EKONOMICKÁ FAKULTA TUL

Bakalářská práce

Automatizované testování frontend aplikace v automotive

Studijní program:	B0688P140002 Informační management
Autor práce:	**Petr Motl**
Vedoucí práce:	Ing. David Kubát, Ph.D., Ing.Paed.IGIP
	Katedra informatiky

Liberec 2024

EKONOMICKÁ FAKULTA TUL

Zadání bakalářské práce

Automatizované testování frontend aplikace v automotive

Jméno a příjmení:	**Petr Motl**
Osobní číslo:	E21000231
Studijní program:	B0688P140002 Informační management
Zadávající katedra:	Katedra informatiky
Akademický rok:	2023/2024

Zásady pro vypracování:

1. Popis metod a prostředí používaných v automotive.
2. Orientace v mobilní aplikaci a jejích vlastnostech.
3. Tvorba testu pro mobilní zařízení.
4. Uvedení navrženého testu do praxe.
5. Vyhodnocení vytvořeného řešení.

Rozsah grafických prací:
Rozsah pracovní zprávy: 30 normostran
Forma zpracování práce: tištěná/elektronická
Jazyk práce: čeština

Seznam odborné literatury:

- GRAHAM, Dorothy, Rex BLACK a Erik VAN VEENENDAAL, 2020. *Foundations of software testing: ISTQB certification*. Fourth edition. Andover, Hampshire: Cengage. ISBN 978-1-4737-6479-8.
- MYSLÍN, Josef, 2016. *Scrum: průvodce agilním vývojem softwaru*. Brno: Computer Press. ISBN 978-80-251-4650-7.
- NATNAEL, Gonfa Berihun, Cyrille DONGMO a der Poll JOHN ANDREW VAN, 2023. The Applicability of Automated Testing Frameworks for Mobile Application Testing: A Systematic Literature Review. *Computers* [online], vol. 12, no. 5, s. 97. ISSN 2073-431X.
- PECINOVSKÝ, Rudolf, 2022. *Začínáme programovat v jazyku Python*. 2. přepracované a rozšířené vydání. Praha: Grada Publishing. ISBN 978-80-271-3609-4.
- SHAW, Zed A., 2017. *Learn Python 3 the hard way: a very simple introduction to the terrifyingly beautiful world of computers and code*. Boston: Addison-Wesley. Zed Shaw's hard way series. ISBN 978-0-13-469288-3.

Konzultant: Ing. Martin Tahadlo, IT Business Analyst – ConnectedCar, Škoda Auto a.s.

Vedoucí práce: Ing. David Kubát, Ph.D., Ing.Paed.IGIP
Katedra informatiky

Datum zadání práce: 1. listopadu 2023
Předpokládaný termín odevzdání: 31. srpna 2025

L.S.

doc. Ing. Aleš Kocourek, Ph.D.
děkan

Mgr. Tereza Semerádová, Ph.D.
garant studijního programu

V Liberci dne 1. listopadu 2023

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Jsem si vědom toho, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má bakalářská práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

30. dubna 2024 Petr Motl

Automatizované testování frontend aplikace v automotive

Anotace

Bakalářská práce se zabývá návrhem, tvorbou a vývojem automatizovaného testu pro mobilní aplikaci sloužící ke vzdálené obsluze automobilu. Teoretická část se věnuje základním pojmům a strukturám - aplikaci, back-endu, front-endu, testování software, typům testů, testování v automotive, vývojovým prostředím a automatizovanému a manuálnímu testování. Praktická část se částečně věnuje pozadí bakalářské práce, zde zejména oddělení, kde byla vykonávána praxe a její hlavní část je věnována analýze testovacího scénáře, vlastnímu vývoji testu, řešením problémů, které každý nově vznikající software provází a nakonec automatizaci testu. Pozornost bude věnována rovněž rozdílnosti operačních systémů a případně obtížím, kterými tyto rozdíly ovlivňují vývoj testu. Nakonec budou zmíněny a zhodnoceny výsledky ostrého nasazení testu v praxi a bude odhadnut případný finanční benefit zadavatele.

Klíčová slova

Aplikace, automotive, backend, frontend, testování software

Automated testing of frontend application in automotive

Annotation

The bachelor thesis deals with the design, creation and development of an automated test for a mobile application used for remote operation of a car. The theoretical part deals with basic concepts and structures - application, back-end, front-end, software testing, types of tests, testing in automotive, development environments and automated and manual testing. The practical part is partly devoted to the background of the Bachelor's thesis, here in particular the department where the practice was carried out, and its main part is devoted to the analysis of the test scenario, the actual test development, the solutions to the problems that accompany each new software development and finally the test automation. Attention will also be paid to the differences in operating systems and, where relevant, the difficulties that these differences have in test development. Finally, the results of a live deployment of the test will be discussed and evaluated, and the potential financial benefit to the company will be estimated.

Key Words

Application, automotive, backend, frontend, software testing

Poděkování

Na tomto místě bych rád ze všech nejvíce poděkoval své mamince, Andree Motlové, za neutuchající lásku, podporu a nekonečnou trpělivost po celou dobu mého studia. Zároveň děkuji svému bratru Vildovi a sestře Alžbětě za vstřícnost a pochopení. V neposlední řadě bych rád poděkoval za rady a obrovskou pomoc Janu Horsákovi. Bez vás bych to nikdy nezvládl.

