testeři pouze vytvářejí testy pomocí některého z automatizačních nástrojů, avšak vykonávání těchto testů už provádí nástroj. V tomto případě je potřeba, aby měl tester alespoň základní znalost programování, jelikož v automatizovaném testování je ve většině případů potřeba řídit testy pomocí vytvořených skriptů. [19]

Není možné jednoznačně určit, zda je lepší manuální nebo automatizovaný přístup k testování, jelikož obě tyto metody mají své klady a zápory. V ideálním světě by byly všechny testovací scénáře automatizované, což je alespoň v dnešní době stále nereálné. Ne vše se automatizovat vyplatí, a ne vše je automatizovat vhodné. Existují chvíle, kdy stroj člověka stále nemůže v testování plnohodnotně nahradit, jako jsou například průzkumové testy nebo testy použitelnosti. Pro optimální výsledky je potřeba najít vhodnou kombinaci těchto metod a správně je využívat.

3.2.2 Výhody automatizovaného testování

Při správném nastavení způsobu automatizace testování se z této metody stává velmi mocný pomocník. Automatizace nabízí mnoho výhod, které celému týmu šetří peníze a čas. Mezi ty nejzásadnější výhody, které automatizované testování proti manuálnímu nabízí, lze začadit především rychlost, efektivitu, přesnost a neúnavnost.

První výhodou, kterou automatizace poskytuje, je rychlost. Hlavními činnostmi, kde tato vlastnost hraje významnou roli, je při přípravě testovacích dat nebo při samotném provedení testů. Čím komplexnější testovaný software je, tím je zpravidla časové náročnější jeho otestování. Automatizační nástroje jsou schopné provádět testy mnohonásobně rychleji než člověk, což se na větších testovacích sadách velmi rychle projeví. Je však vhodné zde zmínit, že automatizované testy jsou časově mnohem náročnější na přípravu, avšak s každým jejich spuštěním se tento rozdíl velmi rychle snižuje. [2], [20]

Další výhodou, kterou automatizace nabízí, je efektivita. Při manuálním provádění testů jsou testeři limitováni pouze na tuto činnost. Pokud jsou však testy prováděny automatizovaně, testerům se uvolní ruce a mohou se při jejich provádění věnovat jiným důležitým činnostem. [2], [20]

Automatizované testování je také mnohem přesnější, než je to manuální. Nástroj otestuje a nahlásí přesně to, co je definováno. Při provádění automatizovaných testů nehrají lidské faktory žádnou roli. Pokud testeři například musí ověřit jednu funkcionalitu s desítkami různých dat, je velmi pravděpodobné, že se tato práce stane rutinní a nezajímavá a jejich kvalita ověřování bude postupně snižovat. [2], [20]

Poslední ze zmíněné skupiny výhod je neúnavnost, která je velmi úzce spojena s přesností. Opět se jedná o nějaký nežádoucí lidský faktor, kterým automatizační nástroj nedisponuje. Nástroje neznají únavu a testy budou vykonávat bez problému pořád dokola. Testy tedy proběhnou vždy ve stejné kvalitě a nezáleží na tom kdy nebo kolikrát seběhnou. [2], [20]

3.2.3 Vhodné testy pro automatizaci

Většina testů, které člověka napadnou, je možné automatizovat a významně z toho profitovat. Existují však určité případy, které nejen že z automatizace profitují, ale dokonce jejich manuální provedení ani nedává smysl.

Hlavním představitelem testů, které jsou vhodné pro automatizaci, jsou takové testy, které se často opakují. Do této skupiny patří především jednotkové a regresní testy, které jsou často velmi rozsáhlé a měly by proběhnout po každé změně v testovaném objektu. Pokud by testeři prováděli celé regresní sady testů manuálně, zbytečně by se obírali o čas, který by mohli věnovat například důkladnější analýze. [16], [20]

Další ze skupiny testů, které jsou vhodné pro automatizaci, jsou takové, které je provést manuálně velmi náročné, nebo dokonce nemožné. Zcela jasným příkladem jsou v tomto případě některé druhy nefunkčních testů, jako jsou zátěžové testy. Pokud je potřeba ověřit chování testovaného objektu při desítkách až tisících uživatelů v jeden moment, je prakticky nereálné tyto testy provádět manuálně. Existují automatizační nástroje, které jsou na zátěžové testování přímo určené a simulace velkého množství uživatelů pro ně není žádný problém. [16], [20]

Jelikož zde byly zmíněny testy, u kterých je manuální provedení velmi obtížné, je vhodné do této skupiny zařadit i takové testy, které je naopak velmi snadné automatizovat a zároveň jejich provedení není pouze jednorázová činnost. [20]

Posledním příkladem testů vhodných pro automatizaci jsou testy řízené daty a s nimi velmi úzce spojená automatizovaná příprava dat. Takové testování se využívá především na úrovni integračního testování, kde se velmi často ověřuje například chování API (Application Programming Interface) při různých datových vstupech. Manuální příprava takových dat a následné provedení testů je poměrně jednotvárná činnost, při které by mohla být opět v ohrožení kvalita testování. Je zde však velmi důležité myslet na to, že by se testovací data měla také uklízet, nejlépe automatizovaně po dokončení testů. [16], [20]

3.2.4 Nevhodné testy pro automatizaci

Po vymezení skupin testů, které by se měly automatizovat, je na místě popsat i takové testy, které by se naopak automatizovat neměly. Některé skupiny vyplívají z předchozí části, jelikož jsou jejich přímým opakem.

Jelikož je jednou z hlavních motivací automatizovaného testování úspora času, tak první skupinou testů, které zde budou zmíněny, jsou takové testy, které je potřeba provést pouze jednou. Případů, kdy je vytvoření automatizovaného testu rychlejší než jeho manuální provedení, bude velmi málo. Avšak i takové případy se najdou a v takovém případě stojí za zvážení, zda je automatizace vhodná. Automatizované testy profitují především z jejich opakovaného spouštění, a tudíž pokud jsou provedeny pouze jednou, nemají prostor pro návratnost počáteční investice. [20]

Mezi další testy, které nejsou vhodné pro automatizaci, patří takové, které jsou přímo závislé na lidském vnímání a úsudku. Hlavními představiteli této skupiny jsou některé druhy nefunkčních testů, jako jsou například testy použitelnosti a průzkumové testy. Jak již z názvu vyplívá, tak u testů použitelnosti je potřeba, aby někdo testovaný software opravdu používal. Pozorování uživatelů, informování se o jejich zkušenostech s využíváním aplikace a následné vyhodnocování výsledků je činnost, kterou optimálně vykoná pouze člověk. Podobně tomu je i v případě průzkumového testování, jehož hlavním cílem je naučení se více o testovaném produktu jeho prozkoumáváním a následně získané informace využívat pro případná zlepšení. [20]

Poslední skupinou testů, které není vhodné automatizovat jsou takové, jejichž implementace se neustále mění. Typickým příkladem jsou testy uživatelského rozhraní, kde určitě není vhodné automatizaci zavádět společně s prvním funkčním prototypem aplikace. Uživatelské rozhraní se při jeho vývoji velmi často mění, než se dostane do své finální podoby a udržování takových testů může být časově velmi náročné. [20]

3.3 Analýza využívaných nástrojů

Nástrojů pro automatizované testování je k dispozici celá řada. Pokud je pro dané potřeby týmu vybrán vhodný nástroj, může to znamenat obrovskou úsporu času, který je pak možný využít pro detailnější testovací analýzu a další důležité činnosti. Nástroje pro testování softwaru můžeme rozdělit do několika skupin podle jejich cílového využití. Mezi tyto skupiny patří nástroje vhodné pro jednotkové testování, nástroje pro zátěžové testování, nástroje využívané pro integrační a API testování nebo nástroje vhodné pro testování grafického uživatelského rozhraní. Některé nástroje však lze využívat pro pokrytí více oblastí testování, například uživatelského rozhraní i API. Tato práce je zaměřená na automatizované testování uživatelského rozhraní webové aplikace, a proto zde budou vybrány jedny z nejvyužívanějších nástrojů, zaměřených především na tuto oblast.

3.3.1 Selenium

Selenium je open-source nástroj, který je široce využívaný pro automatizované testování webových aplikací. Vznikl v roce 2004 a jeho tvůrcem je Jason Huggins. Skládá se ze tří hlavních komponent, kterými jsou Selenium WebDriver, Selenium Grid a Selenium IDE. [21]

Selenium WebDriver (někdy označován Selenium 2.0) je nejpoužívanější komponentou nástroje Selenium, která je zodpovědná za interakci s webovým prohlížečem pomocí spouštění testovacích skriptů. Celý proces funguje tak, že se vytvořené skripty posílají jako jednotlivé příkazy do ovladače vybraného prohlížeče. Ovladač pak následně tyto příkazy provede ve své instanci prohlížeče. Příkladem takového ovladače je GeckoDriver pro Mozillu Firefox nebo ChromeDriver pro Google Chrome. [21]

Selenium Grid je komponenta, která umožňuje paralelní provádění testovacích skriptů na více počítačích a v různých prohlížečích díky principu hub-node. Hub je server, který přijímá příkazy z WebDriveru. Jednotlivé příkazy zkonvertuje do JSON (JavaScript Object Notation) formátu a následně odešle požadavek na vzdálené ovladače zaregistrovaných nodů. Node je tedy vzdálené zařízení, které provádí příkazy na své instanci. [21]

Selenium IDE je integrované vývojové prostředí, které je implementováno jako rozšíření do webových prohlížečů Google Chrome a Mozilla Firefox. Na rozdíl od WebDriveru v něm pro tvorbu testů není potřeba znalost žádného programovacího jazyka. Umožňuje zaznamenávat interakce v prohlížeči, které pak přehrává. [21]

Charakteristika	Popis
Typ licence	Open-source
Cena	Zdarma
Podporované platformy	Windows, MacOS, Linux
Nutná znalost programování	Ne v případě Selenium IDE, jinak ano.
Podporované jazyky	Java, Python, C#, JavaScript, Ruby, Kotlin, PHP, Perl
Podporované prohlížeče	Google Chrome, Mozilla Firefox, Microsoft Edge, Internet Explorer, Safari, Opera

Tabulka 1 - Charakteristiky nástroje Selenium [22]

3.3.2 Katalon Studio

Katalon Studio je robustní nástroj, který byl vytvořen v roce 2016 společností KMS Technology. Umožňuje testovat nejen webové aplikace, ale také ty mobilní a desktopové. Tento nástroj kromě testování uživatelského rozhraní také umožňuje integrační a API testování. Je navržen tak, aby jeho používání bylo intuitivní a co nejjednodušší, což ocení především týmy, které potřebují začít automatizovat bez předešlých zkušeností se skriptováním. Katalon Studio umožňuje snadné generování testů, které funguje na principu zaznamenání interakcí testera a jejich následné opakování. Při složitějších testech je možné využít také skriptování. [23]

Charakteristika	Popis
Typ licence	Proprietární
Cena	Základní verze bez jakýchkoliv rozšíření je zdarma. Možnost velkého množství placených rozšíření 600-17 000 Kč měsíčně
Podporované platformy	Windows, MacOS, Linux
Nutná znalost programování	Ne
Podporované jazyky	Java, Groovy
Podporované prohlížeče	Google Chrome, Mozilla Firefox, Microsoft Edge, Safari, Opera

Tabulka 2 - Charakteristiky nástroje Katalon Studio [22], [24]

3.3.3 Cypress

Cypress je moderní open-source framework vhodný pro testování webových aplikací. Vytvořil ho Brian Mann v roce 2014 a od té doby si získal velkou přízeň mezi testery a vývojáři. Je navržen tak, aby co nejvíce usnadnil řešení častých problémů, které se při testování moderních webových aplikací vyskytují. Příkladem může být zabudování funkce automatického čekání, což znamená, že framework automaticky počká na zobrazení prvků a dokončení síťových požadavků, než přejde k dalšímu kroku v testu, čímž předchází velmi častým problémům při automatizaci testů uživatelského rozhraní. Cypress má vlastní ovladač pro automatizaci prohlížeče, tudíž pro jeho využívání není potřeba přidávat speciální ovladače pro jednotlivé prohlížeče. Tento nástroj také umožňuje odchytávat požadavky na server, a tudíž lze při testování zahrnout i testy API. Cypress obsahuje dvě velmi důležité komponenty, kterými jsou CypressTestRunner a CypressCloud. [25], [26]

