Porovnání možností frameworků pro automatizované testování mobilních aplikací

Veronika Krajanová

Bakalářská práce
2024

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Veronika Krajanová**
Osobní číslo:	**A21266**
Studijní program:	**B0613A140020 Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Porovnání možností frameworků pro automatizované testování mobilních aplikací**
Téma práce anglicky:	**Comparison of Frameworks for Automated Testing of Mobile Applications**

Zásady pro vypracování

1. Rozepište potřebnou terminologii v kontextu tématu bakalářské práce.
2. Vyberte a popište vhodné technologie pro automatizované testování mobilních aplikací.
3. Navrhněte způsob pro srovnání vybraných technologií.
4. Vyzkoušejte a porovnejte zvolené technologie pro automatizaci testů mobilních aplikací.
5. Výsledky a srovnání vhodně prezentujte a popište.

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. ZAPATA, Belen Cruz a Antonio Hernandez NINIROLA. Testing and Securing Android Studio Applications. Birmigham: Packt Publishing, 2014. ISBN 9781783988808.
2. HANS, Manoj. Appium Essentials [online]. Birmingham: Packt Publishing, 2015. ISBN 978-1-78439-248-2. Dostupné z: https://b4usolution.com/ebooks/Appium-Essentials-manoj-hans.pdf
3. QA Lead. QA Lead [online]. 2023. Dostupné z: https://theqalead.com/tools/best-mobile-test-automation/
4. Appium Documentation. Appium Documentation [online]. 2012. Dostupné z: https://appium.io/docs/en/2.1/
5. Appium vs. Espresso: Which Framework to Use for Automated Android Testing. SmartBear [online]. 2019. Dostupné z: https://smartbear.com/blog/appium-vs-espresso-which-framework-to-use/
6. Getting Started with Espresso : A Complete Guide. HeadSpin [online]. 2022. Dostupné z: https://www.headspin.io/blog/getting-started-with-espresso-a-complete-guide
7. Testování mobilních aplikací – co to je, typy, postupy, přístupy, nástroje a další!. ZAPTEST [online]. Dostupné z: https://www.zaptest.com/cs/testovani-mobilnich-aplikaci-co-to-je-typy-postupy-pristupy-nastroje-a-dalsi
8. XCTest. Apple Developer [online]. Dostupné z: https://developer.apple.com/documentation/xctest

Vedoucí bakalářské práce: **Ing. Petr Žáček, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce: **5. listopadu 2023**
Termín odevzdání bakalářské práce: **13. května 2024**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN
L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 5. ledna 2024

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracovala samostatně a použitou literaturu jsem citovala. V případě publikace výsledků budu uvedena jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 6. 5. 2024 Veronika Krajanová, v.r.

podpis studenta

ABSTRAKT

Bakalářská práce se zabývá analýzou a porovnáním různých frameworků pro automatizované testování mobilních aplikací, se zaměřením na jejich implementaci a efektivitu v různých testovacích scénářích.

V teoretické části jsou představeny základní pojmy a význam testování softwaru, stejně jako detailní popis různých typů mobilních aplikací, jako jsou nativní, webové a hybridní aplikace.

Praktická část se soustředí na srovnání vybraných frameworků, jako je Appium, Espresso, Selendroid, UI Automator a XCTest, přičemž poskytuje podrobný pohled na jejich funkcionalitu, uživatelskou přívětivost a integraci do vývojových prostředí. Autorka zde popisuje postupy a své zkušenosti z jednotlivých frameworků.

Klíčová slova: Testování, automatizované testování, mobilní aplikace, testovací frameworky

ABSTRACT

The bachelor thesis deals with the analysis and comparison of different frameworks for automated testing of mobile applications, focusing on their implementation and effectiveness in different testing scenarios.

The theoretical part introduces the basic concepts and importance of software testing, as well as a detailed description of different types of mobile applications such as native, web and hybrid applications.

The practical part focuses on a comparison of selected frameworks such as Appium, Espresso, Selendroid, UI Automator and XCTest, providing a detailed view of their functionality, user-friendliness and integration into development environments. The author describes the practices and her experiences with each framework.

Keywords: Testing, automated testing, mobile apps, testing frameworks

Chtěla bych poděkovat svému vedoucímu panu Ing. Petru Žáčkovi, Ph.D. za inspiraci k této bakalářské práci, dále za jeho ochotu, cenné rady a připomínky. Dále bych chtěla vyjádřit vděčnost svým kamarádům za jejich podporu a pochopení. Speciální poděkování patří mé mamince, která je pro mě vekou inspirací a oporou.

Prohlašuji, že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Prohlašuji, že při tvorbě této bakalářské práce jsem použila nástroj generativního modelu AI ChatGPT verze 3.5 dostupný na adrese https://chat.openai.com, za účelem úpravy a kontroly textu. Model jsem také použila pro optimalizaci kódů v praktické části. Po použití tohoto nástroje jsem provedla kontrolu obsahu a přebírám za něj plnou zodpovědnost.

OBSAH

ÚVOD

V současné době, kdy mobilní aplikace hrají klíčovou roli v každodenním životě a provozu v různých odvětvích, je nezbytné zajistit jejich kvalitu, spolehlivost a funkčnost, aby splňovaly očekávání uživatelů. Rychlý rozvoj technologií a rozšíření trhu s mobilními aplikacemi zvyšuje potřebu spolehlivých testovacích frameworků.

Tato bakalářská práce se primárně zaměřuje na testovací frameworky pro automatizované testování různých typů mobilních aplikací. Teoretická část obsahuje úvod do testování softwaru a seznámí čtenáře se základními pojmy a důvody, proč je testování softwaru nezbytné. Jelikož se práce zabývá testováním mobilních aplikací, je nutno představit čtenáři i tuto terminologii.

Praktická část přechází k frameworkům pro automatizované testování mobilních aplikací. Frameworky, které tato práce představuje a porovnává jsou Appium, Selendroid, Espresso, UI Automator a XCTest. Každý z nich má své přednosti a používá se za nejvhodnějších podmínek. Pomocí prozkoumání jednotlivých frameworků a testovacích kódů, lze určit, který framework je pro danou situaci vhodný.

Cílem práce je poskytnout srovnání těchto frameworků, osvětlit jejich funkčnost, snadnost použití a integrační schopnosti. V konečném důsledku nasměrovat vývojáře a testery při výběru nejvhodnějšího frameworku pro jejich potřeby.

I. TEORETICKÁ ČÁST

1 TESTOVÁNÍ SOFTWARE

Tato kapitola se věnuje definici a důležitosti testování softwaru, ale také pojmům, které s tímto tématem úzce souvisí. Dále se zabývá chybami, jejich příčinami a náklady s nimi spojenými.

1.1 Základní pojmy, důležitost a definice v testování

Nejprve se budu zabývat definicí, abychom se seznámili s testováním softwaru. Rozhodla jsem se také odpovědět na otázku „Proč testovat?“, jelikož považuji tuto úvahu za nepostradatelnou pro mou práci.

Základní pojmy a definice, které v této podkapitole rozebírám, tvoří hlavní stavební kameny pro porozumění, co vlastně testování softwaru je a k čemu slouží. Je důležité tyto pojmy zmínit a vysvětlit, aby nedošlo k jejich záměně.

1.1.1 Definice testování softwaru

Testování softwaru je jeden z nejdůležitějších procesů pro zajištění kvality softwaru. Pro testování neexistuje jednotná definice, není lehké ho definovat jedním způsobem. Avšak všechny definice vycházejí z podobného základu. Nejčastěji se můžeme setkat s těmito definicemi:

- Testování je proces, který se užívá pro identifikaci bugů a chyb během vytváření softwaru. [1]
- Testování je metoda, která kontroluje, zda software odpovídá požadavkům a zajišťuje, aby neobsahoval chyby. [2]
- Testování je proces hodnocení funkčnosti softwarového systému nebo aplikace.

Hlavním cílem testování je porovnat výsledek skutečné funkce s funkcí očekávanou, snížení nebo odstranění chyb a minimalizaci nákladů jimi způsobené. Zaručuje správné fungování softwaru, který splňuje požadavky zákazníka a neobsahuje žádné závažné defekty. [1], [2]

1.1.2 Důležitost testování softwaru

I když se někomu může zdát, že testování softwaru je zbytečné a zdlouhavé, opak je pravdou. Neotestovaný software může mít velký vliv na několik tisíc uživatelů. Může se jednat o pomalý software, nesprávné funkce nebo dokonce i úplné selhání systému. To vše vede ke zvýšení nákladů na vývoj, a i ke ztrátě reputace. [2]

Lze uvést příklad webové aplikace pro e-shop, která má pomalé načítání a zákazníci jsou netrpěliví, a proto nakonec koupí produkt z jiného e-shopu. Nebo nefunguje správnost tlačítka pro přidání produktu do košíku u některých produktů. Zákazník se po opakovaném snažení nakonec rozhodne produkt nekoupit. To vše vede ke snížení zisků našeho klienta a následně ke ztrátě reputace. [3]

Pro zamezení těchto chyb je důležité správně testovat. Správné testování spočívá v tom, že se začíná s testováním co nejdříve to jde, ideálně od samého počátku vývoje softwaru.

1.1.3 Přesnost a správnost

Nejlépe to vysvětlím na příkladu aplikace pro výpočet průměrného věku lidí ve skupině. Přesnost v tomto případě by znamenala přesný výsledek průměru zadaných čísel. Zadané čísla jsou 10, 20, 30 a očekávaný přesný výsledek, což je 20. Pokud by aplikace vrátila 20,5, znamenalo by to, že výsledek je nepřesný, jelikož má odchylku 0,5 od přesného výsledku.

Správnost by se v tomto případě zaměřovala na správnost výpočtu průměru. Máme například hodnoty 10, 20, 30 a přesný průměr by byl 20. Aplikace nám zobrazuje výsledek průměru 20,01, což je správný výsledek, jelikož se přibližuje hodnotě 20. Pokud by byl výsledek průměru 60, znamená to, že aplikace pracuje nesprávně, jelikož použila pouze součet bez dělení počtem čísel. [3][4]

Rozdíl mezi těmito pojmy přesnost a správnost je:

Přesnost se týká toho, jak přesně aplikace poskytuje numerický výsledek, zatímco **správnost** se týká toho, zda aplikace provádí správný výpočet na základě zadaných čísel.

1.1.4 Verifikace a validace

Tyto dva pojmy jsou velice často zaměňované. Nyní tedy vysvětlím jejich správný význam v testování a uvedu příklad.

Verifikace, jiným slovem ověřování, je proces, který kontroluje, zda software splňuje specifikace neboli návrh. Jednoduše řečeno, zda jsme vyrobili to, co jsme vyrobit chtěli (vytvořili jsme náš software správně). [5], [6], [7]

Validace ověřuje, zda software splňuje požadavky a funguje podle očekávání zákazníka. Ověřujeme tedy, že vyrábíme to, co máme (vytváříme správný produkt). Pomáhá najít problémy, které mohou bránit výkonu softwaru. **Chyba! Nenalezen zdroj odkazů.**[5],[6],[7]

Pro lepší pochopení uvedu jako příklad webovou aplikaci pro rezervaci hotelu. Ve fázi verifikace se provádí testování, aby bylo ověřeno, že všechna tlačítka a odkazy vedou na správné stránky, přesně tak, jak bylo uvedeno ve specifikaci. Po dokončení vývoje webové aplikace následuje validace, která se týká testování na reálných uživatelích. Uživatelé zkouší různé úkony, jako je třeba vyhledávání, výběr hotelu atd. a díky jejich zpětné vazbě je zjištěno, zda je aplikace intuitivní a pracuje rychle dle požadavků zákazníka.

1.1.5 Kvalita a spolehlivost

Kvalita a spolehlivost jsou příbuzné pojmy, ale i přesto je mezi nimi rozdíl.

Kvalita určuje, jak moc dobře software plní svou funkci. Spadá pod ni několik vlastností. Označuje míru, do jaké software odpovídá specifikacím, návrhu a požadavkům. Můžu se zeptat otázkou: Plní produkt zamýšlenou funkci? Pokud ano, tak jak dobře? [8],[9]

Spolehlivost na rozdíl od kvality určuje stabilitu softwaru. U spolehlivého softwaru je očekáváno, že bude plnit své funkce správně a bez chyb i přes nepříznivé podmínky (např. vyšší zátěž). Pro určení spolehlivosti se sleduje také jaké jsou možnosti obnovení provozu, zotavení se z výpadku a kolik času je k tomu potřeba. [8], [9]

Pro měření kvality se používá model **FURPS** vytvořen společností Hewlett-Packard. Vysvětlím zde, co zkratka představuje a definuji tyto pojmy. [10][10]

Tento model se skládá z následujících pěti částí:

- **F – Functionality** (Funkčnost): Popisuje hlavní funkce nebo požadavky, které musíme důkladně otestovat, jelikož jsou primární vlastností softwaru. [10]
- **U – Usability** (Použitelnost): Zaměřuje se na vzhled, ovládání a jak snadné je naši aplikaci používat. Důležité aspekty naší aplikace jsou intuice a jednoduchost ovládání, aby nebyl koncový uživatel zmatený a používání aplikace bylo pohodlné. [10]
- **R – Reliability** (Spolehlivost): Vysvětleno výše na začátku kapitoly. Spolehlivost, stejně jako ostatní zde uvedené pojmy, spadá pod kvalitu. [10]
- **P – Performance** (Výkon): Hodnocení výkonu a efektivity. Zákazník má své požadavky na rychlost odezvy systému a dobu zpracování požadavků. Například doba pro zapnutí aplikace a odezvu akce. Často bývá rozhodujícím faktorem při výběru mezi námi vyvíjeným produktem a konkurenčním. [10]
- **S – Supportability** (Podporovatelnost): Schopnost podpory a údržby systému. Důležité je také, aby bylo možné systém upgradovat, měnit nebo opravit. [10][11]

Metoda FURPS později nebyla dostačující, proto se později rozšířila o další části a dostala název **FURPS+**. Najdeme mnoho různých rozdělení a definicí. Já vybrala následující čtyři kategorie [11][12]:

- **Design Requirement** (požadavky na návrh): Stanovuje se omezení návrhu. Určuje se například typ databázového systému a strukturu databáze.
- **Implementation Requirment** (požadavky na implantaci): Zahrnuje standardy kódování, jazyk programování a platformu.
- **Interface Requirement** (požadavky na rozhraní): Externí zařízení, se kterým musí systém interagovat.
- **Physical Requirement** (požadavky na fyzické vlastnosti): Definuje, na jakém hardwaru bude systém spouštěn.

1.2 Chyba

Záměrem testování je identifikovat chyby, nedostatky, nebo chybějící požadavky z již schváleného návrhu. Nedokážeme jednotnou definicí popsat, co to vlastně chyba je. Existuje mnoho dalších slov, kterými lze chybu označit a jejich přesná definice není ve všech firmách stejná. V anglickém jazyce se používají slova jako je bug, error, mistake, fault. [15][13]

Zde je uveden příklad rozdílů mezi těmito slovy. V této práci je nebudu rozlišovat a budu používat obecně slovo chyba.

Tabulka 1. Bug vs fault vs error vs mistake

Název	**Definice**
Bug	Bug se týká defektů, což znamená, že softwarový produkt nebo aplikace nefungují podle stanovených požadavků. [12][14]
Error	Error je chyba v kódu, kvůli které se nezdaří kompilace nebo spuštění. **Chyba! Nenalezen zdroj odkazů.**[14]
Fault	Fault je stav, který způsobuje selhání softwaru, a ten nedosahuje své potřebné funkce. [14]
Mistake	Mistake je lidská chyba, která vede ke stavu fault. [14]

Vysvětlení pojmu chyba lze docílit díky specifikaci softwaru, který je vyvíjen. Specifikace je dohoda mezi vývojářským týmem. Definuje produkt, který vytvářejí, popisuje, co to bude, jak se bude chovat nebo nechovat a co udělá nebo neudělá. Dohoda může být pouze verbální mezi spolupracovníky nebo formální písemný dokument, to už záleží na samostatné firmě. [15][15]

Za chybu lze považovat dle R. Patton ([15], s. 15), pokud splníme alespoň jedno z následujících kritérií:

1. Software nedělá něco, co by podle specifikace produktu dělat měl.

2. Software dělá něco, co by podle specifikace produktu dělat neměl.

3. Software dělá něco, co specifikace produktu nezmiňuje.

4. Software nedělá něco, o čem se specifikace nezmiňuje, ale měla by se zmiňovat.

5. Software je nesrozumitelný, obtížně se s ním pracuje, je pomalý nebo – v očích testera – bude koncovými uživateli vnímán jako prostě nevyhovující.[1] (přeloženo autorkou bakalářské práce)

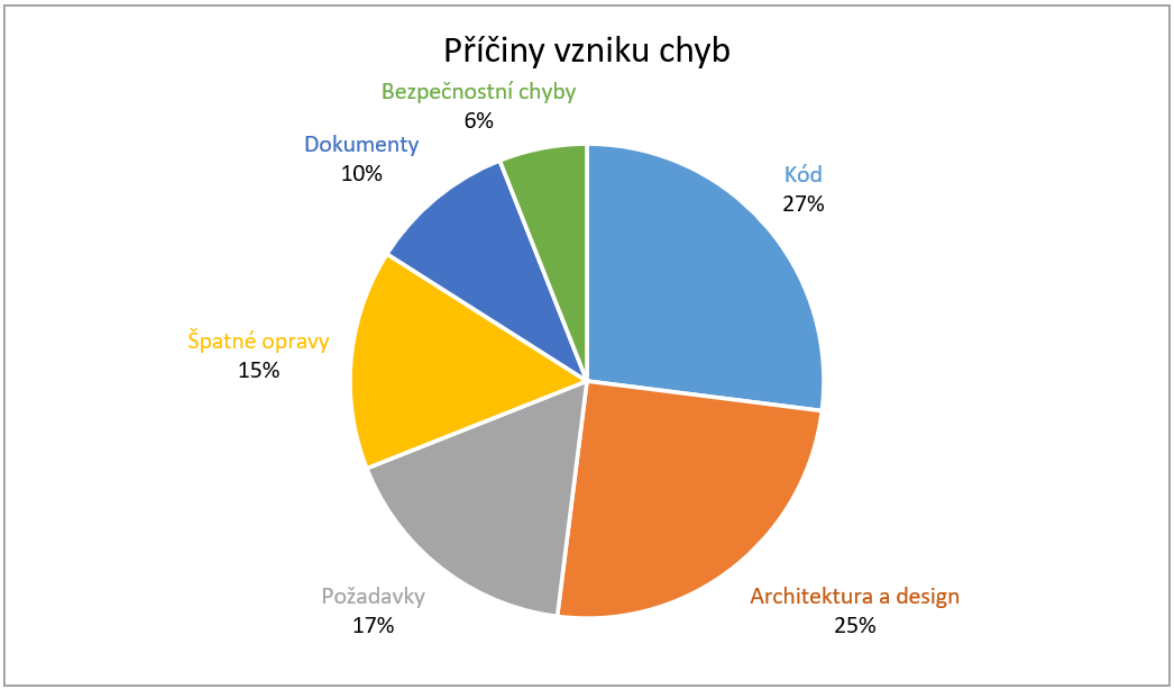
1.2.1 Příčiny vzniku chyb

Teď když bylo vysvětleno, co je to chyba, můžu se věnovat tomu, jak vlastně takové chyby vznikají. Většina lidí by předpokládala, že nejvíce chyb je způsobena chybným kódem, což je dnešní době správné tvrzení. Ovšem ne vždy tomu tak bylo. Vzhledem k technickému pokroku se za posledních několik let zlepšila hardwarová (výpočetní výkon díky modernějším CPU) i softwarová část testování (umělá inteligence). Dnes se pracuje na mnohem složitějších systémech než dříve, což může vést k většímu počtu chyb v kódu.

Z obrázku (Obrázek 1) lze vyčíst, jaké příčiny chyb se v roce 2018 vyskytovaly a jak často. Typ příčiny, který se nejvíce vyskytoval, byl kód, což souhlasí s odhadem pro novodobé testování, tak jak jsem uvedla na začátku kapitoly.

---

[1] 1. The software doesn't do something that the product specification says it should do.
2. The software does something that the product specification says it shouldn't do.
3. The software does something that the product specification doesn't mention.
4. The software doesn't do something that the product specification doesn't mention but should.
5. The software is difficult to understand, hard to use, slow, or—in the software tester's eyes—will be viewed by the end user as just plain not right.