Děkuji tímto i Ing. Davidu Kubátovi, Ph.D., Ing.Paed.IGIP, mému vedoucímu práce a Ing. Martinu Tahadlovi, mému konzultantovi za odborné vedení, cenné rady a pomoc během mé bakalářské práce. Můj dík patří i kolegům z oddělení za sdílení zkušeností a přátelskou atmosféru na pracovišti.

Obsah

Seznam ilustrací (obrázků) **13**

Seznam tabulek **14**

Seznam použitých zkratek, značek a symbolů **15**

Úvod **16**

1 Teoretická část **17**

1.1 Aplikace 17

1.1.1 Front-endová část aplikace 17

1.1.2 Back-endová část aplikace 18

1.2 Testování software 18

1.2.1 Testování 20

1.2.2 Manuální testování 20

1.2.3 Automatizované testování 20

1.2.4 Rozdíl mezi automatizovaným testováním a manálním testováním 21

1.2.5 Typy testování 22

1.2.6 Cíle testování 26

1.2.7 Filozofie testování 27

1.2.8 Metodiky testování 28

1.2.9 Testování v automotive 29

1.3 Software pro vývojáře 30

1.3.1 Visual studio code 31

1.3.2 Pycharm 33

1.3.3 Volba vývojového prostředí 34

1.3.4 Gitlab 34

1.3.5 Jenkins 35

1.3.6 BrowserStack 35

2 Praktická část **37**

2.1 Oddělení firmy 37

2.2 Testovaná aplikace 38

2.2.1 Testovaná funkce 38

2.3 Vytvoření analýzy 39

2.3.1 Konzultace se Splunk specialisty 41

2.4 Vývoj 42

2.5 Zjištění ID elementů aplikace 43
2.5.1 Appium Inspector 43
2.5.2 Nastavení Appium Inspector 44
2.5.3 Používání programu Appium Inspector 45
2.6 Tvorba kódu 46
2.7 Další užité soubory 51
2.8 Lokální spuštění testu 52
2.9 Nahrání na GitLab 52
2.10 Automatizace testu 53
2.11 Spuštění na testovacím serveru 54
2.12 Spuštění testu 54
2.13 Očištění výsledků 55
2.14 Vyhodnocení výsledků 56
Závěr **58**
Seznam použité literatury **59**

Seznam ilustrací (obrázků)

Obrázek 1: Cíle testování 27
Obrázek 2: V model 29
Obrázek 3:Visual Studio Code 32
Obrázek 5:Analýza testovacího scénáře 40
Obrázek 6: Kostra kódu 43
Obrázek 7: Nastavení Appium Inspector 44
Obrázek 8: Appium Inspector 45
Obrázek 9:Vyplnění kostry kódu 46
Obrázek 10: Nastavení času na operačním systému Android 47
Obrázek 11:Nastavení času na operačním systému IOS 48
Obrázek 12: Podmínka if/else 50
Obrázek 13: Soubor helpers 51
Obrázek 14: Soubor elements 52
Obrázek 15: Lokální spuštění testu 52

Seznam tabulek

Seznam použitých zkratek, značek a symbolů

API	Application Programming Interface
CAN	Controller Area Network
CD	Continuous delivery
CFB	Connected fleet backend
CI	Continuous integration
FAPI	Frontend API
HTTP	Hypertext Transfer Protocol
IDE	Integrated development environment
NOK	Not Ok
OCU	Operational Conversion Unit
QR	Quick Response Code
URL	Uniform Resource Locator

Úvod

Tablet, či chytrý telefon se v posledních letech stal hojně užívaným prostředkem ve většině domácností. Jedním z hlavních důvodu je prudký vývoj a rozšíření technologií mobilními zařízeními sledovanými, kontrolovanými, nebo přímo aktivně ovládanými. Může jít o jednoduché ovládání typu rozsviť-zhasni, nebo o ovládání velmi složitých a komplexních celků, jako je v našem případě osobní automobil. Zejména u těchto složitých zařízení je důležité, aby aplikace sloužící k jeho ovládání beze zbytku naplňovala základní požadavky na ni kladené, tedy bezpečnost, přesnost, úplnosti a kvalitu. K tomu, abychom tyto vlastnosti byli schopni zajistit, slouží testování aplikací a testovací software.

A právě vývoji testovacího software je věnována předkládaná práce. Přenesme se tedy do prostředí významného výrobce automobilů. Úkolem je vyvinout automatizovaný test části aplikace sloužící ke spouštění ovládání vyhřívání jednoho z posledních modelů této automobilky. Software byl sice vyvíjen jako téma bakalářské práce, ale ve finále má konkrétní dopad a využití a dočkal se ostrého nasazení v kontrole výrobního procesu.

1 Teoretická část

1.1 Aplikace

Aplikace neboli aplikační program je počítačový program, který používá uživatel počítače k ovládání a/nebo ke komunikaci s dalšími zařízeními, ať již se samotným počítačem (notebookem, tabletem, telefonem…), serverem, nebo – v dnešní době chytrých zařízení – například s automobilem, pračkou, myčkou nebo dalšími součástmi tzv. chytré domácnosti a podobně. Běžný uživatel ovšem vidí jen část aplikace zvanou front-end, o tom, co se skrývá za touto částí a jak aplikace vlastně funguje, obvykle tuší jen programátoři, testeři a další odborníci v oboru.