CypressTestRunner je jádrem nástroje Cypress. Spouští testy, zobrazuje jednotlivé spouštěné příkazy a následně jejich výsledky v reálném čase. Velmi užitečnou funkcí je možnost ladění v čase, která umožňuje zobrazit stav testované aplikace v libovolném okamžiku běhu testů. To znatelně usnadňuje identifikaci a opravu problémů v testech. [25], [26]

CypressCloud je cloudová služba, která umožňuje zaznamenávat průběh testů a zobrazovat jejich podrobné výsledky, včetně snímků obrazovky, videozáznamů a údajů o síťovém provozu. Díky tomu lze snadno sdílet výsledky testů s ostatními a porovnávat výsledky různých testovacích běhů. [25], [26]

Charakteristika	**Popis**
Typ licence	Open-source
Cena	Zdarma
Podporované platformy	Windows, MacOS, Linux
Nutná znalost programování	Ano
Podporované jazyky	JavaScript, TypeScript
Podporované prohlížeče	Google Chrome, Mozilla Firefox, Microsoft Edge, Safari

Tabulka 3 - Charakteristiky nástroje Cypress [27]

3.3.4 TestComplete

TestComplete je komerční nástroj pro automatizované testování vyvinutý společností SmartBear v roce 1999. Jedná se o profesionální nástroj, který mimo testování grafického uživatelského rozhraní webových aplikací umožňuje také testování desktopových a mobilních aplikací. Celý nástroj je velmi snadný na využívání a je vhodný pro rychlé vytváření automatizovaných testů. TestComplete je známý také pro svou skvěle fungující funkcionalitu identifikování objektů umělou inteligencí. Podobně jako je tomu u Katalon Studia, tak i zde se dává přednost vytváření testů pomocí zaznamenání interakcí a jejich následném přehrávání, avšak co se týče skriptování, nabízí TestComplete větší možnosti ve výběru programovacího jazyku. [28]

Charakteristika	**Popis**
Typ licence	Proprietární
Cena	Od 40 000-120 000 Kč za licenci.
Podporované platformy	Windows
Nutná znalost programování	Ne
Podporované jazyky	JavaScript, Python, VBScript, C++, C#, Delphi
Podporované prohlížeče	Google Chrome, Mozilla Firefox, Microsoft Edge, Safari, Opera

Tabulka 4 - Charakteristiky nástroje TestComplete [22], [29]

3.3.5 Závěr analýzy

Analýza poskytuje základní přehled o vybraných nástrojích pro automatizaci testování webových aplikací a na jejím základě byl vybrán nástroj, který bude využit pro realizaci praktické části této práce.

Po srovnání veškerých charakteristik vybraných nástrojů byla jako první vyřazena dvojice Katalon Studio a TestComplete. V obou případech byl jedním z hlavních faktorů jejich cena. Nástroj Katalon Studio je sice ve své základní verzi zdarma, ale pro náročnější testy není v základním balíčku zcela vhodný. Prostředí, ve kterém bude realizována praktická část práce vyžaduje poměrně více složitějších testů na provedení, a tudíž je vhodnější sáhnout po některém z nástrojů, založeném pouze na skriptování.

Následně bylo zapotřebí se rozhodnout mezi nástroji Selenium a Cypress. Výhodou obou nástrojů je ta, že jsou v Komerční bance, kde bude realizována praktická část této práce, široce podporované. Oba nástroje jsou velmi užitečné a dokáží provést veškeré náležitosti, které bude potřeba automatizovaně ověřovat. Výběr byl tedy založen především na zkušenostech autora, který zvolil nástroj Selenium, ve kterém již v minulosti testy vytvářel.

4 Vlastní práce

4.1 Testovaná aplikace

Praktická část práce byla realizována v Komerční bance na webové aplikaci Gate. Jedná se o aplikaci, která umožňuje potencionálním klientům online sjednání jimi vybraného bankovního účtu. Na výběr mají ze tří možností, kterými jsou MůjÚčet, MůjÚčet Plus a MůjÚčet Gold, které se liší v měsíční ceně za provozování a v nabízených benefitech.

Tým, který má Gate ve své správě se zároveň věnuje vývoji nové aplikace, která bude umožňovat stát se klientem banky přes aplikaci v mobilním zařízení. Tím vzniká potřeba, aby se testovací tým soustředil na testování funkcionalit nového softwaru, ale zároveň musí být zajištěna kontrola, že i Gate nadále běží bez nějakých problémů. Vznikl tak požadavek na automatizaci testů klíčových funkcionalit, které týmu uvolní ruce, aby se mohl věnovat novému projektu. Po dohodě testerů se zvolila strategie automatizovat nejdříve všechny funkcionality „happy day" cestou, což znamená vytvořit nejdříve veškeré validní průchody aplikací a až následně se bude zvažovat, zda se budou automatizovat i různé alternativní scénáře.

První kritickou oblastí v procesu sjednání produktu je ověření totožnosti potencionálního klienta. Ověření lze provést pomocí bankovní identity z jiné banky, nebo přes focení dokladů totožnosti. Tato funkcionalita je pro všechny tři varianty účtů stejná, a proto byla možnost vytvořit automatizovaný test pro libovolný z nich. Pro účely této práce si autor zvolil vytvoření testovacího scénáře a automatizovaných testů pro MůjÚčet Plus.

4.2 Tvorba testovacího scénáře

Jak již bylo zmíněno v teoretických východiscích, testovací scénář je sada logicky navazujících testovacích případů sloužící pro ověření konkrétní funkcionality. Aby tedy mohl testovací scénář vzniknout, bylo nejdříve potřeba vytvořit vhodné testovací případy, které společně vytvořily požadovaný scénář. Testovacím případům byly pro zlepšení čitelnosti v rámci této práce přiřazeny zjednodušené identifikátory ve tvaru TP (testovací případ) – 00x (unikátní číslo případu) a v případě testovacího scénáře TS (testovací scénář) – 00x (unikátní číslo scénáře).

Příkladem vytvořeného testovacího případu je „TP-002 Doplnění osobních údajů”, který popisuje, jak validně projít v pořadí druhou obrazovkou v testovaném procesu. Obsahuje veškeré důležité parametry, které by měl podle literární rešerše v teoretické části práce zahrnovat. Sloupec popisující reálný výsledek daného kroku zůstává prázdný a testeři ho vyplňují až po provedení testu. Zbylé testovací případy si lze prohlédnout v příloze této práce.

<table>
<tr><td>TP-002</td><td colspan="3">Doplnění osobních údajů</td></tr>
<tr><td colspan="4">Prerekvizity</td></tr>
<tr><td colspan="4">• Přístup do testovacího prostředí Gate.</td></tr>
<tr><td colspan="4">Vstupní data</td></tr>
<tr><td colspan="4">Uživatelské osobní údaje:
• Jméno – obsahuje pouze písmena abecedy (Tomáš)
• Příjmení – obsahuje pouze písmena abecedy (Marný)
• Email – (tomas.marny@gmail.com)
• Telefonní číslo – obsahuje pouze číslice (731023021)</td></tr>
<tr><td>Krok</td><td>Popis</td><td>Očekávaný výsledek</td><td>Reálný výsledek</td></tr>
<tr><td>1.</td><td>Otevřete v prohlížeči adresu url: https://gate-dev.kb.cz/online/wizad/basic-info/muj-ucet-plus.</td><td>Zobrazí se stránka pro vyplnění základních údajů o klientovi.</td><td></td></tr>
<tr><td>2.</td><td>Doplňte osobní údaje podle kritérií vstupních dat.</td><td>Vyplněné údaje jsou zobrazeny v příslušných textových polích.</td><td></td></tr>
<tr><td>3.</td><td>Klikněte na tlačítko „POKRAČOVAT”.</td><td>Vygeneruje se pin kód, který se uloží do databáze a zobrazí se stránka pro jeho vyplnění.</td><td></td></tr>
</table>

Tabulka 5 - Testovací případ „Doplnění osobních údajů"

Po vytvoření jednotlivých testovacích případů bylo potřeba je uspořádat do logicky navazujících kroků testovacího scénáře. Po konzultaci s testovacím týmem bylo rozhodnuto, že pro lepší orientaci je vhodnější vytvořit pro obě varianty ověření totožnosti uživatele samostatný scénář. První scénář se věnuje ověření pomocí focení dokladů totožnosti, druhý je pak zaměřený na ověření pomocí bankovní identity z jiné banky.

Oba scénáře spolu sdílejí většinu testovacích případů, jelikož celý proces ověření identity se liší pouze v momentě, kdy klient zvolí cestu přes ověření bankovní identitou, nebo přes focení dokladů totožnosti. Po obou variantách následuje testovací případ, ověřující správnost načtených údajů.

V tabulce níže lze vidět vytvořený testovací scénář „TS-001 Identifikace uživatele pomocí dokladů totožnosti". I v případě tvorby testovacího scénáře autor vycházel především z poznatků z teoretické části, kde je popsáno, jaké náležitosti mají testovací scénáře ideálně obsahovat. Druhý scénář si lze prohlédnout v příloze práce.

TS-001	Identifikace uživatele pomocí dokladů totožnosti	
Oblast k otestování		
Testovací scénář je zaměřen na „happy day" ověření uživatelské cesty ověření totožnosti přes doklady totožnosti a pořízení selfie fotografie.		
Testovací případ	Očekávaný výsledek	Reálný výsledek
TP-001 Možnost založit účet u KB	Zobrazí se stránka pro doplnění osobních údajů uživatele.	
TP-002 Doplnění osobních údajů	Zobrazí se stránka pro jeho vyplnění pin kódu, který je uložený v databázi.	
TP-003 Zadání pin kódu	Zobrazí se stránka s výběrem cesty pro ověření totožnosti.	
TP-004 Volba cesty ověření identity	Zobrazí se stránka informační stránka pro ověření cestou s doklady.	

TP-005 Vložení dokladů totožnosti	Potřebné doklady jsou správně zpracovány a zobrazí se stránka pro kontrolu načtených údajů.	
TP-007 Kontrola načtených údajů	Veškeré údaje jsou shodné s údaji na dokladech totožnosti.	

Tabulka 6 - Testovací scénář „Identifikace uživatele pomocí dokladů totožnosti"

4.3 Manuální testování

Jedním z cílů této práce je porovnání časové náročnosti manuálního a automatizovaného testování. Po dokončení tvorby testovacích scénářů tedy bylo potřeba nejprve provést manuální testy. Pro tyto účely byli využiti všichni testeři z týmu, který má na starosti testování aplikace Gate a nově vznikající mobilní aplikaci. Testovací tým se skládá ze čtyř členů, z nichž jedním je autor této práce. Pro přesnější měření nebyl autor součástí měření časové náročnosti provedení testů, jelikož by výsledky měření mohly být zkreslené. Je potřeba zmínit, že jeden tester má s testováním aplikace Gate bohaté zkušenosti a dva tuto aplikaci nikdy netestovali a bylo to pro ně zcela nové prostředí, jelikož se do této chvíle zabývali pouze ověřováním vznikající mobilní aplikace.

Veškeré testování prováděli testeři samostatně v přítomnosti autora bakalářské práce. Jednotlivá měření vždy započala až ve chvíli, kdy byli testeři seznámeni s testovacími případy a měli připravená potřebná data a přístupy. Ukončena pak byla ve chvíli, kdy byl ověřen poslední testovací případ v daném scénáři. Veškerá měření prováděl autor pomocí stopek na mobilním telefonu, tudíž mohou být výsledky o maximálně o dvě vteřiny nepřesné. Tento fakt je však pro účely této práce zcela zanedbatelný.