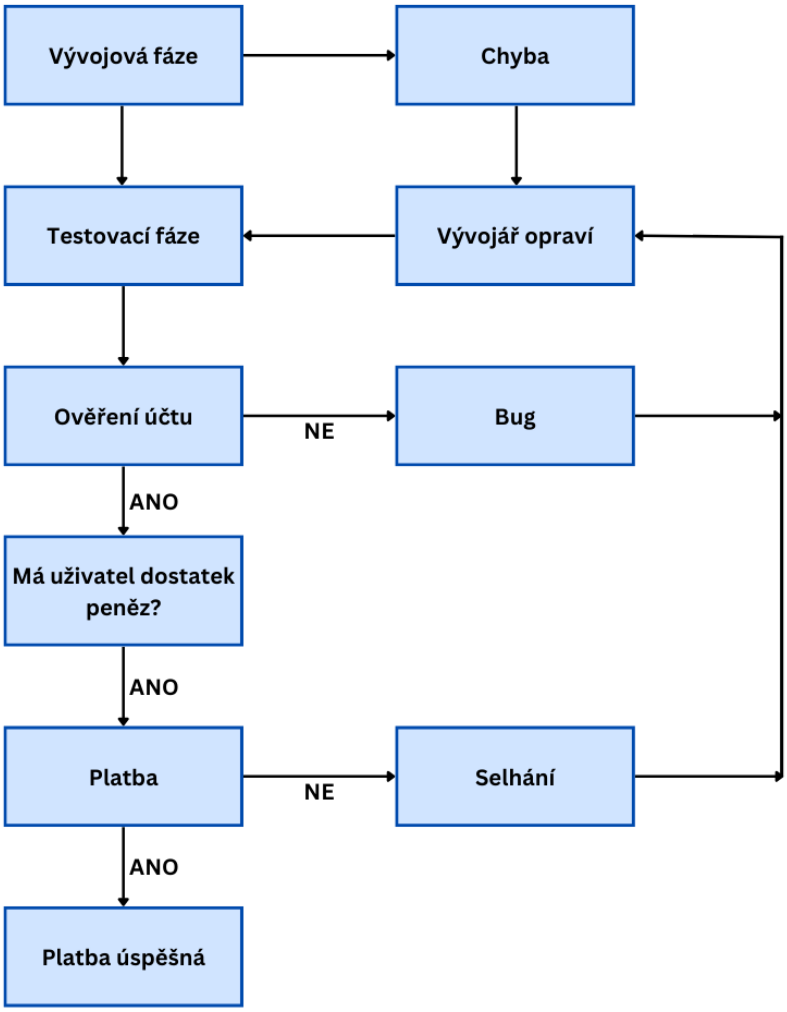


Obrázek 1. Příčiny vzniku chyb v roce 2018 [16]

Příčina chyby (defektu) je obecně **omyl**, který je přirozenou součástí procesu vývoje softwaru. Může se jednat o lidský omyl nebo také i chybu v hardwarové či softwarové části. To má zá následek **defekt**, což je chyba nebo bug. Chyba se projeví například vypsáním chybové hlášky. Bug se může projevit neočekávaným chováním aplikace. [17] Následek těchto akcí se je **selhání**, které znamená že systém nefunguje, tak jak by měl. V některých případech může dojít i k pádu systému. Všechny tyto situace spolu souvisí. Na obrázku (Obrázek 2) můžeme jasně vidět vztah mezi nimi pomocí implikace. [20][18]

Obrázek 2. Omyl, defekt, selhání

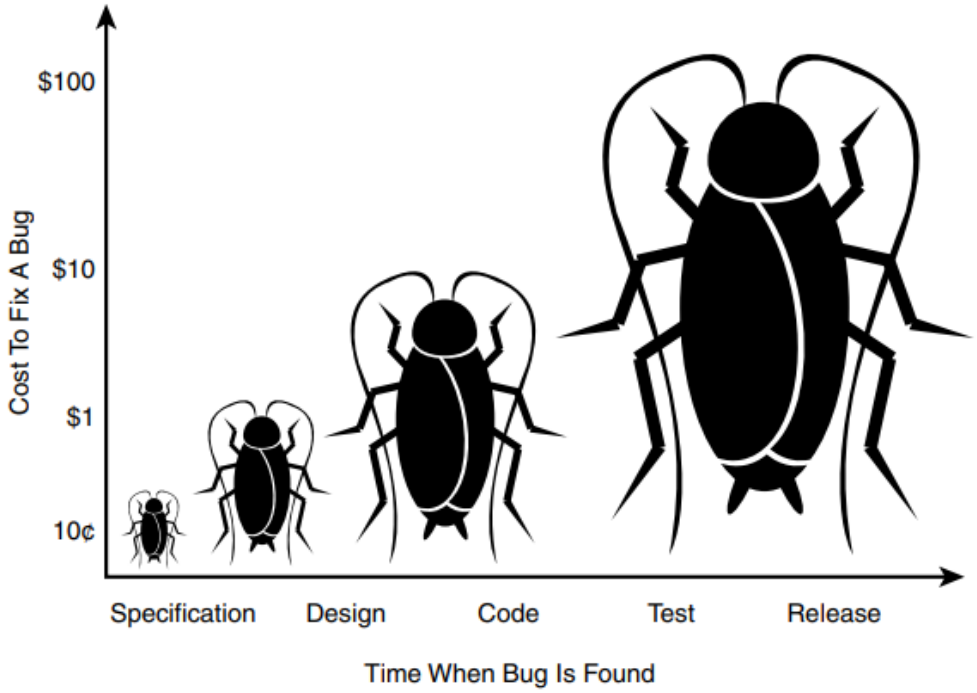
Pro jasnější pochopení uvedu příklad aplikace bankovnictví (Obrázek 3). Začíná postupně vývojem aplikace, testováním a dále pokusem o přihlášení na správný účet, na kterém je dostatek peněz. Poté následuje pokus o provedení platby. Pokud je vše v pořádku, platba proběhne úspěšně. Při objevení nesrovnalostí v různých fázích se vždy obrátíme na vývojáře, který bude mít za úkol aplikaci opravit tak, aby co nejméně docházelo k její selhání.



Obrázek 3. Příklad chyby, bugu, selhání v bankovnictví

1.2.2 Náklady na odstranění chyb

S testováním by se mělo začít co nejdříve, nejlépe hned od samotného začátku. Důvodem je zvyšující se cena opravy. Čím později zjistím chybu, tím bude dražší a náročnější ji opravit. Když zákazník zjistí chybu na začátku psaní specifikace, nebude chyba stát skoro nic. Naopak když se tatáž chyba najde v pozdější části, kdy už se kóduje, bude stát až stonásobně více, jak je uvedeno na obrázku (Obrázek 4). Nechceme, aby zákazník objevil chybu až po uvedení programu do provozu. Cena tehdy už bude vysoká a může přesahovat až 100$.



Obrázek 4. Graf náklady na opravu chyby v závislosti na fázi vývoje [15]

V potaz musí být brán také dominový efekt. Podle slovníku cizích slov ABZ [20][19] „*se jedná o řetězovou reakci vzájemně se ovlivňujících příčin a následků, která je často spuštěna i malým podnětem*“. Může se tedy stát, že při zjištění jedné malé chyby, nebude stačit opravit jen ji, ale zároveň bude potřeba provést další změny v kódu. [20]

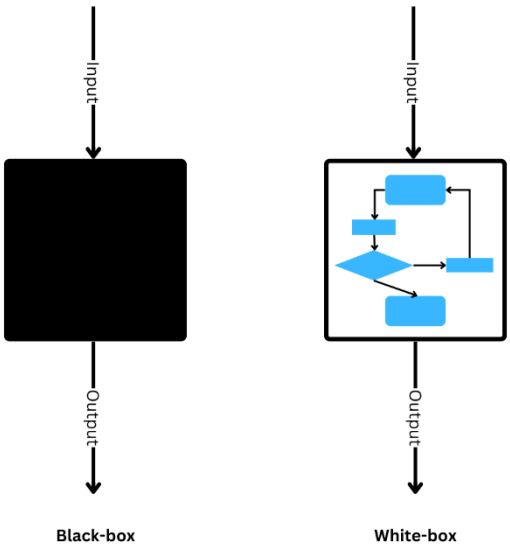
1.3 Testovací techniky

Tato kapitola se bude věnovat technikám testování, které se liší podle úrovně dostupných informací. Tento typ klasifikace rozděluje testování do dvou hlavních kategorií: black-box a white-box.

1.3.1 Black-box a white-box

Black-box (černá skříňka) se zabývá pouze vstupem a výstupem aplikace. Znamená to, že tester nemá přímý přístup ke kódu a nezná vnitřní logiku. Laicky to můžu vysvětlit na příkladu zhasnutí světla. Když zhasnu světlo v boxu, neuvidím, co se v něm nachází a bude v něm pouze černo. Testování se provádí z pohledu uživatele, kde se testují pouze různé vstupy a probíhá kontrola, zda výstupy splňují očekávaný výsledek. Z toho vyplývá, že smyslem testů black-box je ověřit uživatelské požadavky. [21]

Naopak **white-box** (bílá skříňka) pro testera znamená, že má přístup ke všem informacím včetně kódu a vnitřní logiky aplikace. Stejný příklad můžu použít i zde, akorát v situaci, kdy mám světlo rozsvícené. Rozsvícené světlo v boxu poskytuje pohled na všechny objekty a jejich rozmístění. Při této situaci se testuje většinou přímo zdrojový kód, díky němuž lze otestovat i situace, které z pohledu uživatele nemusí být tak znatelné. [21]



Obrázek 5. Black-box a white-box

2 AUTOMATIZOVANÉ TESTOVÁNÍ

V tuto chvíli už jsou známé všechny důležité pojmy pro pokračování v mé práci. Máme představu o tom, co testování zahrnuje a jak je důležité. Nyní se tedy budu věnovat automatizovanému testování a poté navážu na hlavní část, kde seznámím čtenáře s testováním mobilních aplikací.

2.1 Automatizované versus manuální testování

Automatizované testování, jak už jeho název napovídá, automatizuje manuální testování. K tomuto procesu slouží automatizované softwarové nástroje. [22] *Nástroje pro testování softwaru nejsou náhradou za testery softwaru - pouze pomáhají testerům softwaru lépe vykonávat jejich práci.*[2] R. Patton (přeloženo autorkou práce z [15], s. 221)

Manuální testování provádí tester ručně, bez použití softwarových nástrojů a skriptů. V roli koncového uživatele testuje a identifikuje chyby v oblasti jako je funkčnost, použitelnost nebo výkon. Jedním příkladem je ověření, zda odkazy na webové stránce správně přesměrovávají pomocí interaktivního prokliku. [23]

2.1.1 Výhody a nevýhody automatizovaného testování

Zde uvedu hlavní příklady výhod a nevýhod automatizovaného testování.

Výhody:

- **Rychlejší provádění testů.** Probíhají souběžně, takže se za krátký časový úsek spustí více testů, než kdyby probíhali sériově. [24], [25]
- **Snížení nákladů.** Zkrátí se potřebný čas na provádění testů. Mohou také odhalit chybu už v dřívější fázi vývoje aplikace. [24], [25]
- **Spolehlivější výsledky.** Testy se provádí ve větším počtu a eliminuje se lidská chyba. [24]
- **Efektivita je vyšší.** Díky automatizaci nemusí tester vše provádět ručně. [24], [26]

[2] Software test tools aren't a substitute for software testers—they just help software testers perform their jobs better.

Nevýhody:

- **Složitost**. Vývoj automatizovaných testů může trvat déle než vývoj manuálních.
- **Vysoká cena při zavedení do provozu.** [24]
- **Neotestujeme grafické rozhraní a user experience.** [24], [25]
- **Pro malé aplikace je neefektivní.** [25]

2.1.2 Výhody a nevýhody manuálního testování

Nyní se zaměřím na výhody a nevýhody manuálního testování.

Výhody:

- **Flexibilita.** Lze vyzkoušet různé scénáře, které nás mohou napadnout pouze při manuálním testování. [27]
- **Můžeme otestovat grafické rozhraní a user experience.**
- **Efektivní pro malé aplikace.** [25]

Nevýhody:

- **Vyšší náklady než u automatizovaného testování.** Je časově a pracovně náročné, je nutno platit například více testerů. [27]
- **Monotónnost může vést k chybám.** [27]
- **Neefektivní pro náročnější aplikace.** [25]

2.2 Testování mobilních aplikací

Přesouvám se k hlavní části, která se týká mobilních aplikací a rozdělení podle typů.

Mobilní aplikace je druh aplikace, která je určena pro použití na chytrých telefonech, tabletech, počítačích a dalších elektronických zařízení. [28], [29]

Existuje velká škála typů mobilních aplikací, avšak jejich vývoj musí vždy začít výběrem jednoho ze tří hlavních typů. Jedná se o nativní, webový a hybridní typ mobilní aplikace. Výběr z těchto tří zmíněných druhů závisí na požadavcích zákazníka, speciálních funkcích, určení cílové skupiny a celkovém průzkumu. V následujících podkapitolách hlavní tři druhy mobilních aplikací představím podrobněji. [30], [31]

2.2.1 Nativní aplikace

Jak už název napovídá, nativní aplikace jsou napsány v programovacím jazyce, který je nativní (*nativní mobilní aplikace - aplikace vytvořené přímo pro prostředí mobilního zařízení* [31][32]) **pro danou platformu** (Android, iOS, atd.) Pro Android se nejčastěji používají jazyky Java nebo Kotlin, zatímco pro iOS je to Objective-C nebo Swift. Znamená to tedy, že aplikace vytvořené pro Android jsou distribuovány pouze na nativní distribuční službě Google Play a aplikace pro Apple na App Store. [31], [33]

Pro zjednodušení vývoje lze použít například open-source framework Flutter, který umožňuje napsat pouze jeden zdrojový kód v jazyce Dart. Tento kód poté bude fungovat na různých platformách a ušetří vývojáři čas a usnadní vývoj. [34] Dále se můžeme setkat s platformou Xamarin nebo s frameworkem React Native. [35], [36]

2.2.2 Webová aplikace

Webová aplikace je **webová stránka**, která působí **jako aplikace** v telefonu. Tento typ aplikace se nestahuje a neinstaluje do mobilního zařízení, ale otevírá se pomocí prohlížeče (Google, Safari, atd.). Všechna data aplikace se ukládají online, proto pro využití webové aplikace je zapotřebí přístupu k internetu. Pro vývoj se nejčastěji používají technologie HTML, CSS, JavaScript a mezi nejznámější frameworky patří například Ionic nebo Framework7. [31]

2.2.3 Hybridní aplikace

Kombinace nativní a webové aplikace se nazývá hybridní. U tohoto typu je zapotřebí stáhnout aplikaci do mobilního zařízení stejně jako u typu nativního. To znamená, že část aplikace je napsána pomocí webových technologií, jako jsou HTML, CSS a JavaScript, zatímco zbytek aplikace je napsán pomocí nativního kódu pro určitou platformu, jako je iOS nebo Android. [37][38]

Webová část vytváří uživatelské rozhraní a provádí interakci s uživatelem. Tato část je vytvořena stylem webové stránky nebo webové aplikace, která je zabalena v nativním kontejneru (Ionic). [38], [39]

Nativní kontejner obaluje webový obsah do nativní aplikace. Díky tomu umožňuje přístup k funkcím platformy. Pro vytvoření kontejneru se používá například framework Ionic nebo React Native. [37], [39]

2.2.4 Výhody a nevýhody jednotlivých typů mobilních aplikací

Tabulka 2. Výhody a nevýhody jednotlivých typů mobilních aplikací [30], [31]

Typ	Výhody	Nevýhody
Nativní aplikace	• Efektivita díky přístupu ke specifickým funkcím platformy (fotoaparát, notifikace, GPS …) • Není nutné připojení k internetu • Působivé uživatelské prostředí na Android nebo iOS • Nejlépe responzivní	• Vyšší náklady na vývoj • Údržba může být komplikovaná
Webová aplikace	• Rychlejší vývoj než u nativních aplikací • Jednoduchá údržba • Nižší náklady na vývoj • Není třeba distribuovat software na zařízení • Aktualizace aplikace a okamžitá dostupnost	• Pomalejší výkon než nativní • Nutné připojení k internetu • Není optimalizováno pro danou platformu • Nemá přístup k funkcím platformy • Lze přistupovat pouze k webovým funkcím
Hybridní aplikace	• Jednoduchá údržba • Nižší náklady na vývoj • Zaměření na více platforem	• Pomalejší výkon než nativní • Nelze přistupovat ke všem funkcím platformy • Není optimalizován pro platformu

II. PRAKTICKÁ ČÁST

3 AUTOMATIZOVANÉ FRAMEWORKY

Praktická část této bakalářské práce se zaměřuje na frameworky pro automatizaci testování mobilních aplikací, což jsou nástroje, které automatizují testování mobilních aplikací. Tyto frameworky poskytují platformu pro vytváření, spouštění a správu automatizovaných testů mobilních aplikací a zajišťují, aby aplikace fungovaly dle očekávání. V následujících podkapitolách bude popis vybraných frameworků. [40][40]

V poslední podkapitole Nástroje pro kontinuální integraci vysvětlím, co je to kontinuální integrace a podrobněji představím nástroj Jenkins.

3.1 Popis automatizovaných frameworků

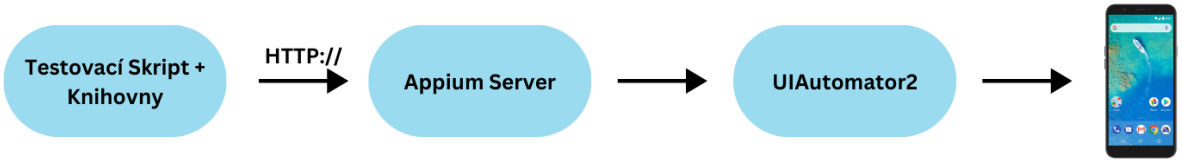
Jak již bylo uvedeno, nyní se kapitola bude věnovat popisu jednotlivých frameworků. Pro tuto práci jsem vybrala nejčastěji zmiňované a používané frameworky pro platformu Android nebo i iOS.

3.1.1 Appium

Díky své jednoduchosti, dostupnosti, podpory jazyků a univerzálnosti se jedná o nejznámější a nejpoužívanější open-source framework vytvořen komunitou vývojářů. Umožňuje black-box testování nativních, webových i hybridních aplikací pro Android i iOS zařízení. [41]

Appium využívá standardní specifikaci **WebDriver** jako své rozhraní API pro automatizaci. Tato specifikace poskytuje základní funkce a operace, které zahrnují vyhledávání prvků, interakci s prvky a načítání stránek, jsou použitelné na různých typech zařízení a platformách. [41][42]

Appium implementuje tento standard pomocí ovladačů (drivers). Mezi oficiální, které používá Appium tým patří například UIAutomator2, Espresso, Chromium. [43] Ovladače jsou zásuvné moduly, které dávají frameworku Appium schopnost komunikovat s danou platformou a implementovat požadované chování. Díky těmto ovladačům může Appium pracovat s různými zařízeními a aplikacemi a umožňuje psát jednotné programy pro jejich ovládání. Takže pokud chceme napsat program, který ovládá jakýkoli typ aplikace na telefonu, tabletu nebo dokonce i televizi, můžeme použít Appium. [44]



Obrázek 6. Architektura Appium

Spuštění Appium frameworku a testů je podrobně vysvětleno v kapitole Testování aplikací, kde se tomu dopodrobna věnuji.

3.1.2 Espresso

Espresso je open-source framework vyvinutý společností Google pro psaní testů uživatelského rozhraní platformy Android. Je tedy určený primárně pro testování nativních aplikací. Ovšem některé jeho funkce mohou být využity i pro testování hybridních aplikací, které obsahují WebView prvky. Pro testování webových aplikací se nepoužívá, jelikož existují vhodnější nástroje. [45][46]

Espresso obsahuje velké množství tříd pro testování. Lze je rozdělit do pěti kategorií, které uvedu a popíšu níže.