1.1.1 Front-endová část aplikace

Front-endová část aplikace (nebo zkráceně také pouze front-end) je ta část, se kterou uživatel pracuje, kterou vidí a se kterou interaguje. Její kvalita je velmi důležitá. Nejen, že se od ní odvíjí zákazníkovy pocity, tedy zda-li se bude zákazníkovi „na webu líbit“, v případě pracovní aplikace zda se mu bude dobře pracovat, či nikoliv, ale do značné míry významně ovlivňuje a spoluvytváří i zákazníkovo hodnocení celé aplikace. Zjednodušeně můžeme říci, že pokud nebude zákazník spokojen s front-end částí aplikace, bude negativně hodnotit celou aplikaci. Nezřídka zákazníci zavrhnou celou aplikaci kvůli tomu, že jim front-end část „nesedne na první dobrou“.

Front-end je tedy zodpovědný za zobrazení dat, odpovídá za příjem a prvotní zpracování požadavku, podává nám informace. Jeho úkolem je přijmout požadavek od uživatele a poté ho zaslat na server, který vykoná daný požadavek a vrátí front-endu výstup. (K. Liu et al., 2017)

Front-end může být vyvíjen přímo samotným programátorem, tedy za pomoci protokolů html, css a JavaScriptu, nebo pomocí rozličných nástrojů, které mohou celou práci vývoje usnadnit a urychlit a kterých je k dispozici velké množství. (Mozilla 1)

1.1.2 Back-endová část aplikace

Jako back-end bychom mohli označit část kódu, která se běžně nachází na serveru. Výhodou užívání kódu na straně serveru je dynamické zpracování dat dle potřeby a práv uživatele, přičemž tato data jsou čerpána z databází, ke kterým má aplikace přístup. Díky těmto databázím je možné na front-endové části zobrazovat zákazníkovi relativnější obsah. Další výhodou užívání kódu na straně serveru je možnost rozšíření přístupných funkcionalit pro uživatele, který si může uložit potřebné informace, jako například hesla, údaje o platební kartě a další. Pokročilejší stránky umožňují například zasílání oznámení na emailovou adresu. (Mozilla 2)

Komunikace mezi webovým serverem a webovým prohlížečem, tedy v podstatě mezi back-endovou a front-endovou částí aplikace, probíhá mimo jiné pomocí protokolu HTTP. V protokolu http (Hypertext Transfer Protocol) je například obsažena definice požadované akce (smazání, získání nebo odeslání prostředku). Dále je při komunikaci využívána URL, která slouží k identifikaci konkrétního zdroje, specifikuje například jméno, typ zdroje a adresář kde lze zdroj nalézt. Pokud klient (webový prohlížeč) zasílá HTTP GET požadavek na určitou URL, žádá o získání obsahu tohoto zdroje. (Databáze Národní knihovny ČR)

Server zasílá zpět odpověď, která může obsahovat mnoho sdělení, v tuto chvíli budeme předpokládat, že vše proběhlo v pořádku. Od serveru tedy prohlížeč dostane odpověď, že vše proběhlo v pořádku a v případě požadavku prohlížeč obdrží požadovaný výsledek. (mozilla.org 2, 2023)

1.2 Testování software

Máme tedy aplikaci. Zákazník, budoucí uživatel, ale i programátor, autor kódu, musí mít jistotu, že daná aplikace bude provádět přesně to, pro co byla vytvořena a její výstupy budou správné, přesné, úplné a opakovatelné. Aplikace také musí být dostatečně robustní, tedy odolná proti nestandardním zadáním, bezpečná atp. Pro ověření funkčnosti aplikace ve všech těchto směrech slouží proces, zvaný testování software.

Co je a není chyba

V anglickém technickém slangu se chybě říká bug, což se dá přeložit jako brouk. Etymologie tohoto termínu je jistě zajímavá. Ostatně i v češtině říkáme, že "to má mouchy", když něco nefunguje. V

programátorské praxi existuje několik druhů chyb a je potřeba si uvědomit, že ne za vše může programátor.

Programátor většinou nemá žádný vliv na hardwarové chyby. Existují ale i softwarové chyby, na které programátor nemá vliv. Příkladem může být například nedostatek volné paměti, kterou může operační systém přidělit našemu programu, nebo chyba v programu, se kterým náš program spolupracuje. Extrémním případem pak mohou být chyby v samotném operačním systému. Tyto chyby můžeme nazvat vnější chyby, tedy chyby, za něž programátor nenese přímou zodpovědnost a které nemá možnost ovlivnit. (D. Martínek, 2012)

S některými z těchto chyb ale může pomoci operační systém nebo ovladač tím, že umožní detekovat chybové kódy pro různé chybové stavy. Pokud to má smysl (velmi často to smysl opravdu má), měl by programátor tyto kódy testovat a ošetřovat. Čím větší škodu může chyba způsobit (program pro řízení jaderné elektrárny, projekt do školy), tím pečlivěji by si měl programátor počínat.

Programátor se může dopustit syntaktických a sémantických chyb. Syntaktické chyby jsou prohřešky proti gramatice používaného programovacího jazyka, takže je při překladu odhalí překladač. Menší syntaktické chyby odhalí i některé chytřejší editory. Je tudíž dobré soubor pro kontrolu přeložit vždy po napsání nějakého kousku kódu (cyklus, tělo funkce). Pokud je projekt dlouhý, trvalo by to dlouho, proto je výhodné dělit programy na moduly. Některé editory (např. Vim) umí program spouštět, aniž byste je museli opustit. Vývojová prostředí typicky umožňují spustit překlad příkazem z menu nebo klávesovou zkratkou.