4.4 Automatizované testování

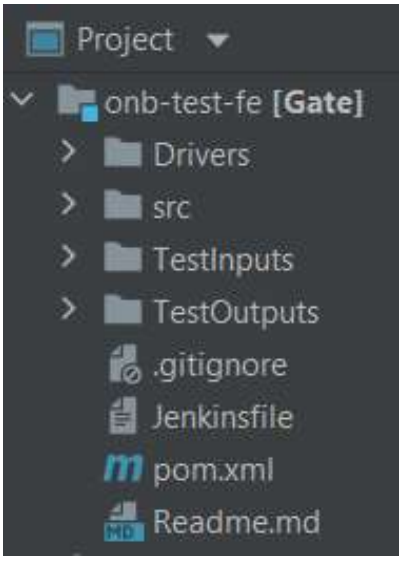
Po dokončení měření časově náročnosti manuálních testů nastal čas ověřit dané funkcionality s využitím automatizačního nástroje. Na základě analýzy nástrojů pro automatizaci testování webových aplikací v teoretické části práce byl vybrán nástroj Selenium WebDriver. Pro psaní programového kódu byl zvolen jazyk Java, se kterým má autor největší zkušenosti a jako vývojové prostředí bylo zvoleno moderní prostředí IntelliJ IDEA Ultimate.

Jak již bylo zmíněno v analýze nástrojů, Selenium je v prostředí Komerční banky široce podporovaný nástroj. Většina aplikací v bance je poměrně rozsáhlá a složitá, a tak nad tímto nástrojem již v minulosti vznikl interní framework, který má za cíl testerům znatelně usnadnit tvorbu automatizovaných testů. Tento framework testerům poskytuje například jednodušší a přehlednější práci s webovými prvky, řeší veškerou logiku pro práci s různými ovladači jednotlivých prohlížečů nebo obsahuje velké množství různých metod, které usnadňují interakce s webovými prvky při provádění testů. Dále framework řeší integrace s dalšími nástroji, jako jsou například nástroje pro zaznamenávání výsledků testů nebo nástroj pro automatické spouštění testů podle vydefinovaných parametrů.

Jelikož se jedná o poměrně rozsáhlou nadstavbu nad Seleniem, je ve zbytku této práce používán pro tento nástroj název SeleF (Selenium Framework), jelikož je tento název používán i v prostředí banky.

4.4.1 Struktura projektu

Nejprve bylo zapotřebí si stáhnout doporučenou šablonu projektu pro SeleF. Ta je navržena tak, že i při větším množství vytvořených testů je orientace v projektu poměrně snadná a intuitivní. Po stažení šablony bylo potřeba projekt otevřít ve zvoleném vývojovém prostředí. V IntelliJ IDEA byl projekt otevřen kliknutím na záložku File, následně byla zvolena možnost Open, a nakonec byl vybrán stažený projekt. Po dokončení těchto kroků se zobrazila připravená struktura pro tvorbu automatizovaných testů.



Obrázek 2 - Struktura SeleF projektu

Složka Drivers obsahuje veškeré potřebné ovladače pro provádění automatizovaných testů. Obsahuje ovladače pro jednotlivé prohlížeče, jako jsou například ChromeDriver, GeckoDriver nebo IEDriverServer. Jelikož je v obou testovacích scénářích obsažen případ, který vyžaduje připojení do databáze, musel autor ještě stáhnout a vložit do projektu ovladač, který toto umožňuje. Pro tyto účely byl vybrán ovladač OJDBC (Oracle Java Database Connectivity).

Dále se v projektu vyskytuje složka src, která je dále rozdělena na podsložky main a test. Do složky main patří veškeré pomocné třídy a metody, které se zde následně skládají do navazujících kroků podle potřeb testovacího scénáře. Do složky test pak patří veškeré spouštěcí třídy pro vytvořené scénáře.

Dalšími složkami v projektu jsou TestInputs a TestOutputs. Do TestInputs se vkládají veškerá vstupní data a parametry, které se následně v testech využívají. TestOutputs naopak obsahuje veškeré výstupy testů, jako jsou snímky obrazovky, výstupy běhu testů z konzole a mnoho dalších.

Soubor .gitignore slouží pro práci s nástrojem Git, což je nástroj pro správu verzí projektu. V tomto souboru jsou vypsány veškeré soubory a složky, které nejsou vhodné do verzování zahrnovat, díky čemuž s nimi Git již následně nepracuje.

Dále je v projektu obsažen soubor Jenkinsfile, který v sobě obsahuje parametry pro automatické spouštění testů v nástroji Jenkins. Tento nástroj umožňuje velké množství funkcí, jako je například spouštění testů ve vydefinované časy nebo spouštění testů na různých prostředích.

Soubor pom.xml je v projektu obsažen pro nástroj Maven, který slouží pro automatické sestavování projektů. V tomto souboru jsou obsaženy veškeré závislosti, které jsou potřeba při sestavení projektu stáhnout, aby bylo možné nástroj Selenium využívat, včetně SeleF nadstavby.

Posledním souborem projektu je Readme.md, který obsahuje informace o SeleF. Jsou zde popsány nově přidané funkcionality podle jednotlivých verzí nebo například kontakt na správce tohoto frameworku.

4.4.2 Identifikace webových prvků

Ve chvíli, kdy byl celý projekt připravený, přišla na řadu samotná tvorba testovacích skriptů. Jako první bylo zapotřebí si uložit do repozitáře webových prvků veškeré potřebné elementy. Pro tyto účely musel autor projít testovací scénáře ještě jednou manuálně, při čemž do souboru testElementRepository.csv, který je ve složce TestInputs, uložil využívané prvky. Pro identifikaci správných elementů byly využity nástroje pro vývojáře, které jsou zabudované přímo v prohlížeči Google Chrome, kde je v záložce Elements vidět celá struktura webové stránky.

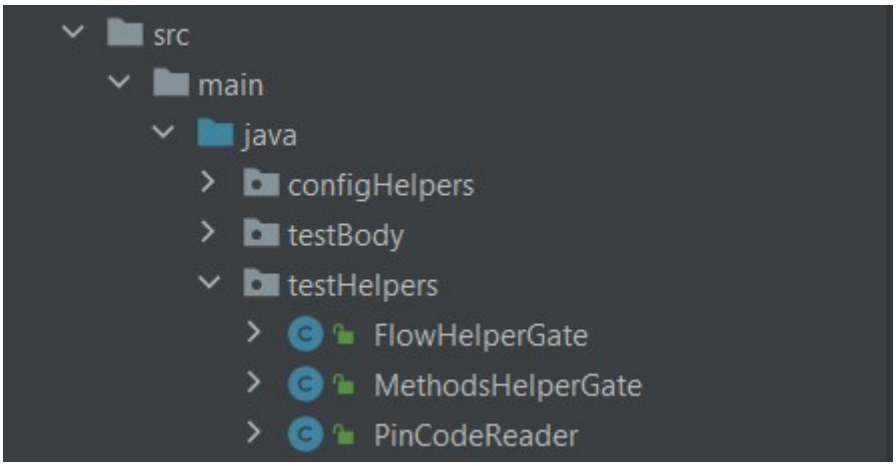
Na následujícím obrázku lze vidět soubor testElementRepository.csv. Na prvním řádku souboru jsou znázorněna pravidla, jak lze jakýkoliv element identifikovat a uložit. První sloupec vždy obsahuje autorem zvolený název pro element. Následují další tři sloupce, z nichž vždy musí být právě jeden vyplněný. Každý z nich umožňuje identifikovat prvek podle jiného pravidla. V případě, že má prvek unikátní id, vyplní se do sloupce id jeho hodnota. Tuto možnost lze pozorovat na řádku 37 a 38. Pokud má prvek unikátní jméno, vyplní se jeho hodnota do sloupce name, jak lze vidět na řádcích 16-19. Ve většině případů však webové prvky buď nemají vůbec id nebo jméno, nebo nejsou dostatečně unikátní. V tomto případě bylo potřeba využít identifikaci prvků pomocí XPath, což je jazyk pro adresování částí XML (Extensible Markup Language) dokumentů, který umožňuje vyhledání prvku například podle textu, třídy nebo typu. Tento typ identifikace lze vidět u všech ostatních elementů. Pro snadnou orientaci v prvcích jsou elementy rozděleny podle toho, na jaké stránce se v aplikaci Gate vyskytují.

```
elementName;id;name;xpath
------------------------MULTI-PAGE LOCATORS-----------------------------------------
primaryButton;;;//button[contains(@class,"btn-primary")]
acceptAllButton;;;//button[contains(text(), "Přijmout vše")]
bankConfirmButton;;;//input[@class="input button"]

------------------------BASIC INFO PAGE---------------------------------------------
basicInfoHeader;;;//h1[contains(text(), "Něco málo")]
basicInfoFirstName;;;//gt-text-input[@name="firstName"]//input
basicInfoLastName;;;//gt-text-input[@name="surname"]//input
basicInfoPhoneNumber;;;//gt-text-input[@name="mobilePhone"]//input
basicInfoEmail;;;//gt-text-input[@name="email"]//input

------------------------PIN CODE PAGE-----------------------------------------------
pinHeader;;;//h1[contains(text(), "Napište kód")]
pinFirstNumber;;number1;
pinSecondNumber;;number2;
pinThirdNumber;;number3;
pinFourthNumber;;number4;

------------------------IDENTIFICATION WAY PAGE-------------------------------------
identificationHeader;;;//h1[contains(text(), "Identifikace")]
documentsIdentificationButton;;;//button[text()="Nemám Bankovní identitu"]
bankIdentificationButton;;;//div[@class="bank-buttons"]/button

------------------------DOCUMENTS PAGES---------------------------------------------
documentsIntroHeader;;;//h1[contains(text(), "Nevadí")]
documentFileInput;;;//input[@type='file']
documentContinueButton;;;//button[text()="Pokračovat"]

------------------------SELFIE PAGE-------------------------------------------------
documentsSelfieFaceCheckbox;;;//gt-checkbox[@label="můj obličej"]//input
documentsSelfieIdCheckbox;;;//gt-checkbox[contains(@label, "občanský průkaz")]//input

------------------------BANK PAGES--------------------------------------------------
bankHeader;;;//h1/strong[text() = "Gate DEV"]
bankLoginFormUsername;username;;
bankLoginFormPassword;password;;

------------------------PERSONAL INFO CONFIRMATION PAGE-----------------------------
personalInfoHeader;;;//h1[contains(text(), "vše správně")]
personalInfoName;;;//gt-personal-info-basic//span[1]
personalInfoEmail;;;//gt-personal-info-basic//span[2]
personalInfoIDNumber;;;//p[text()="Číslo občanského průkazu:"]/strong
personalInfoBirthNumber;;;//p[text()="Rodné číslo:"]/strong
personalInfoBirthPlace;;;//p[text()="Místo narození:"]/strong
personalInfoDLNumber;;;//p[text()="Číslo řidičského průkazu: "]/strong
personalInfoPermanentResidenceStreet;;;//gt-personal-info-main-address//span[1]
personalInfoPermanentResidenceCity;;;//gt-personal-info-main-address//span[2]
personalInfoPermanentResidenceCountry;;;//gt-personal-info-main-address//span[3]
```