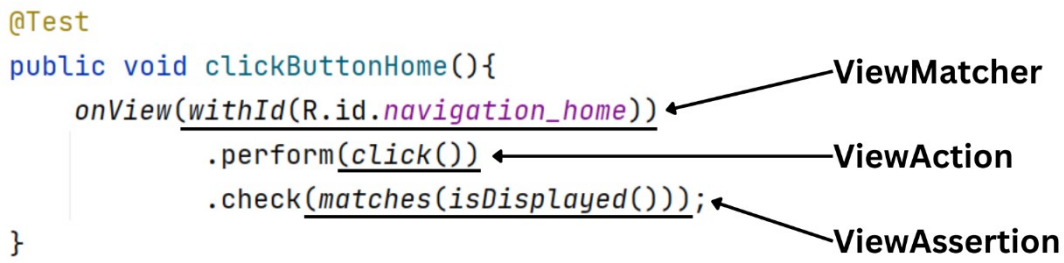
JUnit runner – poskytuje runner *AndroidJUnitRunner* pro spouštění testovacích případů espresso, které jsou napsány ve JUnit3 a JUnit4. Je specifický pro aplikace pro Android a zpracovává načítání testovacích případů Espresso a vybranou testovací aplikace v reálném zařízení nebo v emulátoru, provádí testovací případy a vyhodnocuje jejich výsledky. Implementace se provádí pomocí *@RunWith*. [47]

JUnit rules – poskytuje pravidlo *ActivityTestRule* pro spuštění aktivity Androidu (obrazovka aplikace, část uživatelského rozhraní) před tím, než se značně provádět testovací případ. Tuto aktivitu spustí před každou metodou (testem) začínající *@Test* a *@Before*. Po provedení metody aktivitu ukončí pomocí *@After*. To zajistí, že každý test probíhá v izolovaném prostředí, tím se snižuje pravděpodobnost, že se testy vzájemně ovlivní nebo naruší. [47]

ViewMatchers – Metoda *onView()* očekává argument typu *ViewMatcher*, který slouží k vyhledání odpovídajícího pohledu v uživatelském rozhraní. Ten poté vrátí objekt *ViewInteraction*, který umožňuje provádět různé akce s tímto pohledem, jako je kliknutí nebo ověření jeho stavu. [47][48]

ViewActions – Jakmile *onView()* odpovídá a vrací objekt *ViewInteraction*, lze vyvolat libovolnou akci zavoláním metody *perform()* objektu *ViewInteraction* a předat mu příslušné akce jako například kliknutí. [47][48]

ViewAssertions – Je to kolekce objektů, které lze použít po zavolání metody *onView*, jež nám vrátí objekt *ViewInteraction*, k ověření aktuálního stavu pomocí metody *check()*. Nejčastěji se používá *matches()*, který využívá *ViewMatcher* k zjištění, zda se shoduje aktuální stav s očekávaným. [47][48]



Obrázek 7. Ukázka zdrojového Java kódu pro testování tlačítka navigation_home ve frameworku Espresso

V následujícím testování pomocí frameworku Espresso je využíván nástroj Android Studio, ve kterém je vytvořen základní projekt se třemi tlačítky. Netestuji tedy .apk soubor, ale testuji rovnou ve vývojářském projektu aplikace (white-box). To je jeden z hlavních rozdílů mezi ostatními frameworky. Základ je přidání správných konfigurací v souboru *build.gradle.kts* (Obrázek 8. Konfigurace souboru *build.gradle.kts* pro Espreeso). Následně je přidána vzorová ukázka (Obrázek 9. Ukázka zdrojového kódu (Java) s využitím frameworku Espresso) pro testování tlačítek a správného zobrazení. Ukázka obsahuje také funkci pro přidání čekání *waitFor()*.

```
plugins {
    alias(libs.plugins.androidApplication)
}

android {
    namespace = "com.example.homebuttontest_espresso"
    compileSdk = 34

    defaultConfig {
        applicationId = "com.example.homebuttontest_espresso"
        minSdk = 21
        targetSdk = 34
        versionCode = 1
        versionName = "1.0"

        testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            isMinifyEnabled = false
```

```
            proguardFiles(
                getDefaultProguardFile("proguard-android-optimize.txt"),
                "proguard-rules.pro"
            )
        }
    }
    compileOptions {
        sourceCompatibility = JavaVersion.VERSION_1_8
        targetCompatibility = JavaVersion.VERSION_1_8
    }
    buildFeatures {
        viewBinding = true
    }
}

dependencies {

    implementation(libs.appcompat)
    implementation(libs.material)
    implementation(libs.constraintlayout)
    implementation(libs.lifecycle.livedata.ktx)
    implementation(libs.lifecycle.viewmodel.ktx)
    implementation(libs.navigation.fragment)
    implementation(libs.navigation.ui)
    testImplementation(libs.junit)
    androidTestImplementation(libs.ext.junit)
    androidTestImplementation(libs.espresso.core)
    androidTestImplementation ("androidx.test.espresso:espresso-accessibility:3.5.1")
}
```

Obrázek 8. Konfigurace souboru *build.gradle.kts* pro Espreeso

```
@RunWith(AndroidJUnit4.class)
 public class TC_E_01 {
     @Rule
     public ActivityScenarioRule<MainActivity> activityScenarioRule = new Activi
tyScenarioRule<>(MainActivity.class);

     @Test
     public void clickButtonHome() {
         onView(withId(R.id.navigation_home)).perform(waitFor(2000));
         onView(withId(R.id.navigation_home)).perform(click()).check(matches(isDis
played()));
     }

     @Test
     public void clickButtonDashboard() {
         onView(withId(R.id.navigation_dashboard)).perform(waitFor(2000));
         onView(withId(R.id.navigation_dashboard)).perform(waitFor(2000),click());
         onView(ViewMatchers.withId(R.id.text_dashboard)).check(matches(with
        Text("This is dashboard fragment")));
         onView(isRoot()).perform(ViewActions.pressBack());
         onView(withId(R.id.text_home)).perform(waitFor(5000)).check(matches(isDis
        played()));
     }

     @Test
     public void clickButtonNotifications() {
         onView(withId(R.id.navigation_notifications)).perform(waitFor(2000));
         onView(withId(R.id.navigation_notifications)).perform(click());
         onView(ViewMatchers.withId(R.id.text_notifications)).check(matches(with
        Text("This is notifications fragment")));
     }

public static ViewAction waitFor(final long millis) {
    return new ViewAction() {
        @Override
        public Matcher<View> getConstraints() {
            return isDisplayed();
```

```
33.             }
34.
35.             @Override
36.             public String getDescription() {
37.                 return "wait for " + millis + " milliseconds";
38.             }
39.
40.             @Override
41.             public void perform(UiController uiController, View view) {
42.                 uiController.loopMainThreadForAtLeast(millis);
43.             }
44.         };
45.     }
46. }
```

Obrázek 9. Ukázka zdrojového kódu (Java) s využitím frameworku Espresso

3.1.3 Selendroid

Opět se jedná o open-source framework, který je postaven na testování uživatelského rozhraní mobilních aplikací. Umožňuje testování nativních (Android), hybridních ale také webových aplikací. Pro hybridní a webové aplikace se spíše volí možnost využití frameworku Appium, který jsem představila už v předchozí části. Selendroid používá přístup k více vláknům, umožňující **paralelní provádění testů**. Každý test má své vlastní vlákno, což znamená, že testy na různých zařízeních mohou běžet současně. [49]

Hlavní využití tohoto frameworku je zpětná kompatibilita. Selendroid podporuje Android ve verzích 2.3.3 až Android 4.4, zatímco Appium podporuje pouze Android 4.3 a vyšší. [50]

Dříve byl tento framework používaný, avšak s rostoucí popularitou jiných frameworků jako je Appium nebo Espresso, byl podle posledních dostupných informací **vývoj Selendroidu zastaven.** [51][52], [53]

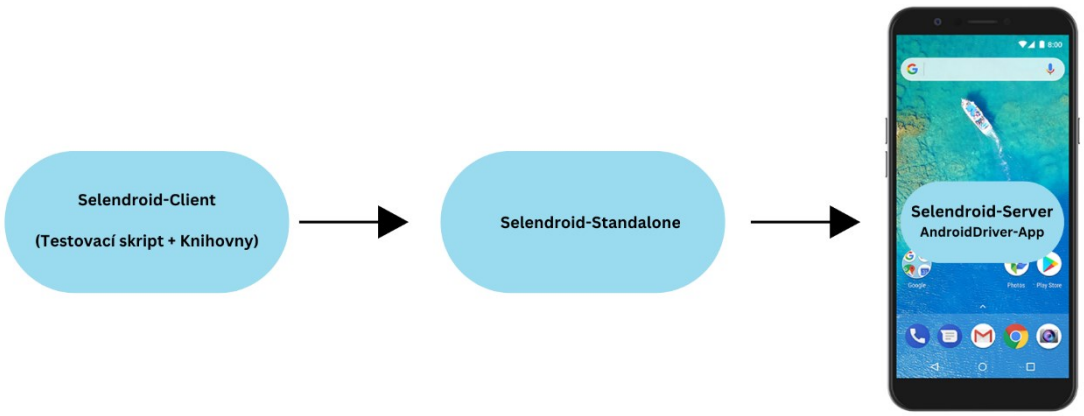
Selendroid funguje na architektuře klient-server a skládá se ze čtyř hlavních komponent:

Selendroid-Client – Jedná se o knihovnu, napsanou v jazyce Java, která komunikuje se serverem. [54]

Selendroid-Standalone – Tento ovladač spravuje různá zařízení a také soubor .apk. Funguje jako proxy server mezi Selendroid-Client (testovací kód) a Selendroid-Server (běžící na zařízení). To znamená, že veškeré požadavky a příkazy od testovacího kódu jsou předány prostřednictvím Selendroid-Standalone ovladače na Selendroid-Server a naopak. [54]

Selendroid-Server – Tento server běží v testované aplikaci na zařízení nebo na emulátoru se systémem Android. Jedná se o hlavní komponentu architektury Selendroid. [49][54]

AndroidDriver-App – Je to vestavěná aplikace integrována v Selendroid-Server, který běží na testovaném zařízení s Androidem. Je určena k testování mobilních webů. [54]



Obrázek 10. Architektura Selendroid

Pro spuštění Selendroidu je potřeba již zmíněný Selendroid-Server, který se nachází na jeho oficiálních stránkách. [55] Zde se nachází i ukázková aplikace pro testování pomocí Selendroidu, jejíž část je otestována na Obrázek 11. Zdrojový kód (Java) selendroid-test-app-0.17.0.apk . Důležité je nastavit také správné proměnné prostředí [56] a spustit emulátor nebo připojit fyzické zařízení se správnou verzí API. Pomocí *http://localhost:4444/wd/hub/status* ve webovém prohlížeči zkontrolujeme, zda je výsledek podobný Obrázek 12. Výsledek *http://localhost:4444/wd/hub/status*

Při zkoušení tohoto frameworku jsem narazila na chybu při kontrolování výstupu z lokálního serveru. Server se mi podařilo spustit, ale nedokázal najít spuštěný emulátor, bez kterého jsem nemohla pokračovat. Pomocí dostupných informací jsem si načetla o psaní testovacích kódů a zjistila, jak by správně měl tento framework pracovat.

```
1. public class SelendroidTest {
 2.
 3.    private WebDriver driver ;
 4.
 5.    @BeforeSuite
 6.    public void setUp() throws Exception {
 7.       SelendroidConfiguration config = new SelendroidConfiguration();
 8.       config.addSupportedApp("selendroid-test-app-0.9.0.apk");
 9.       SelendroidLauncher selendroidServer = new SelendroidLauncher(config);
10.       selendroidServer.launchSelendroid();
11.
12.       SelendroidCapabilities caps = new
13.          SelendroidCapabilities("io.selendroid.testapp:0.9.0");
14.       driver = new SelendroidDriver(caps);
15.    }
16.
17.    @Test
18.    public void selendroidTest() throws Exception {
19.       WebElement inputField = driver.findElement(By.id("my_text_field"));
20.       Assert.assertEquals("true", inputField.getAttribute("enabled"));
```

```
21.         inputField.sendKeys("Selendroid");
22.         Assert.assertEquals("Selendroid", inputField.getText());
23.         WebElement button = driver.findElement(By.id("buttonTest"));
24.         button.click();
25.         button = driver.findElement(By.id("button2"));
26.         button.click();
27.         button = driver.findElement(By.id("startUserRegistration"));
28.         button.click();
29.         WebElement element = driver.findElement(By.id("label_username"));
30.         String text = element.getText();
31.         System.out.println(text);
32.         element = driver.findElement(By.id("inputUsername"));
33.         element.sendKeys("bob");
34.         element = driver.findElement(By.id("inputEmail"));
35.         element.sendKeys("test@gmail.com");
36.         element = driver.findElement(By.id("inputPassword"));
37.         element.clear();
38.         element.sendKeys("test1233");
39.
40.     }
41.
42.     @AfterSuite
43.     public void tearDown(){
44.         driver.quit();
45.     }
46. }
```

Obrázek 11. Zdrojový kód (Java) selendroid-test-app-0.17.0.apk [56]


```
{ status: 0, value: { "os": { "name": "Android" }, "build": { "browserName": "selendroid",
"version": "0.17.0" }, "supportedApps": [{ "appId": "io.selendroid.testapp:0.17.0", "mainActi-
vity": "io.selendroid.testapp.HomeScreenActivity", "basePackage": "io.selendroid.testapp" }],
"supportedDevices": [{ "screenSize": "320x480", "targetPlatform": "ANDROID17", "emulator":
true, "avdName": "latest" }, { "screenSize": "320x480", "targetPlatform": "ANDROID16", "emula-
tor": true, "avdName": "es" }, { "screenSize": "320x480", "targetPlatform": "ANDROID10", "emu-
lator": true, "avdName": "AVD_for_api10" }] }
```


Obrázek 12. Výsledek *http://localhost:4444/wd/hub/status* [55]

3.1.4 UI Automator

UI Automator je framework pro automatizaci testování uživatelského rozhraní, který je vhodný pro různá zařízení a aplikace s operačním systémem Android. Jeho rozhraní API umožňuje interakci s viditelnými prvky (tlačítka, menu, textová pole) na obrazovce zařízení nebo emulátoru. Zde uvedu některé třídy tohoto rozhraní a příklady jejich použití. [57], [58]

UiObject2 – Reprezentuje prvek uživatelského rozhraní, který je viditelný v zařízení nebo emulátoru. Může to být tlačítko, textové pole nebo jakýkoliv jiný prvek, se kterým může uživatel interagovat. Za pomocí této třídy lze provádět různé operace s prvkem pomocí metod – například klikání, získávání a zadávání textu atd. [57], [59]

BySelector – Díky této třídě pomocí specifikace prvku (zobrazený text, ID prvku, obsah popisku atd.) identifikuje konkrétní prvek uživatelského rozhraní. [57],[60]

By – Poskytuje metody pro třídu *BySelector*. Slouží tedy k určení typu objektu *BySelector*. Takto můžeme vytvářet selektory pomocí různých kritérií přehledným způsobem. [57],[61]

Configuration – Umožňuje nastavit klíčové parametry pro spouštění testů. Lze ovlivnit čekací dobu mezi jednotlivými kroky testu, čekání na zobrazení prvku, počet opakování atd. [57]

Pro psaní testů ve frameworku UI Automator budu využívat nástroj Android Studio. Jako u předchozích frameworků i zde je potřeba správná konfigurace, která se nastavuje v souboru *build.gradle.kts* (Obrázek 13. Konfigurace souboru *build.gradle.kts* pro UI Automator). [62] UI Automator je skvělý pro black-box testování uživatelského rozhraní, proto se ukázkový zdrojový kód bude zaměřovat na testování otevření aplikace Gmail, která se nachází na domovské obrazovce (Obrázek 14. Ukázka zdrojového kódu (Kotlin) s využitím frameworku UI Automator).

```
plugins {
    alias(libs.plugins.androidApplication)
    alias(libs.plugins.jetbrainsKotlinAndroid)
}

android {
    namespace = "com.example.uiautomator"
    compileSdk = 34

    defaultConfig {
        minSdk = 19
       testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            isMinifyEnabled = false
            proguardFiles(
                getDefaultProguardFile("proguard-android-optimize.txt"),
                "proguard-rules.pro"
            )
        }
    }
    compileOptions {
        sourceCompatibility = JavaVersion.VERSION_1_8
        targetCompatibility = JavaVersion.VERSION_1_8
    }
    kotlinOptions {
        jvmTarget = "1.8"
    }
    buildFeatures {
        viewBinding = true
    }
}

dependencies {

    implementation(libs.androidx.core.ktx)
    implementation(libs.androidx.appcompat)
    implementation(libs.material)
    implementation(libs.androidx.constraintlayout)
    implementation(libs.androidx.lifecycle.livedata.ktx)
    implementation(libs.androidx.lifecycle.viewmodel.ktx)
    implementation(libs.androidx.navigation.fragment.ktx)
    implementation(libs.androidx.navigation.ui.ktx)
    implementation(libs.androidx.junit.ktx)
    implementation(libs.androidx.runner)
```

```
48.     testImplementation(libs.junit)
49.     androidTestImplementation(libs.androidx.junit)
50.     androidTestImplementation("androidx.test.ext:junit:1.1.2")
51.     androidTestImplementation("androidx.test.espresso:espresso-core:3.3.0")
52.     androidTestImplementation("androidx.test.uiautomator:uiautomator:2.3.0")
53.     androidTestImplementation ("androidx.test.ext:junit:1.1.3")
54.     androidTestImplementation ("androidx.test.uiautomator:uiautomator:2.2.0")
55.     }
```

Obrázek 13. Konfigurace souboru *build.gradle.kts* pro UI Automator

```
1. @RunWith(AndroidJUnit4::class)
2. class OpenGmailTest {
3.
4.     private lateinit var device: UiDevice
5.
6.     @Before
7.     fun setUp() {
8.         device = UiDevice.getInstance(InstrumentationRegistry.getInstrumentation())
9.         device.pressHome()
10.    }
11.
12.    @Test
13.    fun openGmail() {
14.        val gmailSelector = By.text("Gmail")
15.        val gmail: UiObject2? = device.findObject(gmailSelector)
16.        if (gmail != null && gmail.isClickable) {
17.            gmail.click()
18.            device.wait(Until.hasObject(By.pkg("com.google.android.gm").depth(0)), 5000)
19.            assertTrue("Aplikace Gmail není v popředí", device.currentPackageName
               .contains("com.google.android.gm"))
20.        } else {
21.            throw AssertionError("Aplikace Gmail nebyla nalezena nebo není klikatelná")
22.        }
23.    }
24. }
```

Obrázek 14. Ukázka zdrojového kódu (Kotlin) s využitím frameworku UI Automator

3.1.5 XCTest

XCTest je oficiální testovací framework od Apple, který je určen pro vývojáře iOS a mobilních aplikací. Podporuje jak Objective-C, tak i Swift, jedná se o klíčové programovací jazyky pro platformu Apple. XCTest podporuje vytváření a spouštění jednotkových testů (unit tests), testů výkonu (performance tests) a testů uživatelského rozhraní (UI tests). Jednotkové testy slouží k ověření funkčnosti jednotlivých částí. Testy výkonu umožňují měřit výkon částí aplikace, což je klíčové pro optimalizaci. UI testy, realizované prostřednictvím XCUITest, rozšíření XCTest, umožňují automatizovat interakci s uživatelským rozhraním aplikace, což je zásadní pro ověření, že aplikace zobrazuje to, co má. Integrace XCTest do Xcode umožňuje spouštět testy přímo z vývojového prostředí, což usnadňuje identifikaci a opravu chyb v rané fázi vývoje. Díky tomu mohou vývojáři lépe a efektivněji pracovat na vylepšení svých aplikací, a to s menším rizikem, že se do finální verze dostanou nějaké chyby nebo problémy. [63], [64], [65], [66]

U **XCUITest,** podobně jako u Espresso, je potřeba mít přístup ke zdrojovému kódu aplikace (white-box). Pro psaní a spouštění testů je využíván nástroj XCTest, který podporuje nejen testy uživatelského rozhraní, ale další již zmíněné. V ukázce zdrojového kódu testujeme uživatelské rozhraní pomocí XCUITest. Nejsou potřeba speciální konfigurace, pouze stačí vytvořit nový soubor v projektu, vybrat cílovou aplikaci a poté vybrat „iOS UI Testing Bundle“. Emulátor je již implementován v samotném XCTest, takže není potřeba další konfigurace. [67]

Součástí je i nahrávání, které umožňuje nahrávání aktivit na obrazovce, z nichž následně vygeneruje testovací kód.

```
1. class SampleXCUITests: XCTestCase {
2.
3.     override func setUp() {
4.         super.setUp()
5.         continueAfterFailure = false
6.         XCUIApplication().launch()
7.     }
8.
9.     override func tearDown() {
10.    }
11.
12.    func testAlert() {
13.
14.        let app = XCUIApplication()
15.        app.buttons["Alert"].tap()
16.        XCTAssertEqual(app.alerts.element.label, "Alert")
17.        app.alerts.buttons["OK"].tap()
18.        XCTAssertEqual(app.alerts.count, 0)
19.    }
20.
21.    func testText() {
22.        let app = XCUIApplication()
23.        app.buttons["Text"].tap()
24.        let enterText = "Hi Browserstack!!"
25.        XCTAssert(app.textFields["Enter a text"].exists)
26.
27.        app.textFields["Enter a text"].tap()
28.        app.textFields["Enter a text"].typeText(enterText)
29.        app.typeText("\r")
30.
31.        XCTAssertEqual(app.staticTexts.element.label, enterText)
32.    }
33. }
```

Obrázek 15. Ukázka zdrojového kódu (Swift) s využitím frameworku XCUITest [68]

3.2 Nástroje pro kontinuální integraci

Kontinuální integrace je proces, který po každé změně kódu spustí testy. Potvrzením kódu do hlavní větve sdíleného úložiště se aktivuje automatizovaný systém pro sestavení a testování. Mezi nejčastější nástroje pro kontinuální integraci patří Jenkins, Circle CI, GitLab, GitHub (Actions). Nejlepší a nejjednodušší řešení kontinuální integrace je, dle mého názoru,

použití nástroje Jenkins. Funguje se všemi zmíněnými frameworky a podporuje GitHub projekty.