Sémantické chyby jsou prohřešky proti sémantice, tedy proti požadovanému chování programu (sémantika = význam). Všechny tyto chyby principiálně nelze detekovat automaticky (stroji musíme nejprve nějak sdělit, co po něm chceme). Sémantické chyby způsobují chybnou funkci programu a právě na jejich odhalení se zaměřují všechny techniky hledání chyb.

Testování softwaru pak můžeme definovat jako proces, který pomáhá odhalit omezení a chyby v testovaném software. Testování je proces odhalující nedostatky klíčových vlastností software, tj. bezpečnosti, přesnosti, úplnosti a kvality.

Cílem testování je zajištění těchto klíčových vlastností. Zjednodušeně je možné říci, že cílem testování je nalézání nedostatků a chyb, pro které je třeba nalézt odpovídající řešení tak, aby koncový zákazník dostal spolehlivý a bezpečný software. (Z. Ali, 2019)

1.2.1 Testování

Testování lze definovat různě. Jiantao Pan ve své práci Software Testing cituje dvě definice, které ustavil Hertz: „Testování softwaru je proces spouštění programu nebo systému se záměrem najít chyby“ a „Testování softwaru je jakákoli činnost zaměřená na vyhodnocení atributu nebo schopnosti programu nebo systému a určení, zda splňuje jeho požadované výsledky“, uvádí i definici Myerse „Testování softwaru je proces spouštění programu nebo systému se záměrem najít chyby“. (J. Pan, 1999)

Testování je důležitou, povinnou součástí vývoje software, je to technika hodnocení kvality produktu a také jeho nepřímé zlepšování identifikací závad a problémů. (A. Abram et al., 2001)

Testování softwaru lze rozlišit na manuální a automatizované, případně na jejich kombinaci.

1.2.2 Manuální testování

Jak může již název napovědět, při manuálním testování musí někdo manuálně otestovat určitý software, či produkt. Nejčastěji tuto práci vykonává tester, který dle stanovených kritérií předem přesně daným postupem daný software otestuje. Cílem testování je ověřit funkčnost a odhalit případné chyby softwaru, které je potřeba opravit.

Při manuálním testování tester obdrží testovací scénář, ve kterém jsou uvedené kroky, které musí projít a ověřit tak funkčnost. Tento způsob testování je poměrně zdlouhavý, některé scénáře musí být opakovány několikrát. Pokud je více funkcí, které je potřeba testovat, tak je tento způsob i nákladný. Na druhou stranu manuální tester má na rozdíl od strojů kreativitu a může posoudit, jak je na tom vzhled softwaru, jak se mu v daném prostředí pracovalo a co by případně upravil.

1.2.3 Automatizované testování

Automatizované testování je provádění testů bez zásahu vývojáře, mohli bychom říci, že testovací zařízení automaticky provádí kroky, které by prováděl manuální tester. Samozřejmě, že vývojář musí test dle požadavků a zadání vytvořit. Jakmile je test hotový, lze ho spouštět neustále, popřípadě lze test nastavit tak, aby se spustil, při určité události. Vývojář nemusí test procházet ručně, což nám přináší velkou úsporu času a financí. (R. Steller et al., 2019)

Pokud bychom chtěli přímo definici automatizovaného testování mohli bychom použít: „automatizace testů je úkol vytvořit mechanicky interpretovatelnou reprezentaci manuálního testovacího případu“. (S. Thummalapenta el al., 2012)

Firma může automatizovat mnoho druhů testů, záleží na prioritách a potřebách firmy. Firmy se snaží automatizovat co nejvíce testů, jelikož se zvýší pokrytí a kvalita testování. Firmy se také tímto krokem snaží snížit náklady na lidské zdroje a vyvarovat se lidské chyby při vykonávání manuálního testování atd. (R. Steller et al., 2019)

Bylo řečeno, že automatizované testování firmě ušetří mimo jiné čas a finance, proto se firmy snaží automatizovat co nejvíce. Nicméně jsou případy kdy se automatizované testování nevyplácí, nebo není vůbec možné. Některé velmi složité testy je příliš náročné automatizovat, proto je lepší u některých scénářů zůstat u manuálního testování, jelikož se může stát, že náklady na vývoj složitého testu a jeho následnou údržbu převyšují náklady manuálního testování dané funkce. (M. Bureš et al., 2016)

Automatizované testování má i svá úskalí, jedním z nich je údržba testů. Jakmile vývojáři softwaru provedou nějaké změny, mohou se odrazit na funkčnosti testování. Testy se musí upravit, případně kompletně přepracovat, což firmu stojí nejen čas. (M. Bureš et al., 2016)

1.2.4 Rozdíl mezi automatizovaným testováním a manuálním testováním

Rozdílů mezi automatizovaným a manuálním testováním je mnoho, v této kapitole bude ukázáno jen několik.