Obrázek 3 - obsah souboru testElementRepository.csv

4.4.3 Tvorba pomocných metod

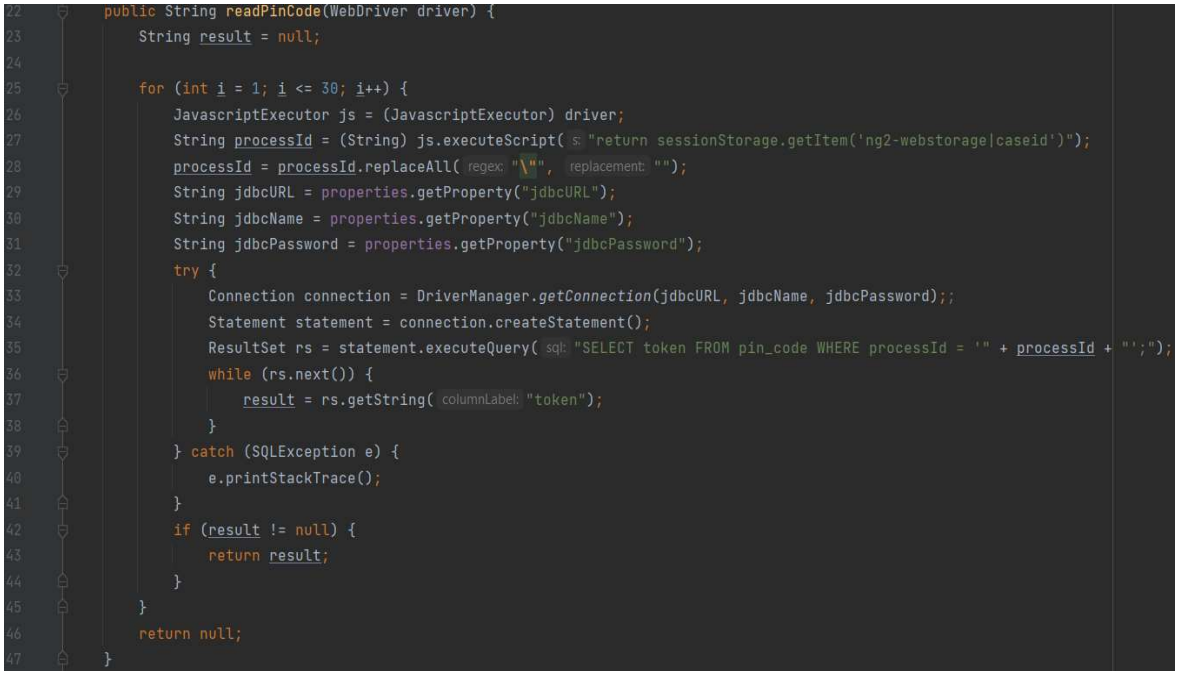
Po uložení veškerých webových prvků započala tvorba pomocných metod. Jak již bylo zmíněno, SeleF již nabízí velké množství připravených metod pro práci s elementy, avšak tyto metody neřeší veškeré problémy všech týmů a zaměřují se spíše na časté problémy, které se mohou vyskytovat ve všech aplikacích. Při testování aplikace Gate se objevují poměrně unikátní problémy, které připravené metody nepokrývají, a pro které bylo potřeba si vytvořit vlastní. Tyto metody jsou obsaženy v jednotlivých třídách v balíčku testHelpers. Kompletní třídy si lze vždy prohlédnout v příloze práce.



Obrázek 4 - Třídy v balíčku testHelpers

V testovacím případu TP-003 Zadání pin kódu je potřeba získat z tabulky pin_code token a ten následně vložit do příslušných polí na webové stránce. Pro logiku vyzvedávání kódu z databáze byla vytvořena třída PinCodeReader, ve které byla vytvořena třída readPinCode. Na následujícím obrázku si lze prohlédnout tělo zmíněné metody.

Pro vyzvednutí pin kódu z databáze je vždy nutné nejdříve zjistit id procesu daného průchodu aplikací. Tento údaj lze získat pouze ze session storage, která obsahuje užitečné informace o aktuální relaci. Aby bylo možné tento údaj dynamicky získávat, bylo potřeba využít rozhraní JavaScriptExecutor, které umožňuje spouštět na webových aplikacích JavaScriptový kód. Poté již stačilo zjistit, v jakém atributu je id procesu uložené, nechat si ho pomocí JavaScriptu příkazu vrátit a uložit si ho do proměnné v potřebném formátu. Následně si tato metoda ukládá do jednotlivých proměnných potřebné hodnoty pro připojení do databáze ze souboru properties.xml, který je ve složce TestInputs, zkusí se s nimi připojit přes rozhraní Connection a provede databázový dotaz na získání pin kódu přes rozhraní Statement. Pokud je vyzvednutí úspěšné, tak metoda vrací pin kód. Pokud se pin kód z nějakého důvodu vyzvednout nepodařilo, celý proces se opakuje a maximální počet opakování je třicet.

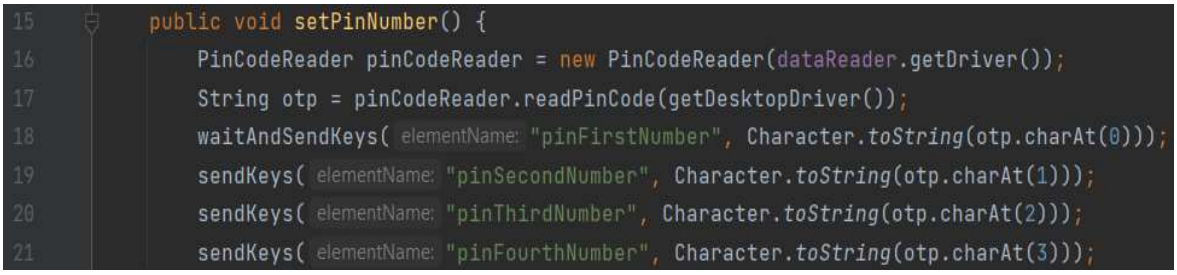


```
public String readPinCode(WebDriver driver) {
    String result = null;

    for (int i = 1; i <= 30; i++) {
        JavascriptExecutor js = (JavascriptExecutor) driver;
        String processId = (String) js.executeScript( s: "return sessionStorage.getItem('ng2-webstorage|caseid')");
        processId = processId.replaceAll( regex: "\"", replacement: "");
        String jdbcURL = properties.getProperty("jdbcURL");
        String jdbcName = properties.getProperty("jdbcName");
        String jdbcPassword = properties.getProperty("jdbcPassword");
        try {
            Connection connection = DriverManager.getConnection(jdbcURL, jdbcName, jdbcPassword);;
            Statement statement = connection.createStatement();
            ResultSet rs = statement.executeQuery( sql: "SELECT token FROM pin_code WHERE processId = '" + processId + "';");
            while (rs.next()) {
                result = rs.getString( columnLabel: "token");
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        if (result != null) {
            return result;
        }
    }
    return null;
}
```

Obrázek 5 - pomocná metoda readPinCode

Dále byla v balíčku testHelpers vytvořena třída MethodHelpersGate. V této třídě se již nacházejí metody, které přímo provádějí akce s webovými elementy, které SeleF neobsahuje a bylo potřeba je pro testování aplikace Gate vytvořit. Aby ve vytvořených metodách bylo možné využívat i SeleF metody, bylo zapotřebí aby třída MethodHelpersGate dědila ze SeleF třídy MethodHelpersDesktop. Autorem zde byly vytvořeny tři metody, kterými jsou setPinCode, uploadDesktopFile a waitMilliseconds.

Metoda setPinCode nejprve vytvoří instanci třídy PinCodeReader a následně provolá metodu readPinCode, která vrací pin z databáze. Hodnota je uložena do proměnné otp (one time password). Dále je v této metodě poprvé možné vidět příklady využití připravených SeleF metod waitAndSendKeys a sendKeys. Metoda waitAndSendKeys automaticky čeká, než bude element existovat a následně pošle do vybraného elementu ze souboru testElementRepository.csv zvolené klávesy. Tato metoda je použita pouze u prvního pole, jelikož pokud existuje právě to první, je jisté, že existují i ty ostatní, a proto na ně lze použít pouze metodu sendKeys.



```
public void setPinNumber() {
    PinCodeReader pinCodeReader = new PinCodeReader(dataReader.getDriver());
    String otp = pinCodeReader.readPinCode(getDesktopDriver());
    waitAndSendKeys( elementName: "pinFirstNumber", Character.toString(otp.charAt(0)));
    sendKeys( elementName: "pinSecondNumber", Character.toString(otp.charAt(1)));
    sendKeys( elementName: "pinThirdNumber", Character.toString(otp.charAt(2)));
    sendKeys( elementName: "pinFourthNumber", Character.toString(otp.charAt(3)));
```

Obrázek 6 - Pomocná metoda setPinNumber

Druhou vytvořenou metodou v této třídě je uploadDesktopFile, která slouží pro nahrání fotografií dokladů totožnosti. Je zde opět využito rozhraní JavaScriptExecutor, které v tomto případě nastavuje viditelnost skrytého prvku pro vkládání souborů. Do tohoto prvku je následně poslán soubor, k němuž se poskládá cesta z hodnoty uložené v properties.xml a názvu souboru, který se posílá jako argument metody. Po vložení souboru je potřeba, aby se celý proces na chvíli zastavil, aby se nahraný soubor stihl načíst, jinak zde v mnoha případech testy selhaly. Pro tento účel je využita metoda waitMilliseconds s parametrem, kolik milisekund je potřeba počkat. Tato metoda je taktéž vytvořena autorem v rámci této třídy. Pokud se jedná o obrazovku s vložením selfie fotografie, metoda uploadDesktopFile ještě potvrdí oba potřebné checkboxy a následně pokračuje v procesu.

```
24 @    public void uploadDesktopFile(String fileName) {
25          Properties properties = dataReader.getXMLProperties();
26          JavascriptExecutor js = (JavascriptExecutor) dataReader.getDriver();
27          waitForElementExist("documentFileInput");
28          js.executeScript( s: "document.getElementsByTagName('input')[0].style.visibility = \"visible\"");
29          waitAndSendKeys( elementName: "documentFileInput",  keys: properties.getProperty("winFileAddress") + fileName);
30          waitMilliseconds( mills: 500);
31          if (fileName.equals("selfie-id.jpg")) {
32              waitAndClickOnElement( elementName: "documentsSelfieFaceCheckbox");
33              clickOnElement( elementName: "documentsSelfieIdCheckbox");
34          }
35          waitAndClickOnElement( elementName: "primaryButton");
36          waitAndClickOnElement( elementName: "documentContinueButton");
37      }
```

Obrázek 7 - Pomocná metoda uploadDesktopFile

Dále byla v rámci balíčku testHelpers vytvořena třída FlowHelperGate, která v sobě obsahuje metody průchody jednotlivými testovacími případy procesu v aplikaci Gate. V rámci této práce bylo vytvořeno sedm testovacích případů, tudíž i zde se nachází sedm metod. Tato třída využívá SeleF metody i metody vytvořené autorem ve třídě MethodHelperGate.