3.2.1 Jenkins

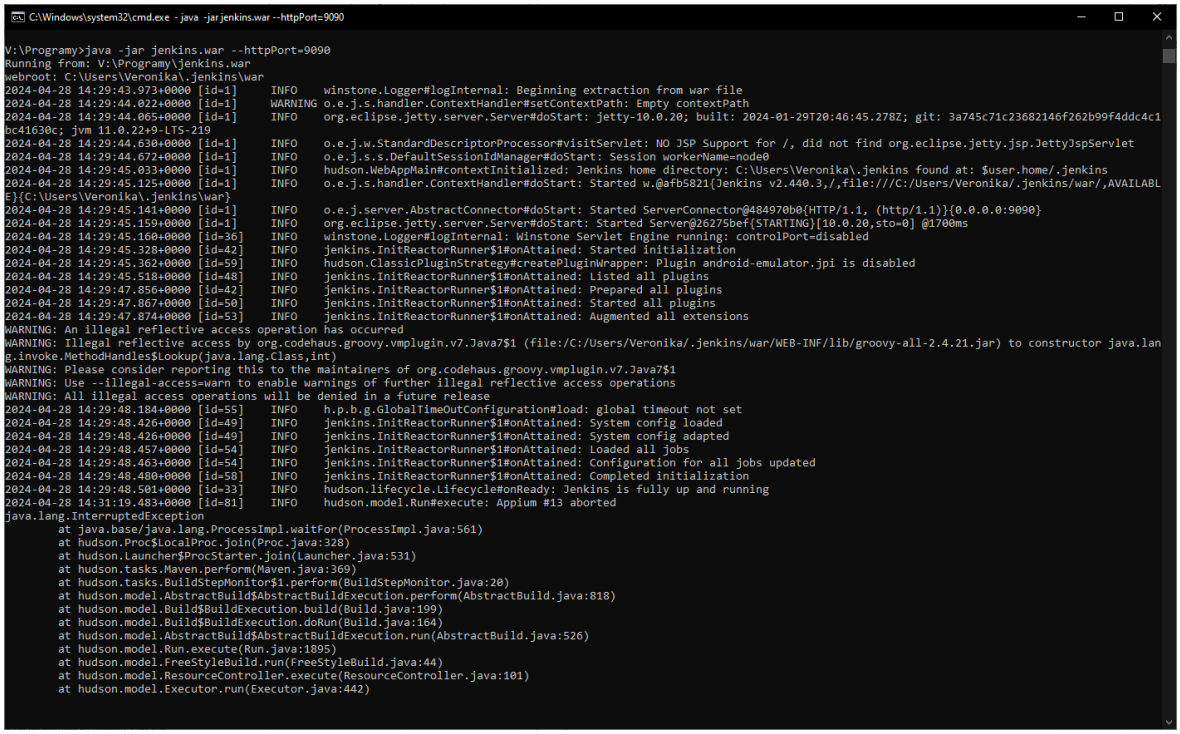
Jenkins je open-source automatizační server pro kontinuální integraci. Z důvodu rozsáhlého výběru pluginů a možnostem přizpůsobení je velmi oblíbený mezi vývojářskými týmy. Je vhodný pro různé potřeby a rozsáhlost projektů. Pro začátečníky může být trochu obtížnější z důvodu nastavení pokročilých konfigurací a modulů. Jenkins je hostován na vlastním serveru, což může zvyšovat provozní náklady. Je kompatibilní pouze s Java JDK verzí 11 a 17.

Podporuje všechny zmíněné frameworky, stačí pouze nastavit vhodné pluginy a skripty. Espresso a UI Automator spouští testy pomocí Gradle nebo Mavenu, XCTest spouští testy pomocí Xcode build kroků nebo nástrojů xcodebuild a fastlane.

Základ je stažení souboru [69]. Pomocí příkazové řádky přejdeme do složky, kde se nachází již stažený soubor „*jenkins.war*“ a spustíme server na *portu 9090* pomocí příkazu „*java -jar jenkins.war --httpPort=9090*“ (

Obrázek 16. Příkazový řádek spuštění serveru Jenkins). Otevřeme v prohlížeči adresu „*http://localhost:9090*“ a zde budeme nastavovat pluginy, skripty a spouštět testy.

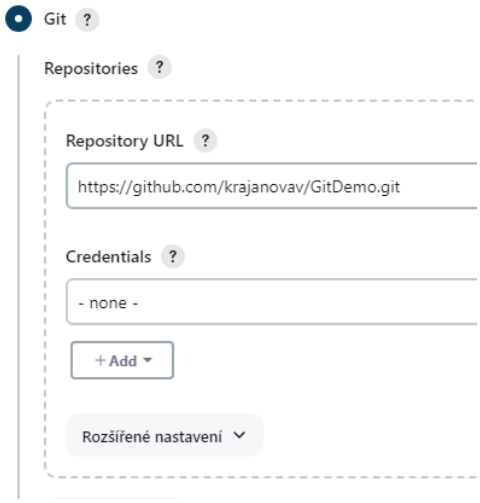
Pro ukázku jsem si vybrala propojení s GitHub. Využila jsem vlastní projekt s frameworkem Appium, kde jsem přidala potřebné závislosti (Obrázek 17. Přidané dependencies do souboru *pom.xml*. Dále je potřeba nastavit instalaci Maven (Obrázek 19. Nastavení Maven), instalaci Java JDK (Obrázek 20. Nastavení Java JDK) a ANDROID_HOME. Plugin pro emulátor je v tuto chvíli nefunkční, takže je potřeba mít lokálně spuštěný emulátor. Ovšem díky rozsáhlé podpoře tohoto nástroje bude zanedlouho k dispozici.



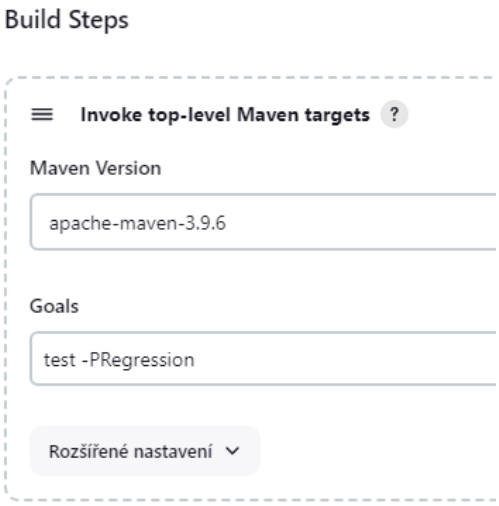
Obrázek 16. Příkazový řádek spuštění serveru Jenkins

```
<profiles>
    <profile>
    <id>Regression</id>
    <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-surefire-plugin</artifactId>
        <version>3.2.5</version>
        <configuration>
          <suiteXmlFiles>
            <suiteXmlFile>testng.xml</suiteXmlFile>
          </suiteXmlFiles>
        </configuration>
      </plugin>
    </plugins>
    </build>
    </profile>
</profiles>
```

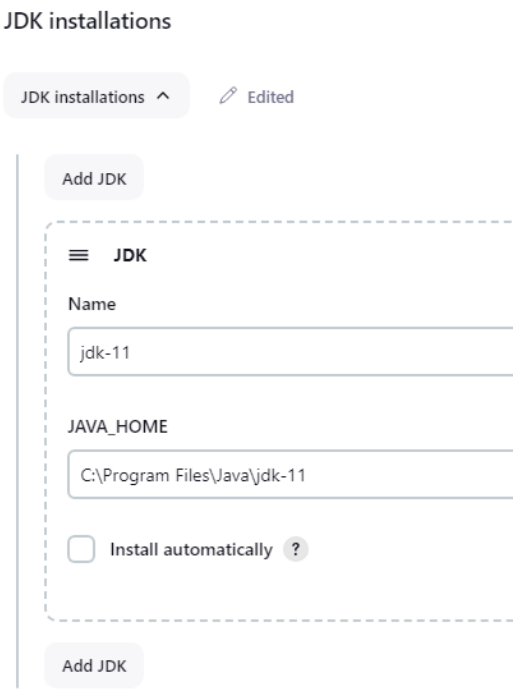
Obrázek 17. Přidané dependencies do souboru *pom.xml*



Obrázek 18. Nastavení Git repozitáře



Obrázek 19. Nastavení Maven pro spuštění



Obrázek 20. Nastavení Java JDK

```
[INFO] Tests run: 6, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 91.63 s -- in TestSuite
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 6, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  01:34 min
[INFO] Finished at: 2024-04-28T16:33:30+02:00
[INFO] ------------------------------------------------------------------------
Finished: SUCCESS
```

Obrázek 21. Výsledek sestavení a testování – Úspěch

4 POROVNÁNÍ AUTOMATIZOVANÝCH FRAMEWORKŮ

V této kapitole se zaměřím na srovnání vybraných automatizovaných frameworků pro testování mobilních aplikací. Porovnány budou frameworky Appium, Espresso, Selendroid, UI Automator a XCTest na základě několika klíčových kritérií.

Pro srovnání frameworků jsem si zvolila kritéria, jako jsou podporované platformy (Android, iOS, nebo obě), typy aplikací (nativní, hybridní, webové), podporované programovací jazyky, architekturu a implementaci, dostupnost a podporu, kontinuální integraci a specifické vlastnosti a funkce. Tyto faktory umožní poskytnout objektivní pohled na možnosti a omezení každého frameworku a pomohou určit, který z nich je nejvhodnější pro specifické testovací požadavky, s ohledem na podporované platformy, jazykovou podporu a typy aplikací, které lze testovat.

4.1 Porovnání podle platformy

Appium je univerzální nástroj podporující jak Android, tak iOS, což ho činí výbornou volbou pro týmy pracující napříč platformami. **Espresso** je specificky zaměřený na Android, zatímco **XCTest** slouží výhradně pro iOS, což je zase výhodné pro týmy specializující se na jednu platformu. **Selendroid** a **UI Automator** jsou také zaměřeny na Android s tím, že Selendroid je vhodnější pro starší verze Androidu.

Tabulka 3. Porovnání frameworků podle platformy

Nástroj	Platforma	Zaměření
Appium	Android, iOS	Univerzální, podporuje jak Android, tak iOS, black-box
Espresso	Android	Specifický pro Android, vhodný pro detailní testování na této platformě, white-box
Selendroid	Android	Zaměřený na starší verze Androidu, black-box
UI Automator	Android	Zaměřený na novější verze Androidu, black-box
XCTest	iOS	Specifický pro iOS, vhodný pro detailní testování na této platformě, white-box

4.2 Porovnání podle typu aplikací

Všechny zmíněné frameworky podporují nativní aplikace, avšak **Appium** a **Selendroid** se vyznačují schopností testovat také hybridní a webové aplikace. **Espresso** a **UI Automator** se soustředí primárně na nativní aplikace pro Android, zatímco **XCTest** je určen pro nativní a někdy hybridní aplikace na iOS.

Tabulka 4. Porovnání frameworků podle typu aplikací

Framework	Nativní aplikace	Hybridní aplikace	Webové aplikace
Appium	Ano	Ano	Ano
Espresso	Ano	Ne	Ne
Selendroid	Ano	Ano	Ano
UI Automator	Ano	Ne	Ne
XCTest	Ano	Spíše ne	Ne

4.3 Jazyková podpora

Appium a **Selendroid** podporují mnoho programovacích jazyků díky svému WebDriver API, včetně Java, Python, a Ruby. [[53][70] **Espresso** a **UI Automator** jsou převážně používány s Javou a Kotlinem, zatímco **XCTest** a jeho rozšíření XCUITest jsou integrovány s Objective-C a Swift pro iOS vývoj.

Tabulka 5. Jazyková podpora frameworků

Framework	Podporované programovací jazyky
Appium	Java, Python, Ruby and C#, atd.
Espresso	Java, Kotlin
Selendroid	Java, Python, Ruby, C#
UI Automator	Java, Kotlin
XCTest	Objective-C, Swift

4.4 Architektura a implementace

Appium funguje na principu klient-server architektury, kde testovací skript komunikuje s mobilním zařízením přes server. Tato architektura poskytuje flexibilitu v testování napříč různými zařízeními a platformami. **Selendroid** také využívají klient-server, přičemž podporuje paralelní spuštění testů na různých zařízeních.

Espresso a **XCTest** jsou integrovány přímo do oficiálních vývojových prostředí pro Android a iOS (Android Studio a Xcode). Tato integrace zjednodušuje vývoj a spuštění testů tím, že poskytuje vývojářům nástroje přímo v IDE, což umožňuje efektivněji psát, spouštět a debugovat testy uživatelského rozhraní.

UI Automator nevyužívá tradiční klient-server architekturu jako Appium nebo Selendroid. UI Automator spustí testovací skripty přímo na Android zařízení, kde tyto skripty interagují s uživatelským rozhraním bez potřeby komunikace přes externí server. UI Automator není integrován do Android Studio, jak je to u Espresso. Nicméně je součástí Android SDK a může být používán v Android Studio, ale s menším rozsahem integrace ve srovnání s Espresso. [71]

Tabulka 6. Architektura a implementace

Framework	Architektura	Integrace s IDE	Poznámky k použití
Appium	Klient-server	Ne	Flexibilita v testování napříč různými zařízeními
Espresso	Integrované v testovací aplikaci	Ano (Android Studio)	Umožňuje efektivní psaní a spouštění testů v IDE
Selendroid	Klient-server	Ne	Podporuje paralelní spuštění testů
UI Automator	Lokální (zařízení)	Omezená (Android Studio)	Integrovaný v Android SDK, nevyžaduje server
XCTest	Integrované v testovací aplikaci	Ano (Xcode)	Podpora nativní pro iOS, integrace s Xcode

4.5 Komunita a podpora

Framework **Appium** je vyvíjen a udržován komunitou a různými organizacemi. Díky tomu, že je open-source, kdokoli může přispět k jeho vývoji nebo se zapojit do komunity, což přispívá k jeho inovaci a aktualizaci. Na webových stránkách najdeme bohatou dokumentaci

a širokou podporou. **Espresso** a **UI Automator** jsou podporovány přímo společností Google, zatímco **XCTest** společností Apple, což zajišťuje velmi dobrou integraci a aktualizace. Vývoj frameworku **Selendroid** je podle nejnovějších informací pozastaven, a proto kolem tohoto nástroje je podpora a komunita méně aktivní než u těch, které jsou udržovány. Najít nové zdroje, aktualizace nebo pomoc je o dost obtížnější, což komplikuje jeho efektivní využití.

4.6 Kontinuální integrace

Integrace těchto frameworků pro automatizaci testování do procesů kontinuální integrace závisí na několika klíčových aspektech, které jsou specifické pro dané platformy a nástroje.

4.6.1 Appium

Integrace do CI: Appium je kompatibilní s mnoha CI nástroji, jako jsou Jenkins, TeamCity, CircleCI, a Travis CI. Integrace obvykle zahrnuje instalaci Appium serveru na CI serveru.

Konfigurace: Pro spuštění Appium testů v CI prostředí je třeba nakonfigurovat environmentální proměnné pro cestu k Node.js a Appium, jak již bylo zmíněno. Skripty spouštějící testy musí být součástí build procesu, často ve formě shell scriptů nebo gradle tasků. [72]

Paralelní testování: Appium podporuje paralelní spouštění testů na různých zařízeních, což zrychluje proces testování ve velkých projektech.[3] [72][73]

4.6.2 XCTest

Xcode a CI: XCTest je integrován s Xcode. To umožňuje využití Xcode Serveru pro automatizaci buildů a testů přímo z Xcode. V ostatních CI systémech je možné použít pluginy, jako je Xcode integration plugin pro Jenkins. [74]

Automatizace: XCTest může automaticky spouštět testy v rámci Xcode projektů pomocí *xcodebuild* příkazů. [74][74]

[3] Appium sám o sobě nemá vestavěnou podporu pro paralelní testování v jedné instanci serveru. Místo toho umožňuje spustit více instancí Appium serverů, každou s vlastním portem a konfigurací, aby mohla komunikovat s různými zařízeními.

4.6.3 Espresso

Gradle integrace: Framework Espresso je úzce spojen s Android SDK a Gradle a díky tomu umožňuje snadné začlenění do Gradle build skriptů. Jenkins a další CI nástroje mohou pak tyto skripty využívat pro spouštění Espresso testů. [75]

Test reporty: Espresso podporuje generování test reportů formátu HTML nebo XML, které lze v Jenkins zobrazovat pomocí vhodných pluginů. [76], [77]

4.6.4 Selendroid

Integrace s CI nástroji: Selendroid lze integrovat s Jenkins a dalšími CI nástroji podobně jako Appium, často pomocí podobných konfiguračních kroků.

Konfigurace: Selendroid vyžaduje nastavení Selenium Grid[4], který rozdělí a koordinuje testy mezi více zařízeními, zatímco CI proces se postará o to, aby byly tyto testy pravidelně spouštěny s každou novou změnou kódu. [78]

4.6.5 UI Automator

Integrace a nástroje: UI Automator lze integrovat do Jenkins pomocí standardních Gradle tasků pro spuštění testů. Může vyžadovat nastavení specifických environmentálních proměnných pro Android SDK.

4.7 Celkové porovnání

Po odzkoušení a zjištění informací o všech frameworcích za mě jednoznačně vítězí **Appium**. Je vhodný pro všechny typy aplikací, testují se v něm dobře všechny druhy aplikací. Není závislý na programovacím jazyce, podporuje jich několik, takže si každý vývojář/tester vybere svůj. Doporučila bych jej pro testery, kteří nemají velkou znalost v oblasti programování, jelikož je kompatibilní se spoustou knihoven a používá nástroje jako je TestNG. Samozřejmě je vhodný i pro vývojáře, kteří chtějí otestovat UI. Hlavní výhodou je, že je bezplatný a vyvíjen komunitou, takže všechny informace, rady a typy, ale i řešení problémů jsem vždy s přehledem našla.

[4] Selenium Grid umožňuje spouštět testy současně na různých zařízeních, operačních systémech a prohlížečích. Funguje jako server, který koordinuje akce mezi různými klienty.

Espresso bych spíše doporučila vývojářům Android aplikací, jelikož se pracuje přímo v projektu Android Studia s kódem aplikace. Většinou vývojáři potřebují rychle a efektivně otestovat aplikaci v rámci vývoje, což jim tento nástroj perfektně umožňuje. Nedoporučila bych jej pro testování webových aplikací, ale opravdu pouze pro Android aplikace, jelikož Appium je mnohem vhodnější a jednodušší pro tento druh aplikací. I přestože je tento framework vyvíjen společností Google, očekávala jsem stejnou podporu a množství informací jako u Appium, což se mi nepotvrdilo. Na druhou stranu má Espresso opravdu rozsáhlou dokumentaci, ve které je spousta užitečných informací.

Selendroid je určen pro testování starších verzí Androidu, avšak kvůli omezenému zájmu o staré aplikace je vývoj takovéto aplikace často neefektivní. Selendroid se používá pouze tehdy, kdy je potřeba udržet aplikaci funkční i na staré verzi Androidu. Vývoj Selendroidu byl pozastaven a je velmi obtížné shánět informace. Na stránkách je sice dokumentace tohoto frameworku, avšak mně, jako začátečníkovi, při problémech moc nepomohla. Tento framework jsem si zvolila ze zajímavosti.

UI Automator je nejvhodnější pro black-box testování UI, ale také přímo pro testování zařízení. Tento framework je navržen pro testery, kteří nepotřebují znát celý kód aplikace. Nejlépe se hodí pro nativní Android aplikace, kde je potřeba testování systému, jako jsou notifikace a systémová dialogová okna. S tímto frameworkem se mi pracovalo dobře, byl intuitivní a po Appium bych ho zařadila k těm nejlepším. Dokumentace, informace, rady a odpovědi na problémy se hledaly snadně, jelikož je hojně využíván.