První velký, možno říci zásadní rozdíl spočívá v možnosti opakování, automat vykonává test relativně rychle, automatizovaný test můžeme opakovat mnohem častěji. K tomuto bodu je nutné nadnést, že manuální testování provádí zaměstnanec, který má nárok na pauzy nebo dovolené, zákonem omezenou pracovní dobu (pro dlouhodobé, nebo dlouhodobě opakované testy je tedy třeba více kvalifikovaných sil) automat žádná takováto omezení nemá. Automat může test opakovat celý den, případně i nepřetržitě 24/7, přes víkendy a přes svátky. (M. Bureš et al., 2016)

Dalším rozdílem je přístup k provedení testu. Výsledky testů provedené automatem jsou při stejných vstupních podmínkách neustále stejné, automat pracuje přesně podle toho, jak je naprogramován. Automat nebude tedy zkoušet neobvyklou kombinaci, či postup, důsledkem čehož

může být neodhalení neschopnosti aplikace pokrýt některé neobvyklé nebo výjimečné stavy nebo situace. Naopak manuální tester může provést úkol trochu jinak pokud mu to intuice napovídá. Vzhledem k tomu, může manuální tester objevit chybu, kterou by automat neodhalil. Automat neumí být kreativní. (M. Bureš et al., 2016)

Automatizovaný test je naprogramován a následně funguje přesně dle zadání, problémem je, že pokud se testovaný systém byť jen nepatrně odchýlí od očekávaného chováni, test neumí situaci vyřešit a bude vyhodnocen jako NOK, tedy test neprošel. Takovéto chování chceme při velké nečekané změně, která se stát neměla. Nicméně občas se stane, že na stránce chybí jeden nedůležitý prvek a test přestane fungovat. Manuálního testera tento problém nezastaví, bude pokračovat dál a test dokončí. (M. Bureš et al., 2016)

Posledním rozdílem, který si zde uvedeme bude kvalifikace. Tester a vývojář automatizovaných testů mohou mít mnoho společného, oba mají znalosti ohledně pravidel a principů testování a mnoho dalšího, ale vývojář musí mít schopnost programovat skripty automatizovaných testů, umět pracovat v potřebných nástrojích atd. (M. Bureš et al., 2016)

1.2.5 Typy testování

Vzhledem k náročnosti vývoje front-endu a jeho následné údržbě existuje několik typů testů. Každý z nich testuje software jinak. Může jít o jednotkové testování, které zkoumá funkčnost jednotlivých prvků, přes testy, které testují vzájemnou komunikaci komponent, až po testy, které kontrolují celkovou funkčnost aplikace. Každý z uvedených typů má jedinečnou roli při zajišťování kvality a spolehlivosti.

1.2.5.1 Jednotkový test (Unit test)

Unit testing je testování jednotlivých jednotek nebo skupin jednotek příbuzných. (Ermira Daka; Gordon Fraser et al., 2014)

James Whittaker uvádí, že „testování jednotek testuje jednotlivé softwarové komponenty nebo kolekci komponent. Testeři definují vstupní doménu pro příslušné jednotky a ignorují zbytek systému“. (J.A Whittaker, 2000)

Jednotkový test je principiálně jednoduchý. Tim Koomen a Martin Pol definují unit test jako test prováděný vývojářem v laboratorním prostředí, který by měl prokázat, že program splňuje

požadavky specifikované v zadání. K testovanému programu, který má být testován, vytvoříme jiný program, který ho bude testovat. Tento dodatečný program bude testovat jednotlivé funkce testovaného programu na úrovni kódu. Program bude kontrolovat, jestli testovaný program vrací očekávané výsledky. (M. Bureš et al., 2016). Obecně by to mělo fungovat tak, že programátor napíše kód (metodu) a bezprostředně nato pro ni napíše test (dokonce existuje přístup psaní testů před kódem – tzv. Test Driven Development). Test by měl testovat chování kódu jak za standardních situací, tak v situacích mimořádných, např. co se stane, dostane-li metoda na vstupu null. (itnetwork)

Mimo jiné je možné pomocí tohoto programu otestovat chování testovaného programu v případě zadání nečekaného vstupu. Na jednotkové testování existuje několik frameworků, které mohou práci usnadnit a testování zefektivnit. (M. Bureš et al., 2016)

Tyto frameworky většinou vychází z xUnit architektury, která má za cíl tuto oblast standardizovat. Framework se mimo jiné vybírá podle programovacího jazyka, který tester zamýšlí použít. Pro Javu je to například JUnit nebo TestNG pro jazyky používané platformou .Net pak například nUnit nebo MbUnit. (M. Bureš et al., 2016)

Jednotkový test je napsaný ve stejném programovacím jazyce jako testovaný program. Jednotkový test se skládá z několika částí: set-up a tear-down metody, části arrange a assert a především testovací metody. Metoda set-up je volána před vlastním provedením testu, tato metoda slouží k nastavení prostředí, například načtení testovacích dat. Metody tear-down slouží k „úklidu" prostředí po provedeném testu. Testovací metody vykonávají test samotný. Frameworky umožňují opakované volání testovacích metod. Testovací metoda je složena z „arrange-act-assert", to je možné volně přeložit jako „připrav test-vykonej akci-vyhodnoť výstup". V arrange sekci je připravené vše, co test bude potřebovat, například definovaný očekávaný výsledek. Act část volá funkce, kterou testujeme. Poslední část assert nám kontroluje výstup s očekávaným výstupem definovaným v arrange části. Podle assert části framework vyhodnotí úspěšnost testu. (M. Bureš et al., 2016)