Jako příklad je na následujícím obrázku zobrazena metoda basicInfoPageFlow, která ověřuje testovací případ pro doplnění osobních údajů klienta. Metoda nejdříve počká, než se objeví textové pole pro zadání jména a následně do něj pošle definovanou hodnotu. V tomto případě se jedná o hodnotu z testovacího případu „Tomáš". Ve chvíli, kdy existuje textové pole pro zadání jména, existují z pravidla i ostatní pole, a tudíž u zbylých prvků není nutné využívat čekání. V případě telefonního čísla si lze všimnout využití SeleF metody generatePhoneNumber, která generuje náhodné telefonní číslo. Při tvorbě testů a neustálého

spouštění pro ladění bylo totiž zjištěno, že pokud je spuštěno několik procesů se stejným telefonním číslem ve velmi krátkém intervalu, nefunguje zcela správně generování pin kódu. Nejedná se však o defekt, jelikož taková situace není v produkčním prostředí možná. Metoda je zakončena kontrolou, že se zobrazí stránka pro zadání pin kódu.

```
21    public void basicInfoPageFlow() {
22        waitAndSendKeys( elementName: "basicInfoFirstName",  keys: "Tomáš");
23        sendKeys( elementName: "basicInfoLastName",  keys: "Marný");
24        sendKeys( elementName: "basicInfoPhoneNumber", Integer.toString(generatePhoneNumber()));
25        sendKeys( elementName: "basicInfoEmail",  keys: "tomas.marny@seznam.cz");
26        waitAndClickOnElement( elementName: "primaryButton");
27        waitForElementExist("pinHeader");
28        assertText( elementName: "pinHeader",  text: "NAPIŠTE KÓD Z SMS ZPRÁVY");
29    }
```

Obrázek 8 - Metoda pro testovací případ „Doplnění osobních údajů"

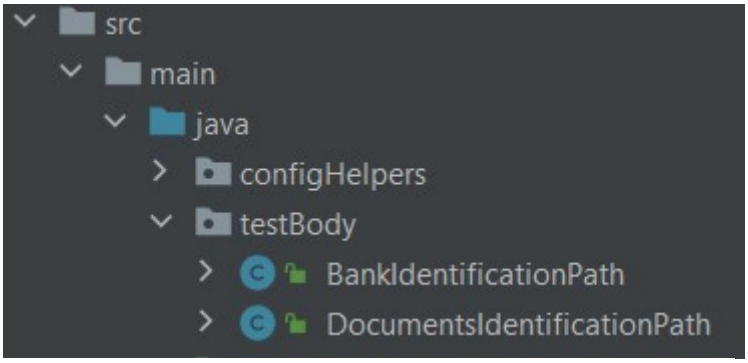
Druhým příkladem, který tu bude uveden je metoda pinNumberPageFlow pro testovací případ se zadáním pin kódu. Tato metoda nejdříve díky instanci třídy MethodHelperGate, která je inicializována v konstruktoru, provede vyzvednutí a zadání pin kódu z databáze Která využívá metodu setPinNumber ze třídy MethodHelperGate. Poté jen počká, než bude možné kliknout na tlačítko s pokračováním a zkontroluje, že se načetla správná webová stránka.

```
31    public void pinCodePageFlow() {
32        methodsHelperGate.setPinNumber();
33        waitForClickable( elementName: "primaryButton");
34        waitForElementExist("identificationHeader");
35        assertText( elementName: "identificationHeader",  text: "IDENTIFIKACE BANKOVNÍ IDENTITOU");
36    }
```

Obrázek 9 - Metoda pro testovací případ „Zadání pin kódu"

4.4.4 Sestavení testovacích scénářů

Jakmile byly vytvořeny veškeré metody pro průchody testovacími případy, nastal čas je poskládat do logicky navazujících kroků testovacích scénářů. Pro tyto účely je využíván balíček testBody, ve kterém vznikly dvě třídy, kde každá náleží právě jednomu testovacímu scénáři. Obě třídy balíčku testBody si lze opět prohlédnout v příloze práce.



Obrázek 11 - Třídy v balíčku testBody

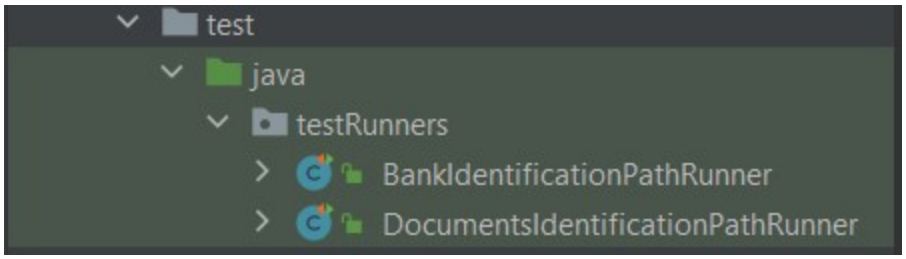
Jediná metoda, která se v obou třídách nachází, je metoda testRun. Tato metoda si nejdříve vytvoří instanci třídy FlowHelperGate, kde jsou metody pro jednotlivé testovací případy a následně na ní provádí požadované průchody aplikací.

```
13    public boolean testRun() {
14        FlowHelperGate flowHelper = new FlowHelperGate(dataReader);
15        flowHelper.welcomePageFlow();
16        flowHelper.basicInfoPageFlow();
17        flowHelper.pinCodePageFlow();
18        flowHelper.identificationWayPageFlow( selectedWayButton: "documentsIdentificationButton");
19        flowHelper.documentsPagesFlow();
20        flowHelper.personalInfoPageFlow( selectedWay: "documentsIdentification");
21        return true;
22    }
```

Obrázek 10 - Metoda pro testovací scénář „Ověření pomocí dokladů totožnosti"

4.4.5 Spuštění testů

Posledním krokem již bylo samotné spouštění testů a provedení měření. Pro tyto účely byl vytvořen balíček testRunners ve složce test. V tomto balíčku byly vytvořeny dvě třídy, jedna pro každý testovací scénář, které dědí ze SeleF třídy TestBaseDesktop. Obě třídy balíčku testRunners jsou opět k nahlédnutí v příloze práce.



Obrázek 12 - Třídy v balíčku testRunners

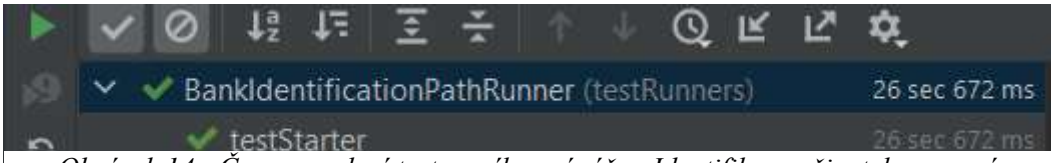
Obě třídy obsahují metody setUp, testStarter a cleanUp. U každé z těchto metod je využita některá z anotací frameworku JUnit, který slouží pro psaní jednotkových testů. U metody setUp je využita anotace @Before. V této metodě je potřeba vydefinovat vše, co

se má vykonat před spuštěním testů. V tomto případě se zde využívá SeleF metoda initDriver, která na základě definovaných argumentů nastaví, jaký ovladač webových prohlížečů bude využíván a jestli testy poběží lokálně, nebo na nějakém serveru. Metoda testStarter s anotací @Test slouží k samotnému spuštění testů. Nejdříve si uloží do proměnné jméno spouštěné třídy a následně do proměnné uloží výsledek běhu testovacího scénáře. Poslední metodou je cleanUp s anotací @After, která slouží pro vydefinování akcí, které se mají stát po doběhnutí testů. V této metodě je využita SeleF metoda quitDriver, která v základu zavře ovladač daného prohlížeče, pořídí snímek obrazovky posledního kroku testovacího scénáře a případně uloží výsledky běhu testů do jiných systémů.

```
12      @Before
13      public void setUp() {
14          initDriver( browser: "chrome",  platform: "local");
15      }
16
17      @Test
18      public void testStarter() {
19          testBodyName = DocumentsIdentificationPath.class.getSimpleName();
20          success = new DocumentsIdentificationPath(dataReader).testRun();
21      }
22
23      @After
24      public void cleanUp() {
25          quitDriver(success, testBodyName);
26      }
27  }
```

Obrázek 13 - Metody třídy DocumentsIdentificationPathRunner

Následně již stačilo testy spustit a naměřit časovou náročnost provedení. Testy lze spustit ve vývojovém prostředí IntelliJ IDEA několika způsoby stejně. Prvním je stisknutím kláves Shift + F10, druhým je kliknutím na tlačítko Run v pravém horním rohu, třetím je kliknutím na tlačítko Run u dané spouštěcí třídy, nebo lze kliknout na tlačítko Run přímo u metody testStarter. Následně se testy spustí a po jejich doběhnutí se v konzoli vývojového prostředí zobrazí výsledek běhu a čas, který to trvalo.



Obrázek 14 - Čas provedení testovacího scénáře „Identifikace uživatele pomocí bankovní identity"

5 Výsledky měření

Provedení manuálního ani automatizovaného testování webové aplikace Gate neodhalilo žádný softwarový defekt a prostředí bylo v průběhu testování zcela stabilní. Veškeré naměřené výsledky jsou tedy vypovídající o skutečné časové náročnosti obou testovacích metod.

5.1 Časová náročnost přípravy na testování

Do měření časové náročnosti přípravy na testování bylo v případě manuálního testování zahrnuto seznámení testerů s testovacími scénáři a následná příprava veškerých potřebných testovacích dat a přístupů. V tabulce níže si lze prohlédnout naměřené časy na přípravu testování manuálního provedení.

	Tester 1	Tester 2	Tester 3	Průměr
Příprava	5:35	12:21	14:55	11:35

Tabulka 7 - Časová náročnost přípravy manuálního testování

Při pozorování jednotlivých naměřených hodnot si lze všimnout znatelného časového rozdílu mezi prvním a posledním testerem. To lze velmi snadno odůvodnit, jelikož první tester je seniorní tester, který má s testováním aplikace Gate velmi bohaté zkušenosti, a zbytek týmu jsou juniorní testeři, kteří se s touto aplikací setkali poprvé. Po naměření jednotlivých hodnot byl vypočítán aritmetický průměr, který reprezentuje průměrnou časovou náročnost pro přípravu manuálního testování. Jeho hodnota byla vypočítána na 11 minut a 35 vteřin.

V případě automatizovaného testování byl do měření zahrnut veškerý čas strávený na tvorbě automatizovaných testů. Celková časová náročnost pro přípravu automatizovaných testů byla naměřena na 3 hodiny a 45 minut.

5.2 Časová náročnost provedení testů

Měření časové náročnosti provedení testů bylo provedeno pro každý testovací scénář zvlášť a následně byl pro naměřené hodnoty vypočítán aritmetický průměr. V tabulce níže lze vidět naměřené časy manuálního provedení testovacích scénářů.

	Tester 1	Tester 2	Tester 3	Průměr
TS-001	2:49	5:48	5:39	5:15
TS-002	2:10	3:36	3:58	3:24

Tabulka 8 - Časová náročnost manuálního provedení testů

Průměrný čas manuálního provedení prvního testovacího scénáře, který se zabývá ověřením identity klienta pomocí dokladů totožnosti a vyfocením selfie fotografie, byl vypočítán na 5 minuty a 15 vteřin. Průměrný čas pro provedení druhého scénáře, který ověřuje funkcionalitu ověření identity klienta přes bankovní identitu z jiné banky, byl vypočítán na 3 minuty a 24 vteřin. Po sečtení průměrné časové náročnosti provedení obou scénářů vychází průměrný čas manuálního provedení testů na 8 minut a 39 vteřin.

V případě automatizovaného provedení testů byly testy spuštěny u obou testovacích scénářů třikrát, aby byl počet naměřených hodnot stejný jako v případě manuálního provedení. V následující tabulce si lze prohlédnout naměřené hodnoty.

	Automatizované provedení 1	Automatizované provedení 2	Automatizované provedení 3	Průměr
TS-001	0:52	0:55	0:53	0:53
TS-002	0:28	0:26	0:27	0:27

Tabulka 9 - Časová náročnost automatizovaného provedení testů

U automatizovaného provedení testů si lze všimnou konzistentních výsledků. V případě testování prvního testovacího scénáře byl naměřen průměrný čas provedení 53 vteřin a v případě druhého scénáře pouze 27 vteřin. Celkový průměrný čas provedení obou testovacích scénářů je tedy 1 minuta a 20 vteřin.

5.3 Srovnání výsledků

Jednotlivé výsledky měření potvrzují tvrzení z teoretické části práce, že automatizované testování je časově náročnější na přípravu, avšak rychlost provedení testů je mnohem rychlejší. Časová náročnost na přípravu manuálního testování byla vypočítána na 11 minut a 33 vteřin a na přípravu automatizovaného testování 3 hodiny a 45 minut. V tomto případě se tedy jedná o zhruba dvacetkrát delší dobu na přípravu testování v případě automatizace.

U automatizovaného testování byla naměřena průměrná rychlost provedení obou testovacích scénářů 1 minuta a 20 vteřin. V případě manuálního testování byla naměřena hodnota průměrného času provedení obou scénářů 8 minut a 39 vteřin. Automatizované provedení testů je tedy u testování těchto funkcionalit téměř sedmkrát rychlejší.

Aby se testovacímu týmu vrátil zainvestovaný čas do tvorby automatizovaných testů, je potřeba, aby testy proběhly třicetkrát. Ověřování těchto funkcionalit do této chvíle probíhalo dvakrát denně každý pracovní den, a i nově vytvořené automatizované testy budou pokračovat v této intenzitě. Celá časová investice se tedy testovacímu týmu vrátí po třech týdnech a veškeré další testování již bude ve prospěch automatizovaných testů.