XCTest má pevnou integraci s XCode a je nejlepší pro vývojáře iOS aplikací, kteří potřebují rychlé a efektivní testování. XCode je celkově velmi intuitivní – psaní a spouštění testů je velmi jednoduché. Není potřeba mít spuštěné servery nebo externí emulátor, vše je krásně na jednom místě. Bohužel jsem neměla tu možnost si XCTest vyzkoušet detailněji, ale díky dokumentaci a spoustě dostupným webovým stránkám a videím jsem frameworku porozuměla a neváhala bych jako vývojář iOS aplikací s ním pracovat.

Výběr frameworku by měl být řízen potřebami projektu, znalostmi týmu a infrastrukturou. Každý z těchto nástrojů nabízí specifické výhody, které mohou ovlivnit efektivnost, produktivitu a úspěšné testy. Tým by měl pečlivě zvážit, které nástroje nejlépe odpovídají jeho projektu.

5 APLIKACE

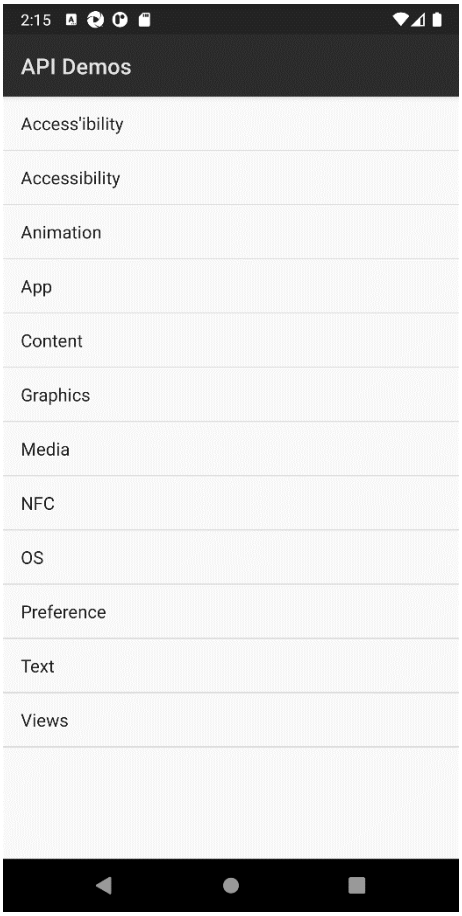
Pro demonstraci testů ve frameworku Appium jsem vybrala dvě aplikace, u kterých se zaměřím na základní a nejdůležitější testování mobilních aplikací. Jedná se o aplikaci **ApiDemos-debug.apk** [79] a aplikaci **General-Store.apk** [80]. Obě aplikace jsem vybrala na základě uvážení jaké testy, funkce a operace chci otestovat.

5.1 Struktura aplikací

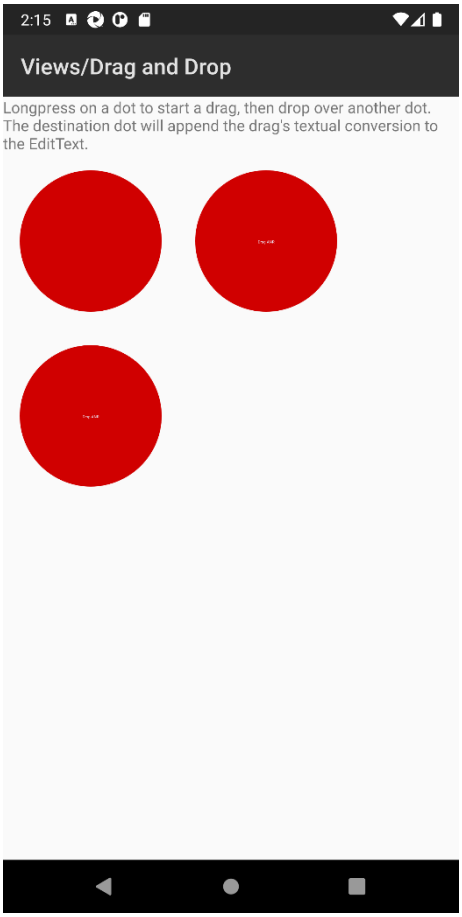
V této kapitole představím testované aplikace, jejich vlastnosti a funkce. Aplikace nejsou uvedeny na trhu, slouží tedy pouze pro testování .

5.1.1 Struktura ApiDemos-debug.apk

Jedná se o **nativní aplikaci**, která je součástí oficiálního repositáře Appium. Je určená pro demonstraci a testování funkcí systému Android pomocí frameworku Appium. Obsahuje různé funkce a rozhraní Android, které umožňují experimentovat s různými testy na mobilních zařízeních.



Obrázek 22. Hlavní stránka aplikace ApiDemos-Debug.apk



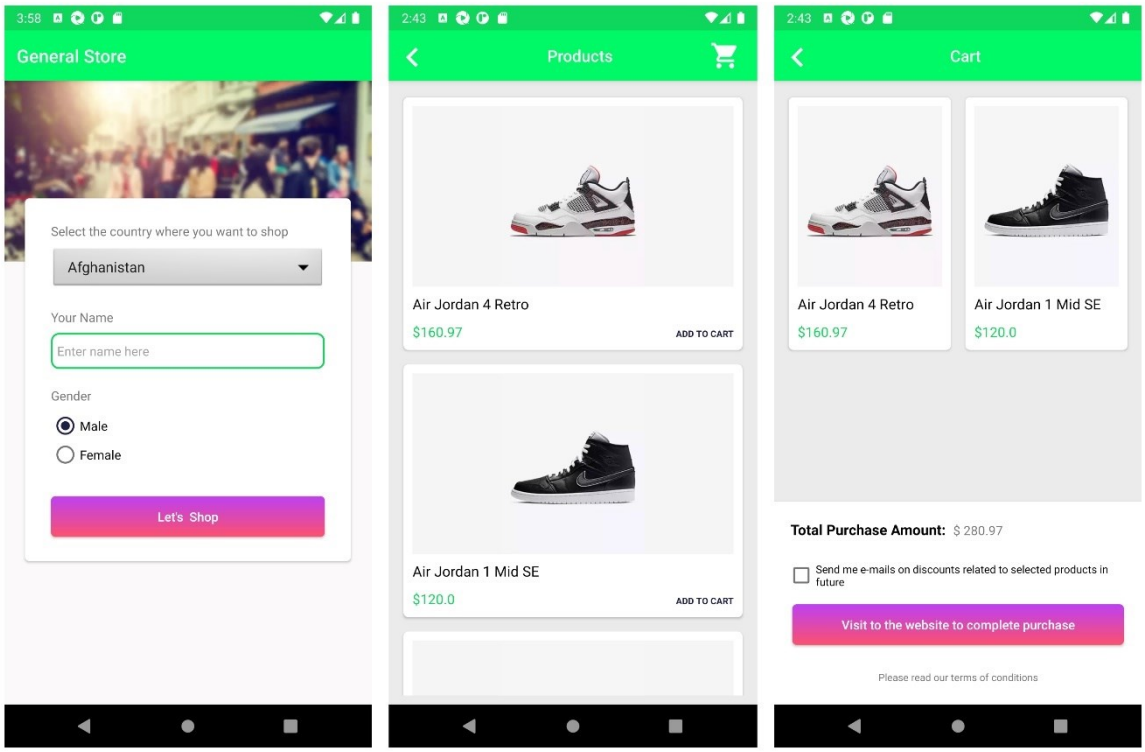
Obrázek 23. Ukázka Drag and Drop rozhraní aplikace ApiDemo-Debug.apk

5.1.2 Struktura General-Store.apk

General-Store.apk je **hybridní aplikace**, jejíž součástí je nativní část pro platformu Android a webová část.

Simuluje jednoduchou mobilní aplikaci pro obchod s botami. Průchod aplikací je poměrně jednoduchý:

1. Vyplnění formuláře
2. Přidání vybraných bot do košíku
3. Kliknutí na tlačítko s ikonou košíku
4. Kliknutí na tlačítko, které přesměrovává na webovou stránku
5. Zobrazení webové stránky přímo v aplikaci General-Store.apk



Obrázek 24. Ukázka rozhraní aplikace *General-Store.apk*

6 TESTOVÁNÍ APLIKACÍ

V předchozí části byly představeny aplikace, které budou použity v následném testování. Podíváme se na jednotlivé testovací případy (test cases). Důležité je však zmínit, jaké nástroje a prostředky je potřeba nainstalovat a jak je nakonfigurovat.

6.1 Nástroje a jejich konfigurace

Nyní se zaměřím na nástroje a jejich přípravu k testování, jež jsem použila při testování mobilních aplikací. Ovšem ne všechny tyto zmíněné nástroje jsou nutností. Testeři si mohou vybrat z více možností podle toho, který nástroj jim bude nejlépe vyhovovat. Každá podkapitola obsahuje detailní popis, proč jsem se rozhodla použít konkrétní nástroj pro testování.

6.1.1 Appium

Framework Appium je podrobně rozebrán v kapitole Appium. Nyní se zaměřím na to, jak ho nainstalovat a co vše je potřeba nastavit.

Jelikož se Appium stahuje pomocí příkazu *npm*, je potřeba stáhnout Node.js[5] https://nodejs.org/en a přidat do systémové proměnné stejně jako Android SDK (zakomponované v Android Studiu). Nyní příkaz *npm install -g appium* v příkazovém řádku Node.js nainstaluje nejnovější verzi Appium. Nutné je doinstalovat driver UI Automator2 příkazem *appium driver install uiautomator2.*

Po úspěšných instalacích a nastavení systémových proměnných už stačí pouze spustit Appium server pomocí příkazu *appium*.

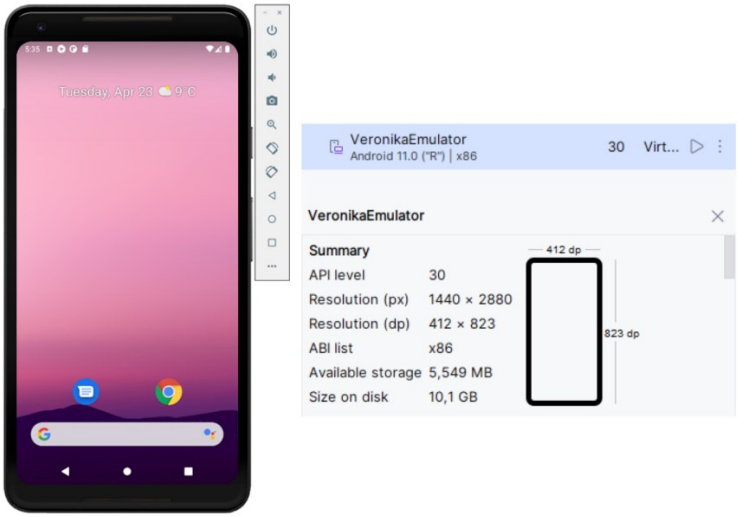
6.1.2 Android studio (Emulátor)

Android studio je integrované vývojové prostředí od společnosti Google, vytvořeno pro vývoj aplikací pro platformu Android. Pro správné použití je potřeba nastavit Android SDK jako ANDROID_HOME do systémové proměnné.

[5] Node.js je bezplatné spouštěcí prostředí JavaScriptu, které vývojářům umožňuje vytvářet servery, webové aplikace, nástroje příkazového řádku a skripty. https://nodejs.org/en

Součástí je **Android Emulator**, který simuluje zařízení se systémem Android v počítači. Umožňuje testovat aplikace na různých zařízeních a úrovních Android API. Díky tomu není potřeba fyzického zařízení pro testování mobilní aplikace. [81]

Pro výběr této možnosti jsem se rozhodla z důvodu rozsáhlé podpory od Google, rychlosti a především kvůli jednoduchosti instalace, navigace a celkové práce s tímto emulátorem. Vybrala jsem si konkrétně zařízení **Pixel 2 XL**, kvůli jeho navigační liště s menu na spodní straně obrazovky. Jelikož nejsem uživatelem Androidu, tento typ rozhraní mi přijde intuitivní a usnadňuje orientaci. Použila jsem **API level 30**, který odpovídá verzi Android 11.



Obrázek 25. Ukázka Android Emulator a informace o konfiguraci

6.1.3 Eclipse

Eclipse je populární **IDE**, které je využíváno pro vývoj Java projektů, včetně psaní testů pro mobilní aplikace. Podporuje Maven projekty, což umožňuje snadné spravování závislostí a sestavování projektů.

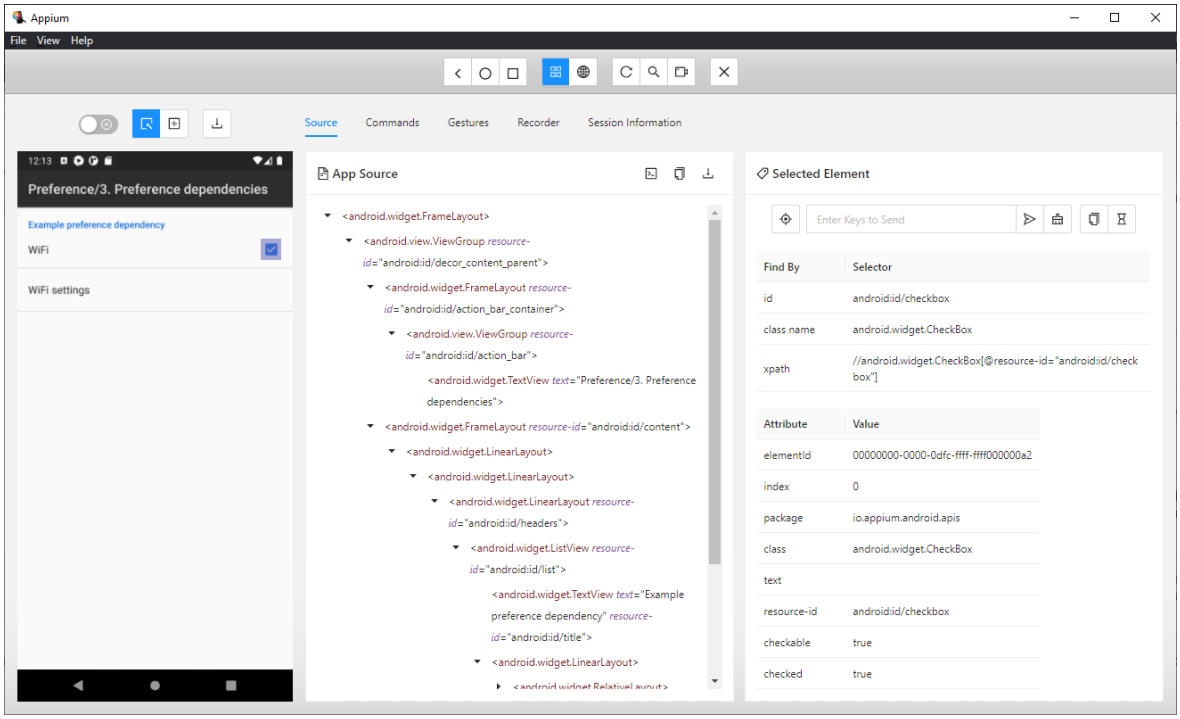
Pro psaní testů jsem tedy zvolila Eclipse a Maven projekt. Nejdůležitější část pro správné psaní a spouštění testů je soubor *pom.xml*, kde se nachází závislosti (dependecies). Vzhledem k tomu, že se jedná o projekt Maven, můžeme použít Maven repository [82]. Důležité je zvolit správné verze projektů, které budou kompatibilní s verzí Javy, Mavenu a dalšími.

```
<dependencies>
      <!-- https://mvnrepository.com/artifact/io.appium/java-client -->
      <dependency>
          <groupId>io.appium</groupId>
          <artifactId>java-client</artifactId>
          <version>9.2.2</version>
      </dependency>
      <!-- https://mvnrepository.com/artifact/org.testng/testng -->
      <dependency>
          <groupId>org.testng</groupId>
          <artifactId>testng</artifactId>
          <version>7.4.0</version>
      </dependency>
      <!-- https://mvnrepository.com/artifact/org.slf4j/slf4j-api -->
      <dependency>
          <groupId>org.slf4j</groupId>
          <artifactId>slf4j-api</artifactId>
          <version>2.0.12</version>
      </dependency>
      <!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-support -->
      <dependency>
          <groupId>org.seleniumhq.selenium</groupId>
          <artifactId>selenium-support</artifactId>
          <version>4.19.0</version>
      </dependency>
</dependencies>
```

Obrázek 26. Ukázka závislostí v souboru *pom.xml*

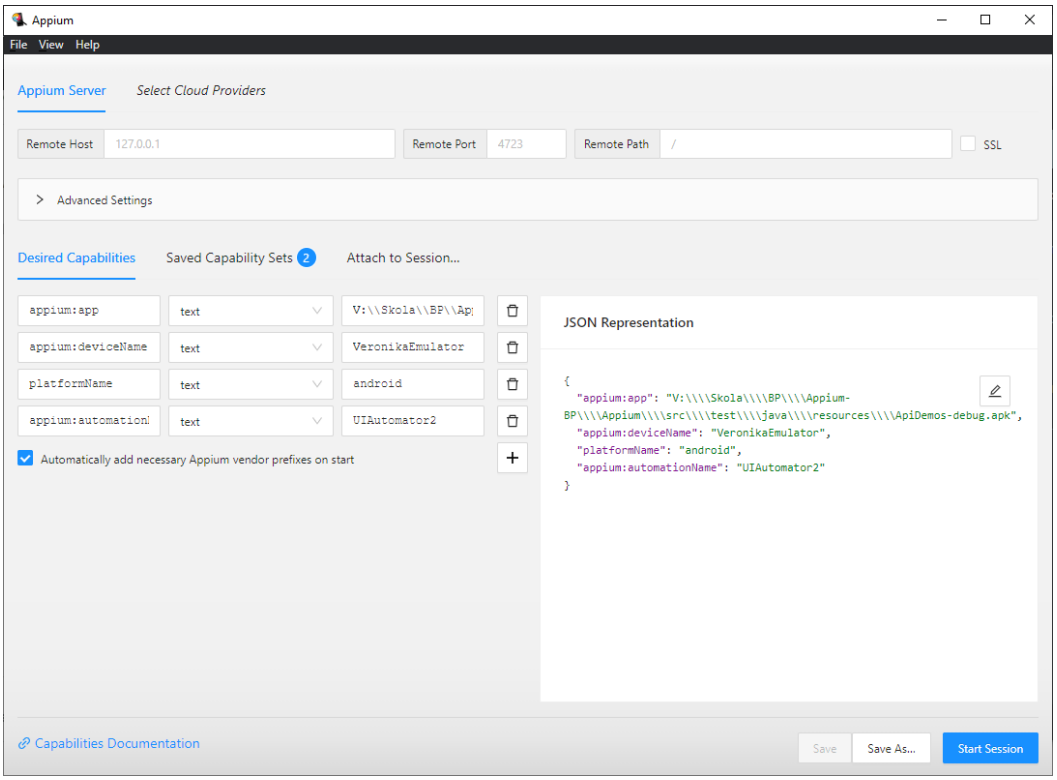
6.1.4 Appium Inspector

Appium Inspector je **inspektor GUI** pro mobilní aplikace Android i iOS. Funguje na základě serveru Appium. V jednoduchosti lze říct, že identifikuje požadované elementy, které jsou viditelné na obrazovce, pomocí id, xpath, class, atd. Vidíme také stav elementu, jako je „*checked*“ (zakliknutý), „*clickable*“ (kliknutelný), atd. To umožňuje tyto elementy najít a otestovat.