1.2.5.2 Assembly testy

Obvykle následují po unit testech. Jejich cílem je ověřit, že jednoduché části kódu, testované v unit testu, lze bezproblémově a plně funkčně sestavit do většího celku. Jde tedy o testy kompatibility a integrovatelnosti jednotlivých částí kódu. Obvykle tyto testy provádí zvláště k tomuto účelu určený vývojář

1.2.5.3 Integrační testy

V automobilech i mimo ně máme mnoho komponent, které spolu komunikují a spolupracují. Integrační testy prověřují funkčnost a bezchybnost této spolupráce, zda probíhá dle požadavků a zadaných parametrů. Jako příklad lze uvést komunikaci API s databází. (R. Steller et al., 2019)

Dnešní moderní aplikace se skládají z několika spolupracujících systémů. Tyto systémy spolu komunikují a posílají data pomocí integračních rozhraní. Často se vyskytují chyby právě v této komunikaci, proto je důležité ji správně testovat. K testování používáme integrační testy. V případě automatických testů jsou využívány dva typy integračních rozhraní. Prvním je lokální API, zde se jedná o různé části jednoho systému. Druhým rozhraním je vzdálené API, které propojuje systém se svým okolím, běžně pomocí rozhraní REST API, případně pomocí webové služby.

Oba typy rozhraní je vhodné testovat automatizovaně. V zásadě lze spustit test skrze uživatelské rozhraní testovaného systému a sledovat výsledek na integračním rozhraní, nebo je možné vyvolat tuto akci přímo na testovaném integračním rozhraní. Tento typ testu je možné použít v situaci, kdy není ještě celý proces dokončený, tedy není možné provést kompletní test. (M. Bureš et al., 2016)

1.2.5.4 Systémové testy

Tyto testy slouží pro ověření, že aplikace funguje tak, jak má. Na rozdíl od unit testů nejsou testovány části kódu, ale celá aplikace. Aplikace je testována, zda splňuje požadavky zadavatele, neobsahuje chyby, jsou zapracována řešení nestandardních situací a podobně. Obvykle jsou prováděny opakovaně – proběhnou testy – jsou nalezeny a opraveny chyby – opětovně probíhají testy.

1.2.5.5 Akceptační testy

Tyto testy slouží k ověření, že aplikace splňuje všechny požadavky zákazníka uvedené v zadání. To znamená nejen požadavky funkční, ale i požadavky například zátěžové, výkonové, chybovost a podobně. Tyto testy někdy provádí stejný tým, který aplikaci vyvíjel, většinou však zadavatel pověří provedením těchto testů tým odlišný. Pokud aplikace testy splní, obvykle si zadavatel aplikaci převezme.

1.2.5.6 Postakceptační testy

I přes veškerou snahu a testování se v aplikaci mohou vyskytnout chyby, které je případně potřeba opravit. Stejně tak je někdy nutné aplikaci rozšířit, nebo přidat další funkce. To se samozřejmě neobejde bez dalšího testování. (swtestovani)

1.2.5.7 Regresní testy

Regresní testy pomáhají kontrolovat chod a funkčnost software po přechodu na novější verzi. Software se neustále vyvíjí, přidávají se nové funkce, vzhledy a mnoho dalšího. Tyto změny přichází s novou verzí, která má částečně nebo zcela odlišný kód než verze předchozí. Regresní testy kontrolují, zda daná část softwaru funguje stejně jako dříve i po aktualizaci na novou verzi. Často se totiž stává, že část, která fungovala bezchybně, prošla testováním, které nenašlo žádné chyby, po nasazení nové verze nefunguje. Díky regresním testům je tedy možné přidávat do software nové funkcionality a zároveň udržet software bez chyb.

Jedna z možných definic regresního testování je na webu ISTQB Glossary: *„Typ testování související se změnami, jehož cílem je zjištění, zda nedošlo na nezměněných částech softwaru k výskytu nových nebo dříve neodhalených (skrytých) defektů.“* (ISTQB Glossary)

Lidé si často regresní testování pletou s konfirmačním testováním. Tyto testy se mohou zdát podobné, ale jejich účel je odlišný. Konfirmační testy se provádí, aby bylo možné potvrdit, že oprava určité chyby proběhla správně a systém nevykazuje ani chybu v předchozím kroku detekovanou a opravenou, ani chyby nové, vzniklé jako následek opravy. Konfirmační testování znovu spouští staré testovací scénáře, které nám vracely chybu. Ve chvíli, kdy je od vývojářského týmu potvrzeno, že je již chyba opravená, pouští se testy znovu, aby se potvrdila správnost opravy a funkčnost systému. (M. Bureš et al., 2016)

1.2.5.8 Smoke a sanity testy

Pojem „smoke test“ vznikl již dávno, dříve, než bylo běžné mít doma počítač, či mobilní telefon. Jednalo se o test, kdy s jistou dávkou nadsázky, tester přišel k novému hardwaru a stiskl tlačítko zapnout a čekal, zdali se ze stroje začne kouřit. Pokud stroj nezačal kouřit test proběhl v pořádku a se zařízením bylo možné pracovat. Smoke testing je krátké testování, zaměřené pouze na hlavní funkce vyvíjeného software. Neprověřuje jeho detaily. Provádí se při dokončení vývoje programu a po každé větší změně kódu programu Cíli smoke testingu jsou rychlé zjištění, zda vyvíjený software v zásadě funguje a je připravený pro další fázi testování a odhalení velkých chyb v co nejčasnějším

stadiu vývoje, což šetří čas při dalším vývoji a testování. Česky se někdy nesprávně nazývá „zahořovací testování“. (IT slovník)