6 Závěr

Cílem teoretické části této bakalářské práce bylo seznámit čtenáře se zásadními pojmy z oblasti testování softwaru a automatizovaného testování včetně analýzy využívaných nástrojů. Byly zde čtenáři vysvětleny pojmy jako je testování softwaru, softwarový defekt, testovací dokumentace, životní cyklus testování softwaru, metody testování a úrovně funkčního testování. Dále zde byla popsána problematika automatizovaného testování, kde byly popsány rozdíly mezi manuálním a automatizovaným testováním, jaké výhody automatizace poskytuje, jaké testy je vhodné automatizovat a naopak jaké testy je nevhodné automatizovat. Teoretická část práce byla následně zakončena analýzou využívaných nástrojů pro automatizované testování webových aplikací, na jejímž základě byl vybrán nástroj Selenium, jako nástroj pro praktickou část práce.

Cílem praktické části bylo vytvořit testovací scénář a automatizovaný test pro webovou aplikaci v bankovním sektoru a následně změřit a porovnat časovou náročnost automatizovaného testu s testem manuálním. Tato část práce se nejdříve věnovala popisu webové aplikace Komerční banky, na které byla celá praktická část realizována. Následovalo vytvoření dvou testovacích scénářů a jejich manuální provedení celým testovacím týmem. Dále byly vytvořeny automatizované testy v nástroji Selenium v kombinaci s programovacím jazykem Java. V obou případech byla při provádění testů měřena celková časová náročnost, na jejímž základě bylo vypočítáno, že časová investice do tvorby automatizovaných testů se v tomto případě testovacímu týmu vrátí po třech týdnech. Po uplynutí této doby začne tým z každého provedení automatizovaných testů získávat potřebný čas pro testování nové aplikace.

Pro čtenáře je hlavním přínosem této práce shrnutí důležitých pojmů z oblasti testování softwaru na jednom místě. Pro banku jsou přínosem vytvořené automatizované testy, které testerům ušetří mnoho času. Pro autora práce je přínosem především prohloubení znalostí této problematiky a získané zkušenosti při realizaci praktické části v reálném prostředí.

7 Seznam použitých zdrojů

[1] OLSEN, Klaus, Meile POSTHUMA a Stephanie ULRICH ISQBT. *Certified tester foundation level* [online]. © ISQBT 2018 [cit. 01.03.2023]. Dostupné z: https://istqb-main-web-prod.s3.amazonaws.com/media/documents/ISTQB-CTFL_Syllabus_2018_v3.1.1.pdf

[2] PATTON, Ron. *Testování softwaru*. Praha: Computer Press, 2002. ISBN 80–722-6636-5.

[3] *What is software testing: Everything you need to know* [online]. © Software Testing Material 2023 [cit. 01.03.2023]. Dostupné z: https://www.softwaretestingmaterial.com/software-testing/

[4] BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽEL, Peter SVOBODA, Zdeněk GRÖSSL, Martin KOMÁREK, Ondřej MACEK a Radoslav MLYNÁŘ. *Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu*. Praha: Grada, 2016. ISBN 978-80-247-5594-6.

[5] *The exponential cost of fixing bugs* [online]. © DeepSource 2023 [cit. 01.03.2023]. Dostupné z: https://deepsource.io/blog/exponential-cost-of-fixing-bugs/

[6] *Testovací dokumentace – plán, scénář, případ* [online]. [cit. 01.03.2023]. Dostupné z: http://test.swtestovani.cz/index.php?option=com_content&view=article&id=15:testovaci-dokumentace-plan-scena-pipad&catid=3:zaklady&Itemid=11

[7] *Test Script – testovací scénář. Testování softwaru* [online]. 2014 [cit. 01.03.2023]. Dostupné z: http://testovanisoftwaru.cz/dokumentace-v-testovani/test-script-testovaci-scenar/

[8] *Testovací tým. Testování softwaru* [online]. 2014 [cit. 01.03.2023]. Dostupné z: http://testovanisoftwaru.cz/manualni-testovani/testovaci-tym/

[9] KRÁLOVÁ, Iveta. *Role v testování. Metodika testování podle mezinárodních praktik a standardů* [online]. © 2013 [cit. 01.03.2023]. Dostupné z: http://metodikamezitest.asp2.cz/Metodika_testovani/roles/test_manazer_FB59FAD6.html

[10] Software Testing Life Cycle (STLC) [online]. © GeeksForGeeks 2023 [cit. 01.03.2023]. Dostupné z: https://www.geeksforgeeks.org/software-testing-life-cycle-stlc/

[11] *Introduction to Software Testing Life Cycle (STLC): Definition and Phases* [online]. © SeaLight 2023 [cit. 01.03.2023]. Dostupné z: https://www.sealights.io/software-quality/an-introduction-to-software-testing-life-cycle-stlc-definition-and-phases/

[12] *Static Testing vs Dynamic Testing* [online]. © 2011 [cit. 01.03.2023]. Dostupné z: https://www.javatpoint.com/static-testing-vs-dynamic-testing

[13] *Black Box Testing vs White Box Testing – Software testing fundamentals*. [online]. 2020 [cit. 01.03.2023]. Dostupné z: https://softwaretestingfundamentals.com/black-box-testing-vs-white-box-testing/

[14] HAMILTON, Thomas. *Functional Testing vs Non-Functional Testing* [online] 2023 [cit. 01.03.2023]. Dostupné z: https://www.guru99.com/functional-testing-vs-non-functional-testing.html

[15] *Smoke vs. Regression Tests*. [online]. © Functionize, Inc 2023 [cit. 01.03.2023]. Dostupné z: https://www.functionize.com/blog/smoke-vs-regression-tests

[16] NAIK, Kshirasagar, Priyadarshi TRIPATHY. *Software Testing and Quality Assurance: Theory and Practice*, John Wiley & Sons, Inc, 2008, ISBN 978-0-471-78911-6.

[17] TAWDE, Swati. *Levels of software testing* [online]. © EDUCBA 2022 [cit. 01.03.2023]. Dostupné z https://www.educba.com/levels-of-software-testing/

[18] KALWAN, Anamika. *What are different levels of software testing* [online]. © Brain4ce Education Solutions 2023 [cit. 01.03.2023]. Dostupné z: https://www.edureka.co/blog/software-testing-levels/

[19] KINSBRUNER, Eran. *Manual testing vs Automated Testing* [online]. © Perfecto 2022 [cit. 01.03.2023]. Dostupné z: https://www.perfecto.io/blog/automated-testing-vs-manual-testing-vs-continuous-testing

[20] CRISPIN, Lisa a Janet GREGORY. *Agile testing: a practical guide for testers and agile teams*. New Jersey: Addison-Wesley, 2009. ISBN 978–0-321-53446-0.

[21] *Selenium: Definition, How it works and Why you need it* [online]. © BrowserStack 2023 [cit. 01.03.2023]. Dostupné z: https://www.browserstack.com/selenium

[22] *A Comparision of Automated Testing Tools* [online]. © Katalon 2023 [cit. 01.03.2023]. Dostupné z: https://katalon.com/resources-center/blog/comparison-automated-testing-tools

[23] *What is Katalon Studio, Overview and Tour Of Features* [online]. © The QA Lead 2023 [cit. 01.03.2023]. Dostupné z: https://theqalead.com/test-management/katalon-studio-overview/

[24] *Katalon – comphrensive platform, flexible pricing* [online]. © Katalon 2023 [cit. 01.03.2023]. Dostupné z: https://katalon.com/pricing

[25] KHETARPAL, Aashish. *What is Cypress: Introduction and Architecture* [online]. © ToolsQA 2021 [cit. 01.03.2023]. Dostupné z: https://www.toolsqa.com/cypress/what-is-cypress/

[26] *Why Cypress: Cypress Documentation* [online]. © 2023 Cypress.io [cit. 01.03.2023]. Dostupné z: https://docs.cypress.io/guides/overview/why-cypress

[27] UNADKAT, Jash. *Cypress vs Selenium: Key Differences* [online]. © BrowserStack 2023 [cit. 01.03.2023]. Dostupné z: https://www.browserstack.com/guide/cypress-vs-selenium

[28] MAHAJAN, Amod. *Selenium vs Testcomplete: A quick comparison* [online]. © KnowledgeHut Solutions 2023 [cit. 01.03.2023]. Dostupné z: https://www.knowledgehut.com/blog/software-testing/selenium-vs-testcomplete

[29] *SmartBear TestComplete: pricing* [online]. © SmartBear Software 2023 [cit. 01.03.2023]. Dostupné z: https://smartbear.com/product/testcomplete/pricing/

8 Seznam obrázků, tabulek a grafů

8.1 Seznam obrázků

Obrázek 1 - Úrovně funkčního testování 26
Obrázek 2 - Struktura SeleF projektu 39
Obrázek 3 - obsah souboru testElementRepository.csv 42
Obrázek 4 - Třídy v balíčku testHelpers 43
Obrázek 5 - pomocná metoda readPinCode 44
Obrázek 6 - Pomocná metoda setPinNumber 44
Obrázek 7 - Pomocná metoda uploadDesktopFile 45
Obrázek 8 - Metoda pro testovací případ „Doplnění osobních údajů” 46
Obrázek 9 - Metoda pro testovací případ „Zadání pin kódu” 46
Obrázek 10 - Metoda pro testovací scénář „Ověření pomocí dokladů totožnosti" 47
Obrázek 11 - Třídy v balíčku testBody 47
Obrázek 12 - Třídy v balíčku testRunners 47
Obrázek 13 - Metody třídy DocumentsIdentificationPathRunner 48
Obrázek 14 - Čas provedení testovacího scénáře „Identifikace uživatele pomocí bankovní identity” 48

8.2 Seznam tabulek

Tabulka 1 - Charakteristiky nástroje Selenium 31
Tabulka 2 - Charakteristiky nástroje Katalon Studio 31
Tabulka 3 - Charakteristiky nástroje Cypress 32
Tabulka 4 - Charakteristiky nástroje TestComplete 33
Tabulka 5 - Testovací případ „Doplnění osobních údajů" 36
Tabulka 6 - Testovací scénář „Identifikace uživatele pomocí dokladů totožnosti" 38
Tabulka 7 - Časová náročnost přípravy manuálního testování 49
Tabulka 8 - Časová náročnost manuálního provedení testů 50
Tabulka 9 - Časová náročnost automatizovaného provedení testů 50

8.3 Seznam grafů

Graf 1 - Náklady na opravu defektu 15
Graf 2 - Příčiny vzniku defektů 16

Přílohy

A Testovací případy

TP-001	Možnost založení účtu u KB		
Prerekvizity			
• Přístup do testovacího prostředí Gate.			
Vstupní data			
• Nejsou potřeba.			
Krok	Popis	Očekávaný výsledek	Reálný výsledek
1.	Otevřete v prohlížeči adresu url: https://gate-dev.kb.cz/online/welcome/muj-ucet-plus.	Zobrazí se stránka uvítací stránka se stručným popisem procesu.	
2.	Klikněte na tlačítko „ZALOŽIT ÚČET U KB”.	Zobrazí se stránka pro vyplnění základních údajů o klientovi.	

Příloha A 1 Testovací případ „Možnost založení účtu u KB”

TP-003	Zadání pin kódu		
Prerekvizity			
• Přístup do testovacího prostředí Gate. • Přístup do testovací databáze pro aplikaci Gate. • Dokončený TP-002 Doplnění osobních údajů.			
Vstupní data			
• Pin kód pro daný proces z tabulky „pin_code” a sloupce „token”.			
Krok	Popis	Očekávaný výsledek	Reálný výsledek
1.	Doplňte do příslušných polí jednotlivé číslice pin kódu.	Pin kód je vyplněný a je možné kliknout na tlačítko „POKRAČOVAT”.	
2.	Klikněte na tlačítko „POKRAČOVAT”.	Zobrazí se stránka pro vybrání způsobu identifikace klienta.	