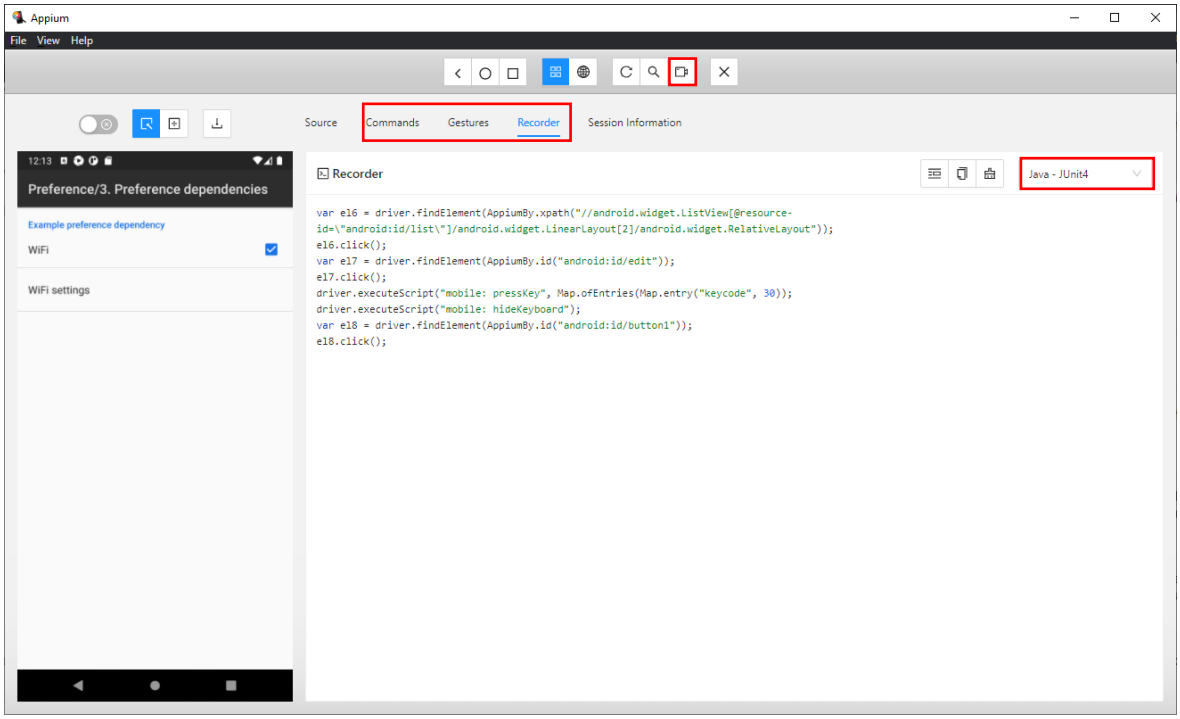
Obrázek 27. Ukázka rozhraní Appium Inspector

Pro přístup k mobilní aplikaci a emulátoru je zásadní vyplnit správné informace do požadovaných možností. Tyto informace získáváme z našeho hlavního kódu v Eclipse, který nastavuje Android driver. [83]



Obrázek 28. Nastavení Appium Inspector pro aplikaci *Api-Demos-debug.apk*

Appium Inspector má velmi zajímavou funkci **„Recorder“**, která umožňuje nahrávat průchod aplikací a následně vytvořit test. Díky záložce „Commands“ a „Gestures“ můžeme provádět různé akce, aniž bychom zasahovali přímo v emulátoru. Všechny akce, které provedeme v Appium Inspector při nahrávání, se převedou na kód v požadovaném jazyce.



Obrázek 29. Ukázka Appium Inspector Recorder

6.2 Testování ApiDemos-debug.apk

V tuto chvíli jsou nainstalovány a nakonfigurovány všechny potřebné nástroje pro začátek testování. Pro všechny následující testy bude potřeba základního testu zvaného **BaseTest01**. Tento test sám o sobě nic netestuje, naopak spouští Appium server, konfiguruje zařízení a aplikaci. Slouží také pro inicializací základních funkcí jako například dlouhý stisk tlačítka (*longPressAction()*), posouvání na konec (*scrollToEndAction()*), přejetí prstem (*swipeAction()*) a nakonec ukončení Appium serveru (*shutDown()*).

Můžeme si všimnou anotace *@BeforeClass* a *@AfterClass*, které nám umožňují určit, kdy se určitá část kódu bude spouštět. *@BeforeClass* se spustí před každou třídou testů a *@AfterClass* naopak po každé třídě testů. Je to perfektní způsob, jak zajistit, že se tyto přípravné kroky spustí před samotným testem. Tyto anotace pocházejí z TestNG, který jsme přidali v předešlé kapitole Eclipse do závislostí.

```
 1. public class BaseTest01 {
 2.
 3.       public AndroidDriver driver;
 4.       public AppiumDriverLocalService service;
 5.
 6. @BeforeClass
 7.       public void ConfigureAppium() throws MalformedURLException
 8.       {
 9.
10.         service = new AppiumServiceBuilder().withAppiumJS(new File("C:/Users/Veronika/App
        Data/Roaming/npm/node_modules/appium/build/lib/main.js"))
11.                         .withIPAddress("127.0.0.1").usingPort(4723).build();
12.                 service.start();
13.
14.                 UiAutomator2Options options = new UiAutomator2Options();
15.                 options.setDeviceName("VeronikaEmulator");
16.                 options.setChromedriverExecutable("V:\\Skola\\BP\\Appium-
        BP\\ChromeDriver\\chromedriver.exe");
17.                 options.setApp("V:\\Skola\\BP\\Appium-
        BP\\Appium\\src\\test\\java\\resources\\ApiDemos-debug.apk");
18.                 options.setNewCommandTimeout(Duration.ofSeconds(120));
19.                 driver = new AndroidDriver(new URL("http://127.0.0.1:4723"), options);
20.                 driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(30));
21.       }
22.
23.       public void longPressAction(WebElement ele)
24.       {
25.                 ((JavascriptExecutor)driver).executeScript("mobile: longClickGesture",
        ImmutableMap.of("elementId", ((RemoteWebElement) ele).getId(), "duration", 2000));
26.       }
27.
28.       public void scrollToEndAction() {
29.              boolean canScrollMore;
30.              do
31.              {
32.              canScrollMore = (Boolean) ((JavascriptExecutor) driver).executeScript("mo
                bile: scrollGesture", ImmutableMap.of(
33.                  "left", 100, "top", 100, "width", 200, "height", 200,
34.                  "direction", "down",
35.                  "percent", 3.0
36.              ));
37.              } while(canScrollMore);
38.       }
39.
40.       public void swipeAction(WebElement ele, String direction)
41.       {
42.              ((JavascriptExecutor) driver).executeScript("mobile: swipeGesture", Immuta
        bleMap.of(
43.                   "elementId", ((RemoteWebElement) ele).getId(),
44.                   "direction", direction,
45.                     "percent", 0.25
46.                 ));
47.       }
48.
49. @AfterClass
50.       public void shutDown()
51.       {
52.         driver.quit();
53.         service.stop();
54.       }
55.
56. }
```

Obrázek 30. Ukázka zdrojového kódu BaseTest01

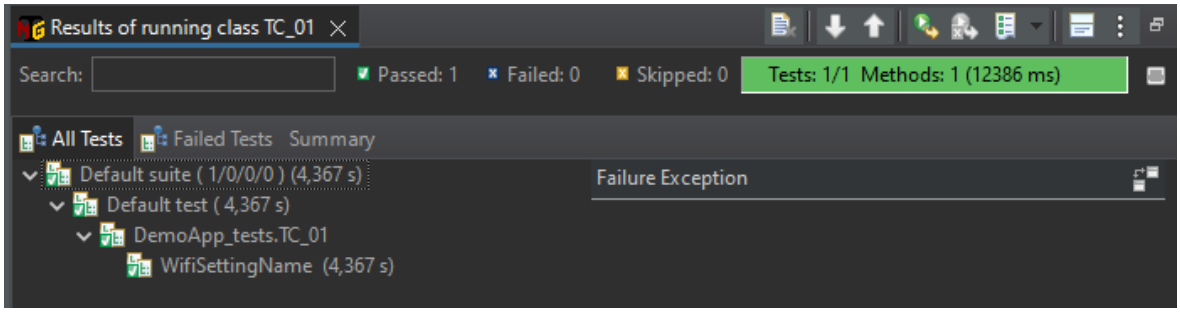
6.2.1 TC_01 Nastavení Wi-Fi

Tabulka 7. Specifikace TC_01 Nastavení Wi-Fi

Název	TC_01 Nastavení Wi-Fi
Popis	Testovací případ testuje nastavení Wi-Fi.
Očekávaný výsledek	Zapnutí Wi-Fi a úspěšné zadání názvu
Pre-conditions	Spuštěný server, emulátor a aplikace
Testovací kroky	1. Kliknutí na tlačítko „Preference“. 2. Kliknutí na tlačítko „Preference dependencies. 3. Kliknutí na zaškrtávací box. 4. Kliknutí na tlačítko „Wi-Fi settings“. 5. Vložení textu „Veronika WiFi“ do textového pole. 6. Kliknutí na tlačítko „OK“.
Vstupní data	1. Validní a. Kliknutí na správné tlačítka ve správném pořadí b. Wi-Fi input: Veronika WiFi 2. Nevalidní a. Kliknutí na nesprávné místo/tlačítko nebo v nesprávném pořadí

```
1. public class TC_01 extends BaseTest01{
 2.
 3.     @Test
 4.       public void WifiSettingName() throws MalformedURLException {
 5.
 6.         driver.findElement(AppiumBy.accessibilityId("Preference")).click();
 7.         driver.findElement(By.xpath("//android.widget.TextView[@content-desc='3. Prefe-
            rence dependencies']")).click();
 8.         driver.findElement(By.id("android:id/checkbox")).click();
 9.         driver.findElement(By.xpath("(//android.widget.RelativeLayout)[2]")).click();
10.         driver.findElement(By.id("android:id/edit")).sendKeys("Veronika WiFi");
11.         String alertTitle = driver.findElement(By.id("android:id/alertTitle")).getText();
12.         Assert.assertEquals(alertTitle, "WiFi settings");
13.         driver.findElements(AppiumBy.className("android.widget.Button")).get(1).click();
14.
15.     }
16. }
```

Obrázek 31. Ukázka zdrojového kódu TC_01 Nastavení Wi-Fi



Obrázek 32. Výsledek TC_01 Nastavení Wi-Fi

6.2.2 TC_02 Scroll

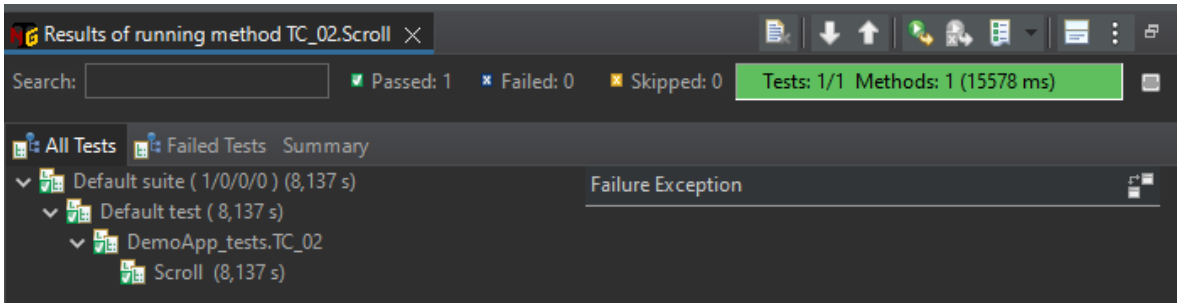
Druhý testovací případ se zaměřuje na „scroll“ (posouvání). Pro otestování této metody můžeme použít již vytvořenou funkci *scrollToEndAction(),* která bude scrollovat až na konec stránky. Pro scroll na určitý element využíváme UI Automator a text elementu.

Tabulka 8. Specifikace TC_02 Scroll

Název	TC_02 Scroll
Popis	Testovací případ testuje scroll na text „WebView“.
Očekávaný výsledek	Scroll na text „WebView“
Pre-conditions	Spuštěný server, emulátor a aplikace
Testovací kroky	1. Kliknutí na tlačítko „Views“. 2. Scroll na text „WebView“.
Vstupní data	1. Validní a. Kliknutí na správné tlačítka ve správném pořadí. b. Scroll na text „WebView“. 2. Nevalidní a. Kliknutí na nesprávné místo/tlačítko nebo v nesprávném pořadí. b. Scroll na text „Picker“.

```
1. public class TC_02 extends BaseTest01{
 2.
 3.     @Test
 4.       public void Scroll() throws MalformedURLException, InterruptedException
 5.     {
 6.
 7.         driver.findElement(AppiumBy.accessibilityId("Views")).click();
 8.         driver.findElement(AppiumBy.androidUIAutomator("new UiScrollable(new UiSe-
           lector()).scrollIntoView(text(\"WebView\"));"));
 9.         //scrollToEndAction();
10.         Thread.sleep(2000);
11.
12.     }
13. }
```

Obrázek 33. Ukázka zdrojového kódu TC_02 Scroll



Obrázek 34. Výsledek TC_02 Scroll

6.2.3 TC_03 Long press

Tento testovací případ slouží k otestování „long press“ (dlouhý stisk tlačítka), pomocí funkce *longPressAction()* z kódu BaseTest01. Na stránce „Custom Adapter“ máme možnost tuto funkci vyzkoušet pomocí podržení tlačítka „People names“. Po long pressu se nám zobrazí menu s textem „Sample menu“, jehož zobrazení otestujeme pomocí metody *isDisplayed()*.

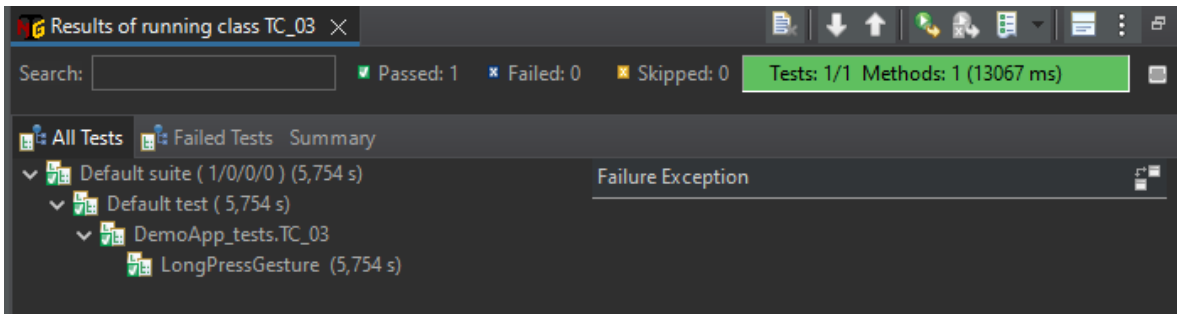
Tabulka 9. Specifikace TC_03 Dlouhý stisk tlačítka

Název	TC_03 Long press
Popis	Testovací případ testuje long press tlačítka „People Names“.
Očekávaný výsledek	Zobrazení „Sample menu“
Pre-conditions	Spuštěný server, emulátor a aplikace
Testovací kroky	1. Kliknutí na tlačítko „Views“. 2. Kliknutí na tlačítko „Expandable Lists“. 3. Kliknutí na tlačítko „Custom Adapter“.

	4. Long press tlačítka „People Names“.
Vstupní data	1. Validní a. Kliknutí na správné tlačítka ve správném pořadí. b. Long press tlačítka „People Names“. 2. Nevalidní a. Kliknutí na nesprávné místo/tlačítko nebo v nesprávném pořadí. b. Long press mimo vyznačená tlačítka.

```
1. public class TC_03 extends BaseTest01{
 2.
 3.     @Test
 4.       public void LongPressGesture() throws MalformedURLException {
 5.
 6.         driver.findElement(AppiumBy.accessibilityId("Views")).click();
 7.         driver.findElement(By.xpath("//android.widget.TextView[@content-desc=\"Expandable
           Lists\"]")).click();
 8.         driver.findElement(AppiumBy.accessibilityId("1. Custom Adapter")).click();
 9.         WebElement ele = driver.findElement(By.xpath("//android.widget.TextView[@text='Peo
           ple Names']"));
10.         longPressAction(ele);
11.
12.         String menuText = driver.findElement(By.id("android:id/title")).getText();
13.         Assert.assertEquals(menuText, "Sample menu");
14.         Assert.assertTrue(driver.findElement(By.id("android:id/title")).isDisplayed());
15.     }
16. }
```

Obrázek 35. Ukázka zdrojového kódu TC_03 Long Press



Obrázek 36. Výsledek TC_03 Long press

6.2.4 TC_04 Swipe obrázku

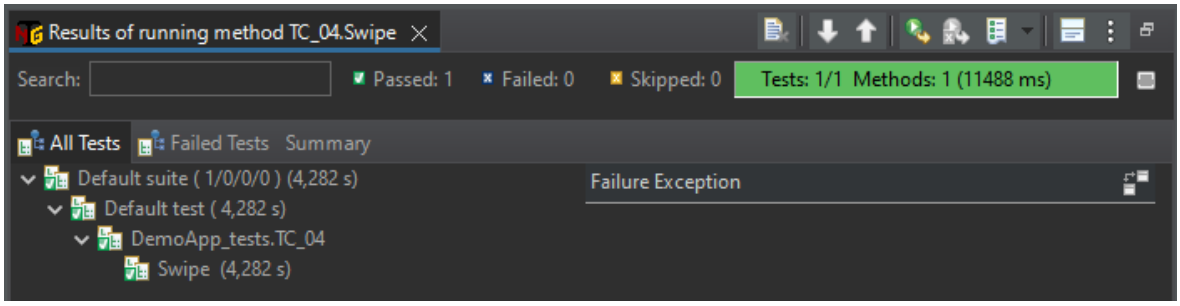
Jednou z dalších hlavních funkcí je „swipe“ (přejetí prstem po obrazovce), která je definovaná v předchozí kapitole v kódu BaseTest01. V tomto testovacím případu funkci otestujeme pomocí swipu obrázku a atributu „*focusable*“ (zaměřený).

Tabulka 10. Specifikace TC_04 Swipe

Název	TC_04 Swipe
Popis	Testovací případ testuje swipe obrázku.
Očekávaný výsledek	První obrázek má nepravdivý atribut *focusable*
Pre-conditions	Spuštěný server, emulátor a aplikace
Testovací kroky	1. Kliknutí na tlačítko „Views“. 2. Kliknutí na tlačítko „Gallery“. 3. Kliknutí na tlačítko „1. Photos“. 4. Swipe doleva na druhý obrázek.
Vstupní data	1. Validní a. Kliknutí na správné tlačítka ve správném pořadí. b. Swipe doleva o jeden obrázek. 2. Nevalidní a. Kliknutí na nesprávné místo/tlačítko nebo v nesprávném pořadí. b. Swipe doprava.

```
1. public class TC_04 extends BaseTest01{
 2.
 3.     @Test
 4.       public void Swipe() throws MalformedURLException
 5.     {
 6.
 7.         driver.findElement(AppiumBy.accessibilityId("Views")).click();
 8.         driver.findElement(AppiumBy.accessibilityId("Gallery")).click();
 9.         driver.findElement(By.xpath("//android.widget.TextView[@text='1. Phot
           os']")).click();
10.         WebElement firstImage = driver.findElement(By.xpath("(//android.wid
           get.ImageView)[1]"));
11.         Assert.assertEquals(driver.findElement(By.xpath("(//android.wid
           get.ImageView)[1]")).getAttribute("focusable"),"true");
12.         swipeAction(firstImage, ("left"));
13.         Assert.assertEquals(driver.findElement(By.xpath("(//android.wi
           dget.ImageView)[1]")).getAttribute("focusable"),"false");
14.
15.     }
16. }
```

Obrázek 37. Ukázka zdrojového kódu TC_04 Swipe



Obrázek 38. Výsledek TC_04 Swipe

6.2.5 TC_05 Přetahování

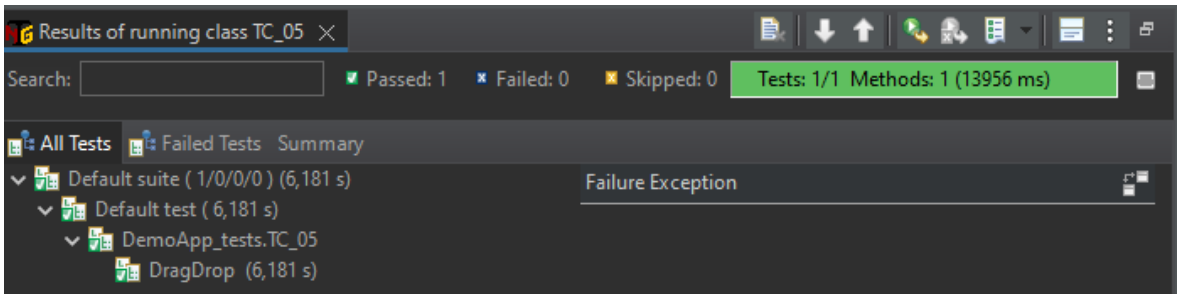
Zajímavá a využívaná funkce, kterou v tomto testovacím případu otestuji, je přetahování. V této aplikaci je možnost na stránce „Drag and Drop“ pomocí long press červeného obrazce zahájit přetahování na jiný obrazec. Pomocí JavaScript kódu spustíme funkci přetažení a definujeme souřadnice.