Gupta a Saxena uvádí dvě definice sanity testování. Sanity test je prováděn obvykle po obdržení softwaru s malými změnami kódu nebo funkčnosti. Sanity test je prováděn jako potvrzení, že byly vyřešeny dříve nalezené chyby a toto řešení nezapříčinilo žádné další problémy. Cílem je zjistit, že navrhovaná funkcionalita pracuje přibližně dle předpokladů. Pokud sanity test selže, je software v dané podobě odmítnut, aby se ušetřily čas a náklady spojené s podrobnějším následným testováním. Druhá definice je poněkud odlišná, podle ní sanity test je zběžným testem, který potvrzuje, zda konkrétní software přináší požadované výsledky nebo ne. To znamená, že software již prošel jinými druhy testování, než se mohl podrobit sanity testu. Je důležité si uvědomit, že sanity test nejde tak do hloubky jako jiné typy testování. Sanity test podle této definice je možné zařadit do kategorie regresního testování, neboť postupy pro tyto testy jsou v podstatě shodné. (V. Gupta et al., 2013)

Smoke i sanity testy jsou velmi podobné regresním testům. Tyto testy ověřují, že části software, které fungovaly dříve, fungují i po drobné změně nebo opravě. Nicméně jejich cíl je mírně odlišný. Tyto testy nám nehledají konkrétní chyby. Ověřují nám funkčnost klíčových funkcionalit. Díky těmto testům se dozvíme, zdali je software stabilní a můžeme na něm začít pouštět detailnější testy, popřípadě zpřístupnit software zákazníkům.

Smoke testy kontrolují čistě funkčnost, v případě testování dotazníků neřeší volitelná políčka, neřeší funkčnost kontrol atd. Sanity testy jsou na druhou stranu tvořeny již více do detailu. Například při testování vyhledávání v autobazaru by nám smoke test mohl testovat, zdali funguje vyhledání při zadání textu „Škoda Superb“. Sanity test by udělal to samé, ale byl by například rozšířen o výběr motoru, či převodovky. (M. Bureš et al., 2016)

1.2.6 Cíle testování

Dříve než začne testování produktu, je důležité si určit, jak jsou cíle testování. Je dobré se na tuto problematiku podívat nezávisle s investorem a budoucími uživateli, jasně definovat priority koncových zákazníků. Investor v této fázi definuje své představy o funkci a výsledcích systému, tím jsou stanoveny obecné požadavky. Budoucí uživatelé, mohou představit velmi podrobné požadavky k jednotlivým částem a funkcím, včetně například jejich přístupnosti, ovládání a podobně.

V knize efektivní testování softwaru jsou cíle rozděleny následující tabulkou.

ID	Cíle testování
1	**V kritických funkcích systému nesmí být výpadky**
1.1	Funkce pořizování letenek musí spolehlivě fungovat
1.2	Systém musí být dostupný v režimu 24×7
1.3	…
2	**Datová migrace musí proběhnout v pořádku**
2.1	V systému musí být možné rychle dohledat historii transakcí ze systému, který se používal předtím
2.2	…
3	**Systém musí splňovat normy dané příslušnými úřady**
3.1	Systém musí splňovat normu X
3.2	Systém musí být možné napojit na centrální evidenci letových časů a linek
3.3	…

Obrázek 1: Cíle testování
Zdroj: (M. Bureš et al., 2016)

V tabulce můžeme vidět zvýrazněné řádky. Zde se jedná o požadavky od investora, které jsou obecnější. Pod tyto řádky jsou přidané řádky od budoucích uživatelů, kteří podrobněji popisují cíle.

Může se stát, že se budou požadavky od investora a od budoucích uživatelů lišit, nebo jsou dokonce v rozporu. Z toho důvodu je dobré, vzít si požadavky odděleně, nejdříve si poslechnout požadavky investora a poté požadavky budoucích uživatelů. Tímto krokem můžeme odhalit rozpory v očekávání. (M. Bureš et al., 2016)

1.2.7 Filozofie testování

Filozofie testování vychází z několika fundamentálních principů, které souvisí nejen s ověřováním správné funkcionality softwaru, ale také se snaží vytvářet udržitelné vývojové procesy.

Jednou z hlavních myšlenek je včasné odhalení chyb, před finálním testováním, jelikož včasná identifikace problému a jeho oprava v rané fázi vývoje minimalizuje náklady a zvyšuje efektivitu celého vývojového cyklu. Proto je důležité testovat software průběžně po celou dobu jeho vývoje.

Dalším důležitým bodem je zapojení všech zúčastněných stran (zákazníci, vývojáři, testeři, manažeři), aby bylo možné co nejlépe předávat informace, ať jde o změnu požadavků, či nalezení chyb.

Dále je důležité stanovit, kde je vhodné zavádět automatizované testování a kde nikoliv. Toto je důležité téma na začátku vývojového cyklu. Správné rozhodnutí může později ušetřit mnoho času a nákladů.

Pří vývoji nastanou chvíle, kdy dojde k chybám. Je však důležité se z těchto chyb poučit a zavést řešení, které bude v budoucnu sloužit jako prevence před opakováním stejných chyb.