Příloha A 2 Testovací případ „Zadání pin kódu”

TP-004	Volba cesty ověření identity		
Prerekvizity			
• Přístup do testovacího prostředí Gate. • Dokončený TP-003 Zadání pin kódu.			
Vstupní data			
• Nejsou potřeba.			
Krok	Popis	Očekávaný výsledek	Reálný výsledek
1.	V případě ověření identity přes doklady totožnosti klikněte na tlačítko „NEMÁM BANKOVNÍ IDENTITU”.	Zobrazí se stránka informační stránka pro ověření cestou s doklady.	
1.	V případě ověření identity přes bankovní identitu klikněte na tlačítko „Mock banka“.	Zobrazí se stránka pro přihlášení do internetového bankovnictví.	

Příloha A 3 Testovací případ „Volba cesty ověření identity”

TP-005	Vložení dokladů totožnosti		
Prerekvizity			
• Přístup do testovacího prostředí Gate. • Dokončený TP-004 Volba cesty ověření identity.			
Vstupní data			
Testovací doklady totožnosti: • Přední strana občanského průkazu – id-front.jpg • Zadní strana občanského průkazu – id-back.jpg • Přední strana řidičského průkazu – dl-front.jpg • Selfie fotografie s občanským průkazem – selfie-id.jpg			
Krok	Popis	Očekávaný výsledek	Reálný výsledek
1.	Klikněte na tlačítko „POKRAČOVAT”.	Zobrazí se stránka pro vložení přední strany občanského průkazu	

2.	Klikněte na tlačítko „VYBRAT FOTKU”, vložte přední stranu občanské průkazu a klikněte na tlačítko „POKRAČOVAT”.	Přední strana občanského průkazu se zpracuje správně a zobrazí se stránka pro vložení zadní strany občanského průkazu.	
3.	Klikněte na tlačítko „VYBRAT FOTKU”, vložte zadní stranu občanské průkazu a klikněte na tlačítko „POKRAČOVAT”.	Zadní strana občanského průkazu se zpracuje správně a zobrazí se stránka pro vložení přední strany druhého dokladu totožnosti.	
4.	Klikněte na tlačítko „VYBRAT FOTKU”, vložte přední stranu řidičského průkazu a klikněte na tlačítko „POKRAČOVAT”.	Přední strana řidičského průkazu se zpracuje správně a zobrazí se stránka pro pořízení selfie fotografie s občanským průkazem.	
5.	Klikněte na tlačítko „VYFOTIT”, potvrďte oba checkboxy o správnosti a „POKRAČOVAT”.	Zobrazí se stránka pro kontrolu načtených údajů.	

Příloha A 4 Testovací případ „Vložení dokladů totožnosti”

TP-006	Přihlášení přes bankovní identitu
Prerekvizity	
• Přístup do testovacího prostředí Gate. • Dokončený TP-004 Volba cesty ověření identity.	
Vstupní data	
Přihlašovací údaje to testovacího internetového bankovnictví:	

• Jméno - LeosV • Heslo - password			
Krok	Popis	Očekávaný výsledek	Reálný výsledek
1.	Do polí „Jméno (Name)“ a „Heslo (Password)“ vložte příslušné hodnoty uvedené ve vstupních datech a klikněte na tlačítko „Přihlásit se (Login)“.	Zobrazí se stránka informující o poskytovaných informacích aplikaci Gate.	
2.	Klikněte na tlačítko „Potvrdit přístup“.	Zobrazí se stránka pro kontrolu načtených údajů.	

Příloha A 5 Testovací případ „Přihlášení přes bankovní identitu”

TP-007	Kontrola načtených údajů		
Prerekvizity			
• Přístup do testovacího prostředí Gate. • Dokončený TP-005 Vložení dokladů totožnosti nebo TP-006 Přihlášení přes bankovní identitu			
Vstupní data			
• Nejsou potřeba.			
Krok	Popis	Očekávaný výsledek	Reálný výsledek
1.	V případě cesty ověření přes doklady totožnosti zkontrolujte správnost načtených údajů z dokladů totožnosti.	Veškeré údaje jsou shodné s údaji na dokladech totožnosti.	
1.	V případě cesty ověření přes bankovní identitu zkontrolujte správnost načtených údajů z mock banky.	Veškeré údaje jsou shodné s údaji z bankovní identity.	

Příloha A 6 Testovací případ „Kontrola načtených údajů”

B Testovací scénář

TS-001	Identifikace uživatele pomocí bankovní identity	
Oblast k otestování		
Testovací scénář je zaměřen na „happy day" ověření uživatelské cesty při ověření totožnosti přes bankovní identitu z jiné banky.		
Testovací případ	Očekávaný výsledek	Reálný výsledek
TP-001 Možnost založení účtu u KB	Zobrazí se stránka pro vyplnění základních údajů o klientovi.	
TP-002 Doplnění osobních údajů	Vygeneruje se pin kód, který se uloží do databáze a zobrazí se stránka pro jeho vyplnění.	
TP-003 Zadání pin kódu	Zobrazí se stránka pro vybrání způsobu identifikace klienta.	
TP-004 Volba cesty ověření identity	Zobrazí se stránka pro přihlášení do internetového bankovnictví.	
TP-006 Přihlášení přes bankovní identitu	Zobrazí se stránka pro kontrolu načtených údajů.	
TP-007 Kontrola načtených údajů	Veškeré údaje jsou shodné s údaji z bankovní identity.	

Příloha B 1 Testovací scénář „Identifikace uživatele pomocí bankovní identity"

C Třídy balíčku testHelpers

```java
package testHelpers;

import coreHelpers.DataReader;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.JavascriptExecutor;

import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Properties;

public class PinCodeReader {
    private final Properties properties;

    public PinCodeReader(WebDriver driver) {
        DataReader dataReader = new DataReader(driver);
        properties = dataReader.getXMLProperties();
    }

    public String readPinCode(WebDriver driver) {
        String result = null;

        for (int i = 1; i <= 30; i++) {
            JavascriptExecutor js = (JavascriptExecutor) driver;
            String processId = (String) js.executeScript( s: "return sessionStorage.getItem('ng2-webstorage|caseid')");
            processId = processId.replaceAll( regex: "\"", replacement: "");
            String jdbcURL = properties.getProperty("jdbcURL");
            String jdbcName = properties.getProperty("jdbcName");
            String jdbcPassword = properties.getProperty("jdbcPassword");
            try {
                Connection connection = DriverManager.getConnection(jdbcURL, jdbcName, jdbcPassword);;
                Statement statement = connection.createStatement();
                ResultSet rs = statement.executeQuery( sql: "SELECT token FROM pin_code WHERE case_id = '" + processId + "';");
                while (rs.next()) {
                    result = rs.getString( columnLabel: "token");
                }
            } catch (SQLException e) {
                e.printStackTrace();
            }
            if (result != null) {
                return result;
            }
        }
        return null;
    }
}
```

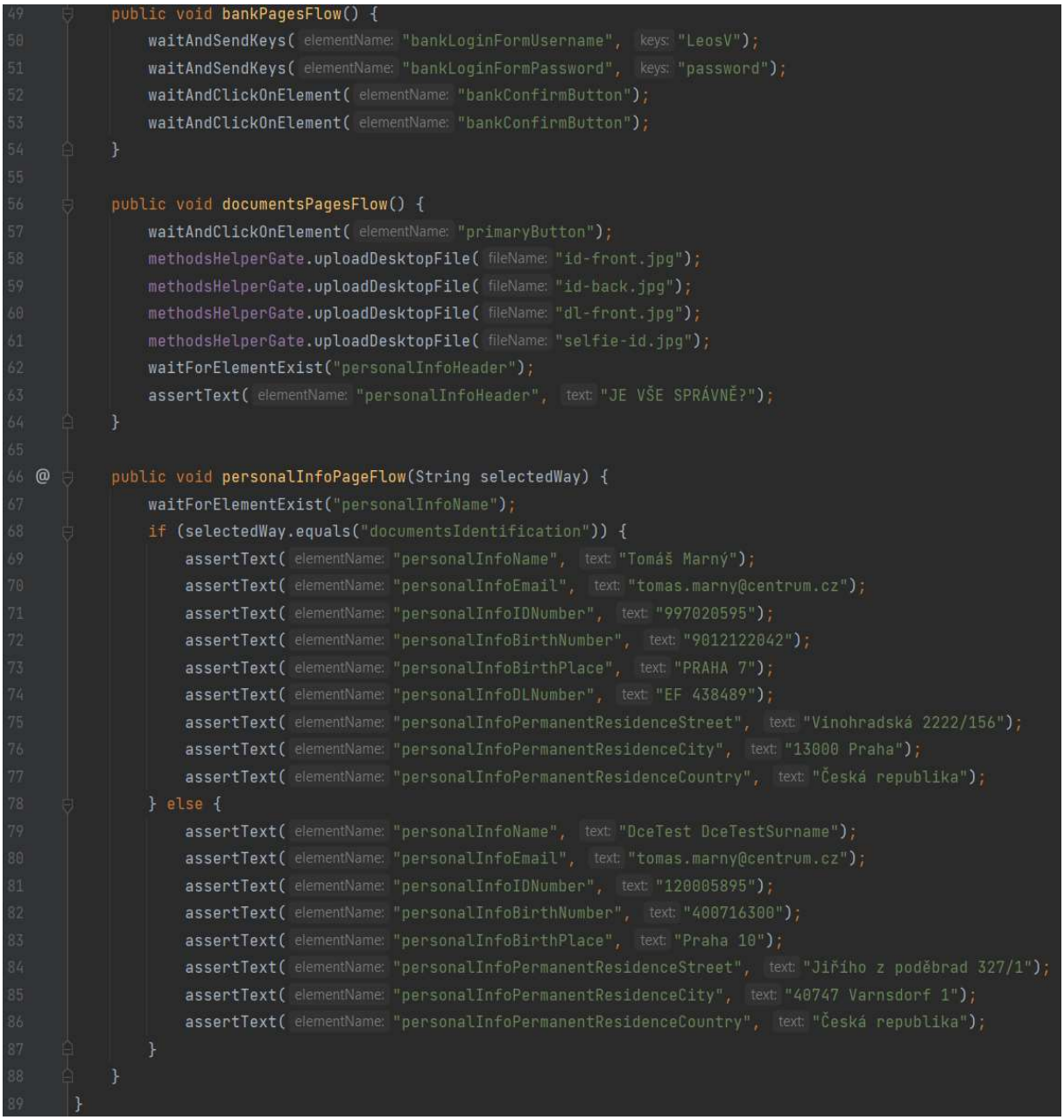
Příloha C 1 Třída PinCodeReader

```java
package testHelpers;

import coreHelpers.DataReader;
import coreHelpers.MethodHelperDesktop;
import org.openqa.selenium.JavascriptExecutor;

import java.util.Properties;

public class MethodsHelperGate extends MethodHelperDesktop {

    public MethodsHelperGate(DataReader dataReader) {
        super(dataReader);
    }

    public void setPinNumber() {
        PinCodeReader pinCodeReader = new PinCodeReader(dataReader.getDriver());
        String otp = pinCodeReader.readPinCode(getDesktopDriver());
        waitAndSendKeys( elementName: "pinFirstNumber", Character.toString(otp.charAt(0)));
        sendKeys( elementName: "pinSecondNumber", Character.toString(otp.charAt(1)));
        sendKeys( elementName: "pinThirdNumber", Character.toString(otp.charAt(2)));
        sendKeys( elementName: "pinFourthNumber", Character.toString(otp.charAt(3)));
    }

    public void uploadDesktopFile(String fileName) {
        Properties properties = dataReader.getXMLProperties();
        JavascriptExecutor js = (JavascriptExecutor) dataReader.getDriver();
        waitForElementExist("documentFileInput");
        js.executeScript( s: "document.getElementsByTagName('input')[0].style.visibility = \"visible\"");
        waitAndSendKeys( elementName: "documentFileInput",  keys: properties.getProperty("winFileAddress") + fileName);
        waitMilliseconds( mills: 500);
        if (fileName.equals("selfie-id.jpg")) {
            waitAndClickOnElement( elementName: "documentsSelfieFaceCheckbox");
            clickOnElement( elementName: "documentsSelfieIdCheckbox");
        }
        waitAndClickOnElement( elementName: "primaryButton");
        waitAndClickOnElement( elementName: "documentContinueButton");
    }

    public void waitMilliseconds(int mills) {
        try {
            Thread.sleep(mills);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

Příloha C 2 Třída MethodsHelperGate