Tabulka 11. Specifikace TC_04 Přetahování

Název	TC_05 Přetahování
Popis	Testovací případ testuje přetahování obrazce.
Očekávaný výsledek	Zobrazení textu „Dropped!“
Pre-conditions	Spuštěný server, emulátor a aplikace
Testovací kroky	1. Kliknutí na tlačítko „Views“. 2. Kliknutí na tlačítko „Drag and Drop“. 3. Přetáhnutí červeného obrazce z levého horního rohu do pravého dolního rohu.
Vstupní data	1. Validní a. Kliknutí na správná tlačítka ve správném pořadí. b. Přetáhnutí červeného obrazce na jiné místo. 2. Nevalidní a. Kliknutí na nesprávné místo/tlačítko nebo v nesprávném pořadí. b. Přetáhnutí na okraj obrazovky.

```
1. public class TC_05 extends BaseTest01{
 2.
 3.     @Test
 4.       public void DragDrop() throws MalformedURLException, InterruptedException
 5.     {
 6.
 7.         driver.findElement(AppiumBy.accessibilityId("Views")).click();
 8.         driver.findElement(AppiumBy.accessibilityId("Drag and Drop")).click();
 9.         WebElement source = driver.findElement(By.id("io.appium.an
           droid.apis:id/drag_dot_1"));
10.         ((JavascriptExecutor) driver).executeScript("mobile: dragGesture", ImmutableMap.of(
11.             "elementId", ((RemoteWebElement) source).getId(),
12.             "endX", 829,
13.             "endY", 752
14.         ));
15.         Thread.sleep(3000);
16.         String result = driver.findElement(By.id("io.appium.android.apis:id/drag_re
           sult_text")).getText();
17.         Assert.assertEquals(result, "Dropped!");
18.
19.     }
20. }
```

Obrázek 39. Ukázka zdrojového kódu TC_05 Přetahování

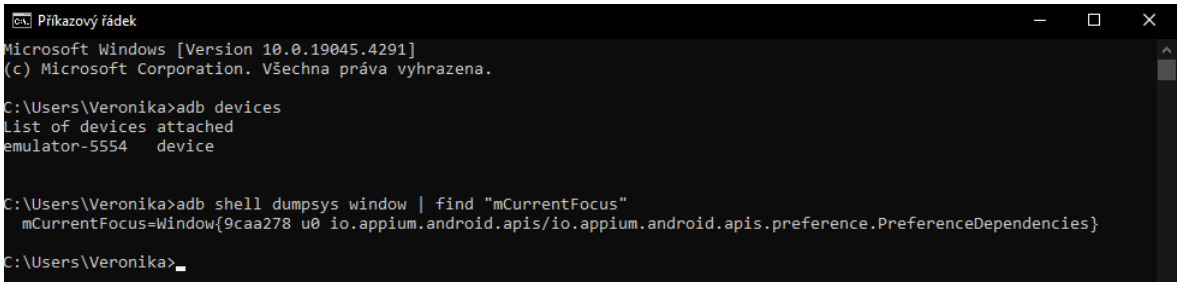


Obrázek 40. Výsledek TC_05 Přetahování

6.2.6 TC_06 Aktivita a systémové funkce

V posledním testu se jedná o trochu komplexnější kód, který bude začínat přímo na stránce „Preference dependencies“. Díky objektu *Activity* a JavaScript kódu lze spustit aplikaci v určité aktivitě (přímo na požadované stránce). Třídu a aktivitu potřebnou pro fungující kód nalezneme pomocí příkazového řádku[6] a emulátoru, na kterém běží daná aktivita. [84]

[6] Příkaz pro zjištění aktivity: *adb shell dumpsys window | find “mCurrentFocus“*



Obrázek 41. Příkazový řádek pro zjištění třídy a aktivity

Dále test obsahuje překlopení obrazovky na šířku pomocí *DeviceRotation()* a *rotate*. Schránka, do které lze ukládat text, je zde otestována funkcemi *setClipboardText()* a *getClipboardText()*.

Nejčastěji užívané funkce mobilního telefonu jsou jeho tlačítka. Třída *AndroidKey* definuje základní tlačítka (zpět, domov, přehled), ale také písmena, hlasitost, atd. Pomocí funkce *pressKey()* se stiskne vybrané tlačítko a provede se jeho akce.

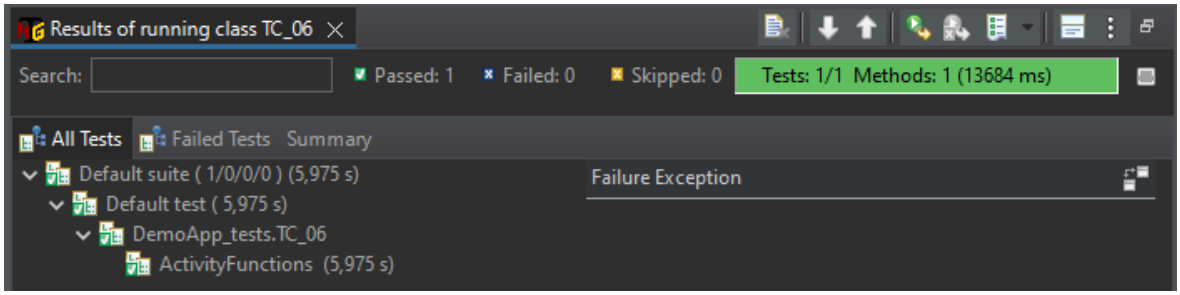
Tabulka 12. Specifikace TC_04 Aktivita a systémové funkce

Název	TC_06 Aktivita a systémové funkce
Popis	Testovací případ obsahuje otevření aktivity, překlopení obrazovky, funkce schránky a základních tlačítek.
Očekávaný výsledek	Zobrazení domovské stránky.
Pre-conditions	Spuštěný server, emulátor a aplikace na stránce „Preference dependencies“
Testovací kroky	1. Kliknutí na zaškrtávací box. 2. Překlopení obrazovky. 3. Kliknutí na tlačítko „Wi-Fi settings“. 4. Uložení textu do schránky. 5. Vložení textu ze schránky. 6. Stisknutí klávesy „Enter“. 7. Stisknutí klávesy „Back“ (Zpět) 8. Stisknutí klávesy „Home“ (Domov)
Vstupní data	1. Validní a. Kliknutí na správná tlačítka ve správném pořadí. b. Uložení textu do schránky.

	c. Stisknutí kláves ve správném pořadí. 2. Nevalidní a. Kliknutí na nesprávné místo/tlačítko nebo v nesprávném pořadí. b. Nevložení textu do schránky. c. Stisknutí kláves v nesprávném pořadí.

```
1. public class TC_06 extends BaseTest01{
 2.
 3.     @Test
 4.       public void ActivityFunctions() throws MalformedURLException {
 5.
 6.       Activity activity = new Activity("io.appium.android.apis", "io.appium.an
          droid.apis.preference.PreferenceDependencies");
 7.       ((JavascriptExecutor) driver).executeScript("mobile: startActivity", Immutable
          Map.of("intent","io.appium.android.apis/io.appium.android.apis.preference.Preferen
          ceDependencies"));
 8.         driver.findElement(By.id("android:id/checkbox")).click();
 9.         DeviceRotation landScape = new DeviceRotation(0,0,90);
10.         driver.rotate(landScape);
11.         driver.findElement(By.xpath("(//android.widget.RelativeLayout)[2]")).click();
12.         String alertTitle = driver.findElement(By.id("android:id/alertTitle")).getText();
13.         Assert.assertEquals(alertTitle, "WiFi settings");
14.         driver.setClipboardText("Veronika WiFi");
15.         driver.findElement(By.id("android:id/edit")).sendKeys(driver.getClipboardText());
16.         driver.pressKey(new KeyEvent(AndroidKey.ENTER));
17.         driver.pressKey(new KeyEvent(AndroidKey.BACK));
18.         driver.pressKey(new KeyEvent(AndroidKey.HOME));
19.
20.       }
21. }
```

Obrázek 42. Ukázka zdrojového kódu TC_06 Aktivita a systémové funkce

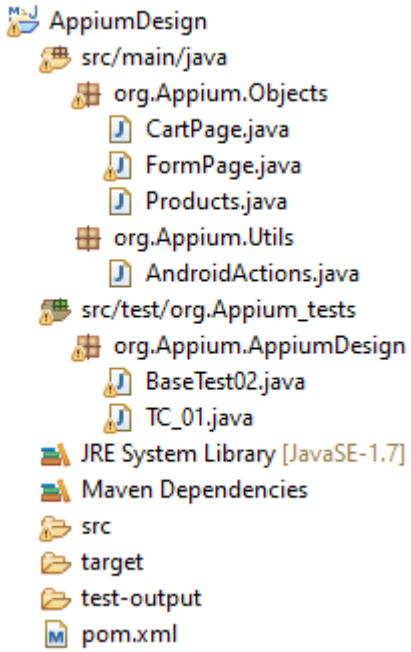


Obrázek 43. Výsledek TC_06 Aktivita a systémové funkce

6.3 Testování General-Store.apk

Aplikace bude testována jedním testem, zaměřeným na lepší organizaci testovacího kódu, a druhým pro testování nativní a webové časti v jednom zdrojovém kódu. U této aplikace je opět základní test **BaseTest02**, který je postaven na stejném principu jako BaseTest01

v předchozí mobilní aplikaci. Ale nejsou zde vytvořeny funkce, ale pouze nastavení a spuštění Appium serveru, konfiguruje zařízení a aplikace.



Obrázek 44. Struktura adresáře

Pro lepší strukturu je vytvořen balík Objects, který je vytvořen na základě návrhového vzoru POM. Obsahuje tři třídy, které představují určité stránky (FromPage, Products, CartPage). Každá třída obsahuje deklaraci jednotlivých elementů, které určitá stránka obsahuje a budou testovány. Dále je zde i deklarace funkcí, které se na dané stránce testují.

```
1. public class FormPage extends AndroidActions {
2.
3.       AndroidDriver driver;
4.
5.       public FormPage (AndroidDriver driver)
6.       {
7.                 super(driver);
8.                 this.driver = driver;
9.                 PageFactory.initElements(new AppiumFieldDecorator(driver), this);
10.      }
11.
12.      @AndroidFindBy(id = "com.androidsample.generalstore:id/nameField")
13.      private WebElement nameField;
14.
15.      @AndroidFindBy(xpath = "//android.widget.RadioButton[@text='Female']")
16.      private WebElement femaleOption;
17.
18.      @AndroidFindBy(xpath = "//android.widget.RadioButton[@text='Male']")
19.      private WebElement maleOption;
20.
21.      @AndroidFindBy(id = "android:id/text1")
22.      private WebElement countrySelection;
23.
24.      @AndroidFindBy(id = "com.androidsample.generalstore:id/btnLetsShop")
25.      private WebElement shopButton;
26.
27.      public void setNameField(String name)
28.      {
29.                nameField.sendKeys(""+name+"");
30.                driver.hideKeyboard();
31.
32.      }
33.
34.      public void setGender(String gender)
35.      {
36.                if (gender.contains("female"))
37.                          femaleOption.click();
38.                else
39.                          maleOption.click();
40.      }
41.
42.      public void SetCountrySelection(String countryName)
43.      {
44.                countrySelection.click();
45.                scrollToText(countryName);
46.
47.
48.                driver.findElement(By.xpath("//android.widget.TextView[@text='"+country
         Name+"']")).click();
49.      }
50.
51.      public Products submitForm()
52.      {
53.                shopButton.click();
54.                return new Products(driver);
55.
56.      }
57.  }
```

Obrázek 45. Ukázka třídy FormPage

Každá třída z Objects dědí základní funkce z třídy AndroidActions, ve které se nachází základní funkce. Tyto funkce jsou z většiny převzaty z BaseTest01, ale objevují se zde i některé jako *scrollCookies()* (posunutí na konec cookies), *getFormattedAmount()* (formátování textu) a *scrollToText()* (posunutí na daný text).

```
1.      public void scrollCookies() {
2.          boolean canScrollMore;
3.          do {
4.              canScrollMore = (Boolean) ((JavascriptExecutor) driver).executeScript("mo
        bile: scrollGesture", ImmutableMap.of(
5.                      "left", 660,
6.                      "top", 1300,
7.                      "width", 100,
8.                      "height", 500,
9.                      "direction", "down",
10.                     "percent", 30.0
11.             ));
12.         } while(canScrollMore);
13.     }
14.
15.     public void scrollToText(String text)
16.     {
17.             driver.findElement(AppiumBy.androidUIAutomator("new UiScrollable(new UiS
        elector()).scrollIntoView(text(\""+text+"\"));"));
18.     }
19.
20.     public Double getFormattedAmount(String amount)
21.     {
22.             Double price = Double.parseDouble(amount.substring(1));
23.             return price;
24.     }
```

Obrázek 46. Vybrané funkce ve zdrojovém kódu AndroidActions

Výhoda této struktury spočívá v lepší organizaci kódu a čitelnosti, snadnější údržbě, snížení duplicit a odolnosti vůči změnám v uživatelském rozhraní.

6.3.1 TC_01 Nativní část

Tento testovací případ testuje průchod napříč celou nativní částí aplikace. Využívá výše zmíněné třídy a vytváří celý spustitelný test.

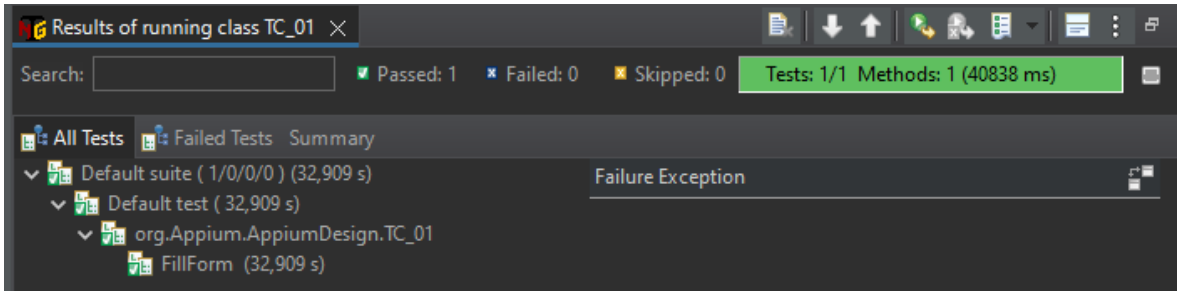
Tabulka 13. Specifikace TC_01 Nativní část

Název	TC_01 Nativní část
Popis	Testovací případ obsahuje průchod celou nativní částí aplikace.
Očekávaný výsledek	Úspěšné vyplnění formuláře, přidání produktů do košíku, suma v košíku souhlasí, checkbox je zaškrtnut a tlačítko pro přechod na webovou stránku je *clickable*.
Pre-conditions	Spuštěný server, emulátor a aplikace
Testovací kroky	1. Vyplnění textu ve formuláři. 2. Nastavení pohlaví. 3. Výběr země v rozbalovacím seznamu. 4. Kliknutí na tlačítko „Let's Shop". 5. Přidání prvního produktu do košíku. 6. Přidání dalšího produktu do košíku. 7. Kliknutí na ikonu košíku v pravém horním rohu. 8. Kontrola, zda suma produktů v košíku odpovídá zobrazené sumě. 9. Zaškrtnutí checkboxu. 10. Kliknutí na tlačítko „Visit to the website to complete purchase".
Vstupní data	1. Validní a. Kliknutí na správné tlačítka ve správném pořadí. b. Name input: Veronika c. Nastavení pohlaví na „female". d. Výběr země Česká republika. e. Přidání prvního a druhého produktu do košíku. 2. Nevalidní a. Kliknutí na nesprávné místo/tlačítko nebo v nesprávném pořadí. b. Name input zůstane prázdný.

	c. Není přidán žádný produkt do košíku. d. Stisknutí kláves v nesprávném pořadí.

```
1. public class TC_01 extends BaseTest02 {
2.
3.
4.   @Test
5.   public void FillForm() throws InterruptedException
6.   {
7.     formPage.setNameField("Veronika");
8.     formPage.setGender("female");
9.     formPage.SetCountrySelection("Czech Republic");
10.    Products products = formPage.submitForm();
11.    products.addItemsToCartByIndex(0);
12.    products.addItemsToCartByIndex(0);
13.    CartPage cartPage = products.goToCartPage();
14.    double totalSum = cartPage.getProductSum();
15.    double displayFormattedSum = cartPage.getTotalAmountDisplayed();
16.    Assert.assertEquals(totalSum, displayFormattedSum);
17.    cartPage.acceptTerms();
18.    cartPage.submitOrder();
19.    Thread.sleep(3000);
20.  }
21. }
```

Obrázek 47. Ukázka zdrojového kódu TC_01 Nativní část



Obrázek 48. Výsledek TC_01 Nativní část

6.3.2 TC_02 End-To-End

Druhý testovací případ vychází z předchozího TC_01. Zaměřuje se na přepínaní mezi nativní částí a částí webovou pomocí metody *context()*. Pro identifikaci elementů ve webové aplikaci se používá počítačová verze Google Chrome, kdy pomocí tlačítka prozkoumat zjistím informace o daném elementu.

První krok pro toto testování obsahuje kód pro vypsání názvu nativní a webové části dané aplikace. (Obrázek 49. Ukázka zdrojového kódu pro získání názvu nativní a webové aplikace Díky této informaci můžeme měnit context z nativní na webovou část.

```
1. Set<String>contexts = driver.getContextHandles();
2. for(String contextName:contexts)
3. {
4.         System.out.println(contextName);
5. }
```

Obrázek 49. Ukázka zdrojového kódu pro získání názvu nativní a webové aplikace

Jakmile se provede část z TC_01, přesune se test do webové aplikace, kde provádí testy. Jako poslední se přepíná context zpět na nativní část.

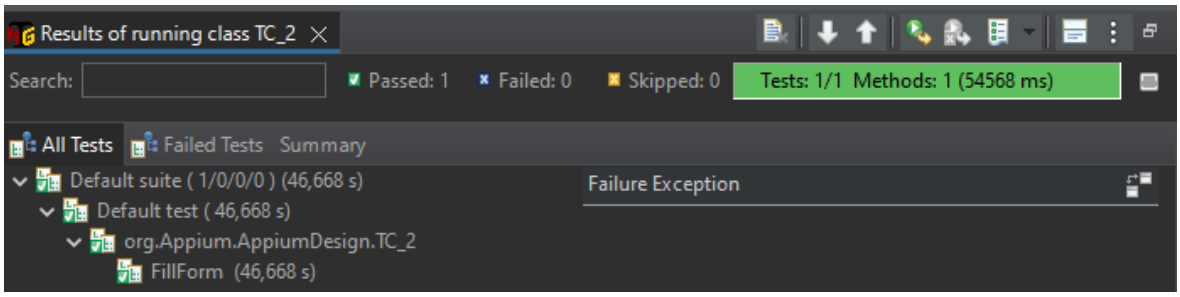
Tabulka 14. Specifikace TC_02 End-To-End

Název	TC_02 End-To-End
Popis	Testovací případ obsahuje průchod celou aplikací včetně webové části.
Očekávaný výsledek	Přepnutí na nativní context a zobrazení stránky FormPage.
Pre-conditions	Spuštěný server, emulátor a aplikace
Testovací kroky	1. Vyplnění textu ve formuláři. 2. Nastavení pohlaví. 3. Výběr země v rozbalovacím seznamu. 4. Kliknutí na tlačítko „Let's Shop". 5. Přidání prvního produktu do košíku. 6. Přidání dalšího produktu do košíku. 7. Kliknutí na ikonu košíku v pravém horním rohu. 8. Kontrola, zda suma produktů v košíku odpovídá zobrazené sumě. 9. Zaškrtnutí checkboxu. 10. Kliknutí na tlačítko „Visit to the website to complete purchase". 11. Odkliknutí cookies. 12. Vložení textu do vyhledávače. 13. Stisknutí klávesy „Enter". 14. Stisknutí klávesy „Back".
Vstupní data	3. Validní a. Kliknutí na správná tlačítka ve správném pořadí. b. Name input: Veronika

	c. Nastavení pohlaví na „female“. d. Výběr země Česká republika. e. Přidání prvního a druhého produktu do košíku. f. Vyhledávač input: utb 4. Nevalidní a. Kliknutí na nesprávné místo/tlačítko nebo v nesprávném pořadí. b. Name input zůstane prázdný. c. Není přidán žádný produkt do košíku. d. Stisknutí kláves v nesprávném pořadí.

```
1.                  driver.context("WEBVIEW_com.androidsample.generalstore");
2.                  scrollCookies();
3.                  driver.findElement(By.xpath("//div[@role='none'][contains(text(),'Přijmout
                    vše')]")).click();
4.                  driver.findElement(By.name("q")).sendKeys("utb");
5.                  driver.findElement(By.name("q")).sendKeys(Keys.ENTER);
6.                  driver.pressKey(new KeyEvent(AndroidKey.BACK));
7.                  driver.context("NATIVE_APP");
```

Obrázek 50. Ukázka zdrojového kódu TC_02 End-To-End (context)



Obrázek 51. Výsledek TC_02 End-To-End

ZÁVĚR

V této bakalářské práci jsem se zaměřila na porovnání nástrojů pro automatizované testování mobilních aplikací, což je klíčový prvek zajištění kvality softwaru v rychle se rozvíjející oblasti mobilních technologií.