```java
package testHelpers;

import coreHelpers.DataReader;

public class FlowHelperGate extends MethodsHelperGate {
    MethodsHelperGate methodsHelperGate;

    public FlowHelperGate(DataReader dataReader) {
        super(dataReader);
        methodsHelperGate = new MethodsHelperGate(dataReader);
    }

    public void welcomePageFlow() {
        openURL(baseURL);
        waitAndClickOnElement( elementName: "acceptAllButton");
        clickOnElement( elementName: "primaryButton");
        waitForElementExist("basicInfoHeader");
        assertText( elementName: "basicInfoHeader",  text: "NĚCO MÁLO O VÁS");
    }

    public void basicInfoPageFlow() {
        waitAndSendKeys( elementName: "basicInfoFirstName",  keys: "Tomáš");
        sendKeys( elementName: "basicInfoLastName",  keys: "Marný");
        sendKeys( elementName: "basicInfoPhoneNumber", Integer.toString(generatePhoneNumber()));
        sendKeys( elementName: "basicInfoEmail",  keys: "tomas.marny@centrum.cz");
        waitAndClickOnElement( elementName: "primaryButton");
        waitForElementExist("pinHeader");
        assertText( elementName: "pinHeader",  text: "NAPIŠTE KÓD Z SMS ZPRÁVY");
    }

    public void pinCodePageFlow() {
        methodsHelperGate.setPinNumber();
        waitForClickable( elementName: "primaryButton");
        waitForElementExist("identificationHeader");
        assertText( elementName: "identificationHeader",  text: "IDENTIFIKACE BANKOVNÍ IDENTITOU");
    }

    public void identificationWayPageFlow(String selectedWayButton) {
        waitForClickable(selectedWayButton);
        if (selectedWayButton.equals("bankIdentificationButton")) {
            waitForElementExist("bankHeader");
            assertText( elementName: "bankHeader",  text: "Gate DEV");
        } else {
            waitForElementExist("documentsIntroHeader");
            assertText( elementName: "documentsIntroHeader",  text: "NEVADÍ... MÍSTO BANKOVNÍ IDENTITY POUŽIJETE DOKLADY");
        }
    }
```

Příloha C 3 Třída FlowHelperGate první část



```
    public void bankPagesFlow() {
        waitAndSendKeys( elementName: "bankLoginFormUsername",  keys: "LeosV");
        waitAndSendKeys( elementName: "bankLoginFormPassword",  keys: "password");
        waitAndClickOnElement( elementName: "bankConfirmButton");
        waitAndClickOnElement( elementName: "bankConfirmButton");
    }

    public void documentsPagesFlow() {
        waitAndClickOnElement( elementName: "primaryButton");
        methodsHelperGate.uploadDesktopFile( fileName: "id-front.jpg");
        methodsHelperGate.uploadDesktopFile( fileName: "id-back.jpg");
        methodsHelperGate.uploadDesktopFile( fileName: "dl-front.jpg");
        methodsHelperGate.uploadDesktopFile( fileName: "selfie-id.jpg");
        waitForElementExist("personalInfoHeader");
        assertText( elementName: "personalInfoHeader",  text: "JE VŠE SPRÁVNĚ?");
    }

@   public void personalInfoPageFlow(String selectedWay) {
        waitForElementExist("personalInfoName");
        if (selectedWay.equals("documentsIdentification")) {
            assertText( elementName: "personalInfoName",  text: "Tomáš Marný");
            assertText( elementName: "personalInfoEmail",  text: "tomas.marny@centrum.cz");
            assertText( elementName: "personalInfoIDNumber",  text: "997020595");
            assertText( elementName: "personalInfoBirthNumber",  text: "9012122042");
            assertText( elementName: "personalInfoBirthPlace",  text: "PRAHA 7");
            assertText( elementName: "personalInfoDLNumber",  text: "EF 438489");
            assertText( elementName: "personalInfoPermanentResidenceStreet",  text: "Vinohradská 2222/156");
            assertText( elementName: "personalInfoPermanentResidenceCity",  text: "13000 Praha");
            assertText( elementName: "personalInfoPermanentResidenceCountry",  text: "Česká republika");
        } else {
            assertText( elementName: "personalInfoName",  text: "DceTest DceTestSurname");
            assertText( elementName: "personalInfoEmail",  text: "tomas.marny@centrum.cz");
            assertText( elementName: "personalInfoIDNumber",  text: "120005895");
            assertText( elementName: "personalInfoBirthNumber",  text: "400716300");
            assertText( elementName: "personalInfoBirthPlace",  text: "Praha 10");
            assertText( elementName: "personalInfoPermanentResidenceStreet",  text: "Jiřího z poděbrad 327/1");
            assertText( elementName: "personalInfoPermanentResidenceCity",  text: "40747 Varnsdorf 1");
            assertText( elementName: "personalInfoPermanentResidenceCountry",  text: "Česká republika");
        }
    }
}
```

Příloha C 4 Třída FlowHelperGate druhá část

D Třídy balíčku testBody

```java
package testBody;

import coreHelpers.DataReader;
import coreHelpers.MethodHelperDesktop;
import testHelpers.FlowHelperGate;

public class DocumentsIdentificationPath extends MethodHelperDesktop {

    public DocumentsIdentificationPath(DataReader dataReader) {
        super(dataReader);
    }

    public boolean testRun() {
        FlowHelperGate flowHelper = new FlowHelperGate(dataReader);
        flowHelper.welcomePageFlow();
        flowHelper.basicInfoPageFlow();
        flowHelper.pinCodePageFlow();
        flowHelper.identificationWayPageFlow( selectedWayButton: "documentsIdentificationButton");
        flowHelper.documentsPagesFlow();
        flowHelper.personalInfoPageFlow( selectedWay: "documentsIdentification");
        return true;
    }
}
```

Příloha D 1 Třída DocumentIdentificationPath

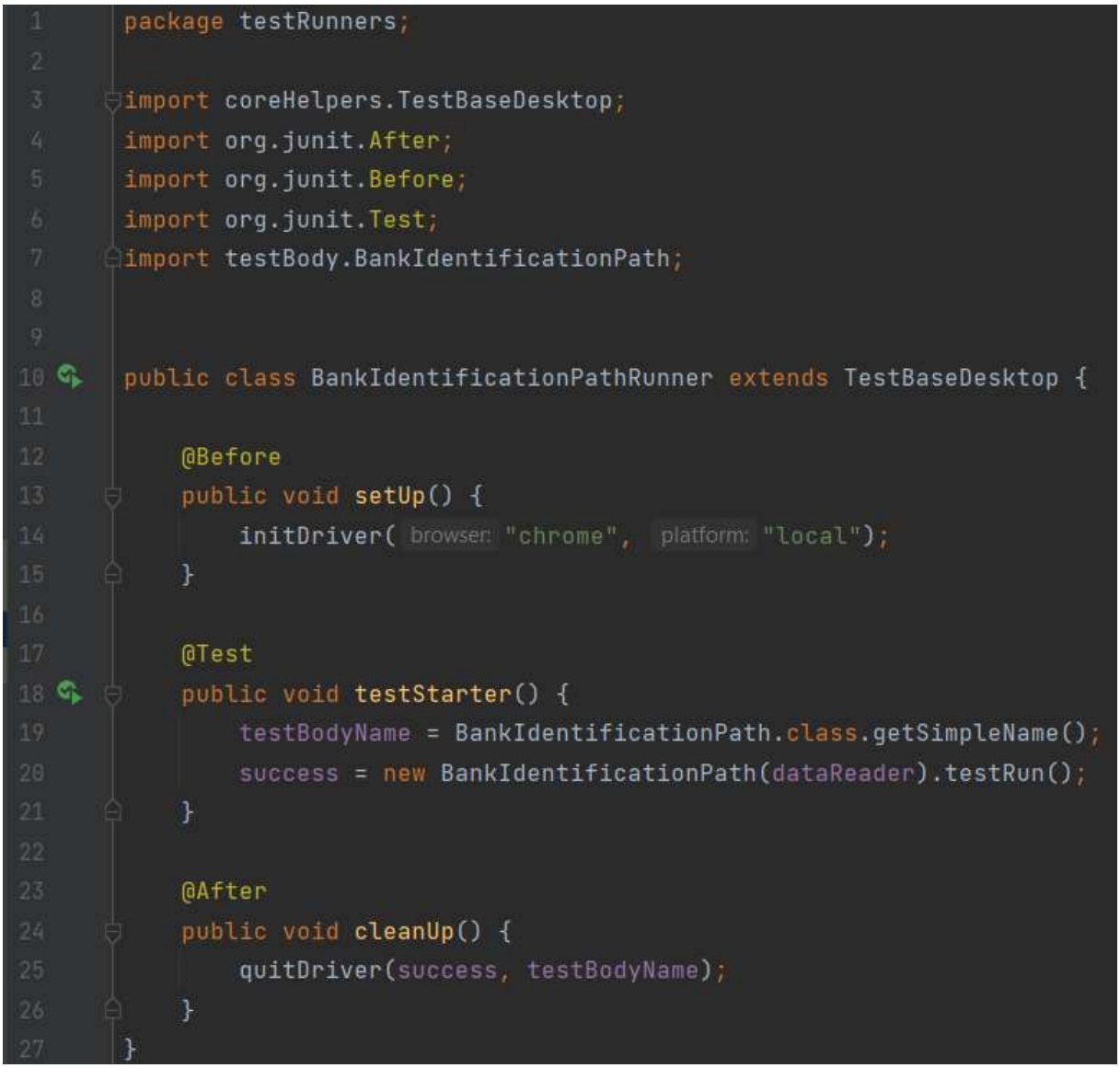
```java
package testBody;

import coreHelpers.DataReader;
import coreHelpers.MethodHelperDesktop;
import testHelpers.FlowHelperGate;

public class BankIdentificationPath extends MethodHelperDesktop {

    public BankIdentificationPath(DataReader dataReader) {
        super(dataReader);
    }

    public boolean testRun() {
        FlowHelperGate flowHelper = new FlowHelperGate(dataReader);
        flowHelper.welcomePageFlow();
        flowHelper.basicInfoPageFlow();
        flowHelper.pinCodePageFlow();
        flowHelper.identificationWayPageFlow( selectedWayButton: "bankIdentificationButton");
        flowHelper.bankPagesFlow();
        flowHelper.personalInfoPageFlow( selectedWay: "bankIdentification");
        return true;
    }
}
```

Příloha D 2 Třída BankIdentificationPath

E Třídy balíčku testRunners

```java
package testRunners;

import coreHelpers.TestBaseDesktop;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import testBody.DocumentsIdentificationPath;


public class DocumentsIdentificationPathRunner extends TestBaseDesktop {

    @Before
    public void setUp() {
        initDriver( browser: "chrome",  platform: "local");
    }

    @Test
    public void testStarter() {
        testBodyName = DocumentsIdentificationPath.class.getSimpleName();
        success = new DocumentsIdentificationPath(dataReader).testRun();
    }

    @After
    public void cleanUp() {
        quitDriver(success, testBodyName);
    }
}
```

Příloha E 1 Třída DocumentsIdentificationPathRunner



```java
package testRunners;

import coreHelpers.TestBaseDesktop;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import testBody.BankIdentificationPath;


public class BankIdentificationPathRunner extends TestBaseDesktop {

    @Before
    public void setUp() {
        initDriver( browser: "chrome",  platform: "local");
    }

    @Test
    public void testStarter() {
        testBodyName = BankIdentificationPath.class.getSimpleName();
        success = new BankIdentificationPath(dataReader).testRun();
    }

    @After
    public void cleanUp() {
        quitDriver(success, testBodyName);
    }
}
```

Příloha E 2 Třída BankIdentificationPathRunner