Teoretická část práce poskytla nejen úvod do základních pojmů a významu testování softwaru, ale také pohled na různé typy mobilních aplikací, včetně nativních, webových a hybridních. Tato analýza byla zásadní pro porozumění různých testovacích potřeb, které jsou nutné pro praktickou část.

V praktické části této práce jsem se zaměřila na porovnání několika nástrojů, které se používají pro automatizované testování mobilních aplikací. Cílem bylo zjistit, které nástroje jsou nejpřívětivější pro uživatele, jak dobře se dají zapojit do běžně používaných programovacích prostředí a jak efektivně umožňují testovat různé druhy mobilních aplikací. V práci jsem uvedla také ukázky kódů a popis implementace těchto nástrojů.

Appium se ukázalo jako ideální řešení pro různé platformy a typy aplikací a je vhodné pro začínající testery. Espresso, navržené pro Android aplikace je přímo integrováno v Android Studiu a poskytuje rychlé testování přímo v IDE, což je nejlepší volba pro Android vývojáře. Selendroid z mého porovnání nevyšel zrovna nejlépe, vzhledem k omezené podpoře a zastavenému vývoji. Přesto se jej mohou využít zkušení testeři, kteří potřebují otestovat aplikaci na starších verzí Androidu. UI Automator je ideální pro black-box testování UI na platformě Android. Je velmi vhodnou volbou pro testery, kteří potřebují otestovat jak UI, tak i systémové funkce. Na druhé straně, XCTest je preferován vývojáři iOS aplikací, kde jeho integrace s XCode poskytuje rychlý a jednoduchý proces testování a jako iOS vývojář bych s použitím tohoto frameworku neváhala.

Více jsem se zaměřila na framework Appium, kde jsem uvedla detailní příklady kódů, které demonstrují jeho implementaci a ukazují, jak s frameworkem pracovat v reálných testovacích scénářích.

Během srovnávací analýzy jsem zjistila, že každý nástroj má své ideální využití v závislosti na typu a cílové platformě aplikace. Výběr správného nástroje může zásadně ovlivnit rychlost vývoje aplikace a celkovou kvalitu finálního produktu. Díky této práci máme nyní lepší přehled o tom, jaký nástroj by byl nejvhodnější pro konkrétní testovací potřeby našich budoucích projektů.

SEZNAM POUŽITÉ LITERATURY

[1] *What is Software Testing? How to Perform and Best Practices*, 2004. Online. Testsigma. Dostupné z: https://testsigma.com/guides/software-testing/. [cit. 2024-02-01].

[2] *Co je testování softwaru*, 2001. Online. Skillmea. Dostupné z: https://skillmea.cz/blog/co-je-testovanie-softveru. [cit. 2024-02-01].

[3] RADEK, Radek. *Co je testování softwaru?* Online. Kitner. Dostupné z: https://skillmea.cz/blog/co-je-testovanie-softveru. [cit. 2024-02-01].

[4] ZITA, Stanislav. *Lekce 3 - Testování ve VB.NET - Dokončení unit testů a best practices*. Online. ITnetwork.cz. Dostupné z: https://www.itnetwork.cz/vbnet/testovani/testovani-ve-vbnet-dokonceni-unit-testu-a-best-practices. [cit. 2024-02-01].

[5] TANK, Uday, 2023. *Verification vs Validation Testing: Key Differences*. Online. Testsigma. Dostupné z: https://testsigma.com/blog/verification-vs-validation-testing/#Difference_Between_Verification_vs_Validation_Testing. [cit. 2024-02-01].

[6] DEVARAJ, Kiruthika, 2023. *Guide to Validation Testing in Software Testing*. Online. Testsigma. Dostupné z: https://testsigma.com/blog/validation-testing/#What_is_Validation_Testing. [cit. 2024-02-01].

[7] *Verification and Validation in Software Engineering*, 2023. Online. GeeksforGeeks. Dostupné z: https://www.geeksforgeeks.org/software-engineering-verification-and-validation/. [cit. 2024-02-01].

[8] *WHAT IS RELIABILITY*. Online. ASQ. Dostupné z: https://asq.org/quality-resources/reliability. [cit. 2024-02-01].

[9] *What are the most common software quality and reliability metrics and how do you measure them*. Online. LinkedIn. Dostupné

z: https://www.linkedin.com/advice/0/what-most-common-software-quality-reliability. [cit. 2024-02-03].

[10] DYSON, Jonathan, 2019. *Conjoining FURPS and MoSCoW to Analyse and Prioritise Requirements*. Online. LinkedIn. Dostupné z: https://www.linkedin.com/pulse/conjoining-furps-moscow-analyse-prioritise-jonathan-dyson. [cit. 2024-02-06].

[11] *Software Supportability*, 2004. Online. Dostupné z: https://www.software-supportability.org/Supportability.html. [cit. 2024-02-03].

[12] DYSON, Jonathan. *Concepts: Requirements*. Online. Tesestec. Dostupné z: https://www.tesestec.com.br/pasteurjr/rup/process/workflow/requirem/co_req.htm#Design%20Requirement. [cit. 2024-02-07].

[13] GANESH, S.G., 2012. *Joy of Programming: Why is a Software Glitch Called a 'Bug'?* Online. Open Source For You. Dostupné z: https://www.opensourceforu.com/2012/03/joy-of-programming-why-software-glitch-called-bug/. [cit. 2024-02-08].

[14] *Software Testing – Bug vs Defect vs Error vs Fault vs Failure*, 2023. Online. GeeksForGeeks. Dostupné z: : https://www.geeksforgeeks.org/software-testing-bug-vs-defect-vs-error-vs-fault-vs-failure/. [cit. 2024-02-08].

[15] PATTON, Ron, 2001. *Software Testing*. Online. Sams Publishing. ISBN 0-672-31983-7. Dostupné z: https://dl.icdst.org/pdfs/files3/aede5f4a7bd951f08d6b4711beb59e42.pdf. [cit. 2024-02-10].

[16] VASYLYNA, Natallia, 2022. *Types of Bugs in Software Testing*. Online. QATestLab. Dostupné z: https://blog.qatestlab.com/2011/03/24/3-types-of-bugs-in-software/. [cit. 2024-02-10].

[17] *Co je to Bug?*, 2023. Online. IT Slovník.cz. Dostupné z: https://it-slovnik.cz/pojem/bug. [cit. 2024-02-10].

[18] ORAKZAI, Rahmat Ullah, 2023. *Software Testing: Defect, Bug, Error, and Failure*. Online. Baeldung. Dostupné z: https://www.baeldung.com/cs/software-testing-defect-bug-error-and-failure. [cit. 2024-02-10].

[19] *Pojem dominový efekt*. Online. ABZ.cz. Dostupné z: Dostupné z: https://slovnik-cizich-slov.abz.cz/web.php/slovo/dominovy-efekt. [cit. 2024-02-11].

[20] *The True Cost of a Software Bug: Part One*. Online. Celerity. Dostupné z: https://www.celerity.com/insights/the-true-cost-of-a-software-bug. [cit. 2024-02-10].

[21] *Černá vs. bílá skříňka*. Online. Testování software. Dostupné z: http://test.swtestovani.cz/index.php?option=com_content&view=article&id=23:erna-vs-bila-skika&catid=3:zaklady&Itemid=11. [cit. 2024-02-15].

[22] *What Is Automated Testing? Ultimate Guide*. Online. Testsigma. Dostupné z: https://testsigma.com/automated-testing. [cit. 2024-02-18].

[23] *Complete Guide to Manual Testing*. Online. Testsigma. Dostupné z: https://testsigma.com/guides/manual-testing/#What_is_Manual_Testing. [cit. 2024-02-18].

[24] DA SILVA, Michael, 2022. *Pros and Cons of Automated Testing*. Online. UIlicous. Dostupné z: https://uilicious.com/blog/pros-cons-automated-testing/. [cit. 2024-02-19].

[25] *Automatizované a manuální testování aplikací – pro a proti*, 2018. Online. Pixelfield. Dostupné z: https://pixelfield.cz/blog/automatizovane-a-manualni-testovani/. [cit. 2024-02-19].

[26] PATEL, Himanshu, 2023. *What is Mobile Automation Application Testing Process? – Check out the De-tails*. Online. Concetto Labs. Dostupné z: https://www.concettolabs.com/blog/mobile-automation-application-testing-process/. [cit. 2024-02-19].

[27] *Complete Guide to Manual Testing*. Online. Testsigma. Dostupné z: https://testsigma.com/guides/manual-testing/#Manual_Testing_Goals. [cit. 2024-02-19].

[28] BARTOSIŃSKA, Inez, 2024. *What Is a Mobile App – All You Should Know as a Future Product Owner*. Online. Droids On Roids. Dostupné z: https://www.thedroidsonroids.com/blog/what-is-a-mobile-app. [cit. 2024-02-25].

[29] *What Is a Mobile App – All You Should Know as a Future Product Owner*, 2024. Online. Indeed. Dostupné z: https://uk.indeed.com/career-advice/career-development/what-is-mobile-app. [cit. 2024-02-25].

[30] KING, Mike, 2022. *What is a mobile app? (With definition, types and examples)*. Online. Indeed. Dostupné z: https://www.businessofapps.com/app-developers/research/types-of-mobile-apps/. [cit. 2024-02-25].

[31] *Main Categories and Types of Mobile Apps [2023 Update]*, 2023. Online. Bamboo Agile. Dostupné z: https://www.businessofapps.com/app-developers/research/types-of-mobile-apps/. [cit. 2024-02-25].

[32] *Nativní (nativus)*. Online. Infoz.cz. Dostupné z: https://www.infoz.cz/nativni/. [cit. 2024-02-25].

[33] *Nativní mobilní vývoj*. Online. DAMI development. Dostupné z: https://www.damidev.com/slovnik/nativni-mobilni-vyvoj. [cit. 2024-02-25].

[34] *Flutter*. Online. Dostupné z: https://flutter.dev. [cit. 2024-04-29].

[35] *Co je Xamarin?*, 2023. Online. Microsoft. Dostupné z: https://learn.microsoft.com/cs-cz/xamarin/get-started/what-is-xamarin. [cit. 2024-02-25].

[36] Online. React Native. Dostupné z: https://reactnative.dev. [cit. 2024-02-25].

[37] MAHAJAN, Parth, 2022. *Hybrid Application: Native-like Experience with WebView*. Online. Medium. Dostupné z: https://medium.com/1mgofficial/hybrid-application-native-like-experience-with-webview-9896f61881cb. [cit. 2024-02-28].

[38] *Druhy mobilních aplikací: výhody, nevýhody a jak vybrat*. Online. Creative Handles. Dostupné z: https://creativehandles.com/cs/blogove-prispevky/125/druhy-mobilnich-aplikaci-vyhody-nevyhody-a-jak-vybrat. [cit. 2024-02-28].

[39] *Hybrid app*. Online. Appsflyer. Dostupné z: https://www.appsflyer.com/glossary/hybrid-app/. [cit. 2024-02-28].

[40] *11 Best Mobile Automation Testing Tools*. Online. Testsigma. Dostupné z: https://testsigma.com/blog/mobile-test-automation-frameworks/. [cit. 2024-03-01].

[41] PAI, Akshay, 2023. *Appium Tutorial for Mobile Application Testing*. Online. Browserstack. Dostupné z: https://www.browserstack.com/guide/appium-tutorial-for-testing. [cit. 2024-03-01].

[42] *How Does Appium Work?*, 2024. Online. Appium. Dostupné z: https://appium.io/docs/en/latest/intro/appium/. [cit. 2024-03-02].

[43] *Appium Drivers*, 2024. Online. Appium. Dostupné z: https://appium.io/docs/en/2.3/ecosystem/drivers/. [cit. 2024-03-02].

[44] *Intro to Appium Drivers*, 2024. Online. Appium. Dostupné z: https://appium.io/docs/en/latest/intro/drivers/. [cit. 2024-03-02].

[45] *Espresso basics*. Online. Developers. Dostupné z: https://developer.android.com/training/testing/espresso/basics. [cit. 2024-03-06].

[46] *Espresso Web*. Online. Developers. Dostupné z: https://developer.android.com/training/testing/espresso/web. [cit. 2024-03-06].

[47] *Espresso Testing Framework - Architecture*. Online. Tutorials Point. Dostupné z: https://www.tutorialspoint.com/espresso_testing/espresso_testing_architecture.htm. [cit. 2024-03-06].

[48] *Espresso basics*. Online. Developers. Dostupné z: https://developer.android.com/training/testing/espresso/basics. [cit. 2024-03-06].

[49] *Selendroid Tutorial: Android Mobile Test Automation Framework (Part 1)*, 2024. Online. Software Testing Help. Dostupné z: https://www.softwaretestinghelp.com/selendroid-tutorial-1/. [cit. 2024-03-15].

[50] *Comparing mobile automation tools - Appium, Selendroid and Calabash*, 2023. Online. QA Mmadness. Dostupné z: https://www.qamadness.com/comparing-mobile-automation-tools-appium-selendroid-and-calabash/. [cit. 2024-03-15].

[51] *Using Selendroid To Automate The User Interactions Over A Mobile App (Part 2)*, 2024. Online. Soft. Dostupné z: https://www.softwaretestinghelp.com/selendroid-tutorial-2. [cit. 2024-03-15].

[52] KUMAR, Rajesh, 2023. *Top 50 Selendroid interview questions and answers*. Online. DevOpsSchool. Dostupné z: https://www.devopsschool.com/blog/top-50-selendroid-interview-questions-and-answers/. [cit. 2024-03-15].

[53] *Appium vs Selendroid*, 2023. Online. GeeksforGeeks. Dostupné z: https://www.geeksforgeeks.org/appium-vs-selendroid/. [cit. 2024-03-15].

[54] *Selendroid's Architecture*. Online. Selendroid. Dostupné z: http://selendroid.io/architecture.html. [cit. 2024-03-15].

[55] *Launching Selendroid*. Online. Selendroid. Dostupné z: http://selendroid.io/setup.html#launchingSelendroid. [cit. 2024-04-20].

[56] *Mobile Testing - Selendroid Framework*. Online. Tutorials Point. Dostupné z: https://www.tutorialspoint.com/mobile_testing/mobile_testing_selendroid_framework.htm. [cit. 2024-04-20].

[57] *Write automated tests with UI Automator*, 2024. Online. Developers. Dostupné z: https://developer.android.com/training/testing/other-components/ui-automator#java. [cit. 2024-03-20].

[58] ABOYI, Pius, 2022. *UIAutomator vs. Espresso: Which Is Better?* Online. Waldo. Dostupné z: https://www.waldo.com/blog/uiautomator-vs-espresso. [cit. 2024-03-20].

[59] *UiObject2*, 2024. Online. Developers. Dostupné z: https://developer.android.com/reference/androidx/test/uiautomator/UiObject2. [cit. 2024-03-20].

[60] *BySelector*, 2024. Online. Developers. Dostupné z: https://developer.android.com/reference/androidx/test/uiautomator/BySelector. [cit. 2024-03-20].

[61] *By*, 2024. Online. Developers. Dostupné z: https://developer.android.com/reference/androidx/test/uiautomator/By. [cit. 2024-03-20].

[62] *Test Uiautomator*. Online. Developers. 2024. Dostupné z: https://developer.android.com/jetpack/androidx/releases/test-uiautomator. [cit. 2024-04-30].

[63] *XCTest*, 2024. Online. Developer. Dostupné z: https://developer.apple.com/documentation/xctest. [cit. 2024-03-30].

[64] *Xcode 15*. Online. Developer. Dostupné z: https://developer.apple.com/xcode/. [cit. 2024-03-30].

[65] *XCTest: A Complete Guide*, 2022. Online. HeadSpin. Dostupné z: https://www.headspin.io/blog/xctest-a-complete-guide. [cit. 2024-03-30].

[66] PAI, Akshay, 2023. *Getting Started with XCUITest : UI Automation Framework on iOS*. Online. BrowerStack. Dostupné z: https://www.browserstack.com/guide/getting-started-xcuitest-framework. [cit. 2024-04-01].

[67] USLU, Amil, 2021. *XCUITest Tutorial – iOS Testing Framework Guide*. Online. SW TEST ACADEMY. Dostupné z: https://www.swtestacademy.com/xcuitest-tutorial-ios-testing-framework-guide/. [cit. 2024-04-03].

[68] *Xcuitest-sample-browserstack*, 2023. Online. GitHub. Dostupné z: https://github.com/browserstack/xcuitest-sample-browserstack. [cit. 2024-04-03].

[69] *Downloading Jenkins*. Online. Jenkins. Dostupné z: https://www.jenkins.io/download/. [cit. 2024-04-28].

[70] TUMMALA, Madhusudhana. *Can Selenium be used for mobile testing?* Online. Get Software Service. Dostupné z: https://www.getsoftwareservice.com/selenium-can-be-used-for-mobile-testing/. [cit. 2024-04-05].

[71] Sruthy, 2024. *UIAutomatorViewer Tutorial: Inspect Elements On Android*. Online. SoftwareTestingHelp. Dostupné z: https://www.softwaretestinghelp.com/uiautomator-tutorial/. [cit. 2024-04-05].

[72] *Integrate BrowserStack App Automate with Jenkins*. Online. BrowerStack. Dostupné z: https://www.browserstack.com/docs/app-automate/appium/integrations/jenkins. [cit. 2024-04-28].

[73] AKHTAR, Hamid, 2023. *How to perform Parallel Test Execution in Appium?* Online. BrowerStack. Dostupné z: https://www.browserstack.com/guide/parallel-test-execution-appium. [cit. 2024-04-28].

[74] *About Continuous Integration in Xcode*. Online. Developer. Dostupné z: https://developer.apple.com/library/archive/documentation/IDEs/Conceptual/xcode_guide-continuous_integration/index.html. [cit. 2024-04-28].

[75] *Jenkins with Espresso*. Online. Perforce. Dostupné z: https://help.perfecto.io/perfecto-help/content/perfecto/integrations/jenkins_with_espresso.htm. [cit. 2024-04-28].

[76] *Espresso test reporting*. Online. Tesult. Dostupné z: https://www.tesults.com/docs/espresso. [cit. 2024-04-28].

[77] *Espresso test reporting*. Online. Tesult. Dostupné z: https://www.tesults.com/docs/espresso. [cit. 2024-04-30].

[78] *Scaling Selendroid by using the Selenium Grid*. Online. Selendroid. Dostupné z: http://selendroid.io/scale.html. [cit. 2024-04-28].

[79] 2022. Online. GitHub. Dostupné z: https://github.com/appium/appium/tree/master/packages/appium/sample-code/apps. [cit. 2024-04-07].

[80] 2022. Online. GitHub. Dostupné z: https://github.com/kmieshkov/appium-General-Store.apk/tree/main/src/test/resources. [cit. 2024-04-07].

[81] *Run apps on the Android Emulator*, 2023. Online. Developers. Dostupné z: https://developer.android.com/studio/run/emulator. [cit. 2024-04-07].

[82] Online. MvnRepository. Dostupné z: https://mvnrepository.com. [cit. 2024-04-07].

[83] LAWRENCE, Brittney, 2024. *How to Use the Appium Inspector*. Online. Kobit. Dostupné z: https://kobiton.com/blog/how-to-use-the-appium-inspector/. [cit. 2024-04-07].

[84] *How to find App Package and App Activity of your Android App*, 2020. Online. Testsigma. Dostupné z: https://support.testsigma.com/support/solutions/articles/32000019977-how-to-find-app-package-and-app-activity-of-your-android-app. [cit. 2024-04-07].

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

FURPS	Functionality, Usability, Reliability, Performance, Security
FURPS+	Functional, Usability, Reliability, Performance, Supportability a rozšíření (+)
CPU	Central Processing Unit
$	Znak amerického dolaru
HTML	HyperText Markup Language
CSS	Application Programming Interface
GPS	Global Positioning System
API	Application Programming Interface
apk	Android application PacKage
UI	User Interface
JDK	Java Development Kit
IDE	Integrated Development Environment
SDK	Software Development Kit
CI	Continuous Integration
XML	Extensible Markup Language
API	Application Programming Interface
GUI	Graphical User Interface
atd	A tak dále
POM	Project Object Model
npm	Node Package Manager

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: CD

Příloha obsahuje text bakalářské práce v PDF a projekty z jednotlivých frameworků.