Posledním bodem je flexibilita a agilita. S ohledem na rychle se rozvíjející moderní vývojové prostředí jsou tyto body nezbytné. Vývojový tým musí být schopen rychle a včas reagovat na změny požadavků, či na změnu okolí. (ChatGPT)

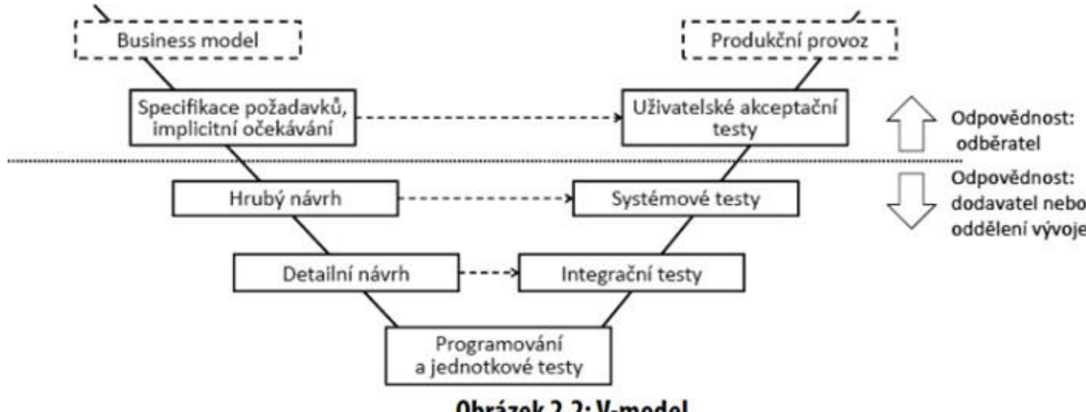
1.2.8 Metodiky testování

„V souvislosti s vývojem, provozem, projektovým řízením a testováním často mluvíme o metodikách. V obecném smyslu jde o postupy a vodítka, které nám mají být nápomocné k řešení určité situace." (M. Bureš et al., 2016)

O metodice tedy můžeme uvažovat jako o návodu, jak realizovat určité činnosti, během životního cyklu systému. Metodika může být interním dokumentem organizace, který je vytvořen tak aby splňoval potřeby firmy. Podle této firemní metodiky jsou podřízeny všechny činnosti spojené s testováním, cílem je zajistit shodný přístup k testování softwaru v rámci celé organizaci. Firmy často využívají V – model. V – model zobrazuje živní cyklus produktu, kde je kladen důraz na testování. (M. Bureš et al., 2016)

1.2.8.1 V model

Jedná se o model životního cyklu, tento model je testerům známý a často vštěpovaný. Byl definovaný v druhé polovině osmdesátých let 20. století. Jedná se o variaci vodopádového modelu. (M. Bureš et al., 2016)



Obrázek 2: V model
Zdroj: (M. Bureš et al., 2016)

Jak můžeme vidět na obrázku, model je tvořen několika dvojicemi. Na levé straně můžeme vidět aktivity, které souvisí se specifikací a návrhem. Na pravé straně se nachází aktivity spojené s testováním. V tomto modelu je zdůrazněna úloha testování ve vazbě na odpovídající analytické aktivity a související artefakty. V – model počítá s tím, že ne všechny požadavky bývají specifikované. V – model nám tedy říká, že i dobře navrhnutý systém je potřeba testovat, abychom mohli ověřit jeho kvalitu. (M. Bureš et al., 2016)

1.2.9 Testování v automotive

Mobilní služby nabírají celosvětově na oblibě. Není tomu jinak ani v automobilovém průmyslu, kde se stávají důležitým faktorem. Lidé chtějí pokrok, tato touha části lidí po autonomní jízdě, bezpečnosti a zábavě během jízdy podnítila rozvoj automobilů propojených s celosvětovou informační sítí a tím i potřebu online služeb. Tyto online služby komunikují a vyměňují si data s vozem.

Moderní agilní metodiky vyvolaly velkou změnu v automobilovém průmyslu. Nové funkce jsou rychle vyvíjené a je možné je okamžitě nasazovat. Na druhé straně máme dlouhý vývoj vozidla a jeho hardwaru. Hardware musí projít bezpečnostním měřením a správnými postupy. Pro bezpečnost vozu je v současnosti software stejně důležitý jako hardware. A stejně jako hardware je nutné software testovat a ladit. (R. Steller et al., 2019)

A právě na testování software v automotive je zaměřena předkládaná práce.

Tato práce je zaměřena na vozy vyráběné po roce 2018. Vozy, které jsou vyrobené před rokem 2018 a ještě neobsahují jednotku OCU (Onboard Communication Unit – řídicí jednotka vestavěného systému nouzového volání). Tyto vozy komunikují tedy pomocí adaptéru, který je zaveden ve slotu palubní diagnostiky (OBD). Adaptér zavedený v diagnostickém slotu posílá data do spárovaného mobilního zařízení pomocí Bluetooth rozhraní. Novější vozy, jsou již vybavené OCU jednotkou, tedy není potřeba zavádět žádné adaptéry. OCU jednotka snímá signály na sběrnici CAN (Controller Area Network) ve vozidle. CAN sběrnice je hlavní komunikační síť pro signály ve vozidle. Veškerá data z vozidla se odesílají do rozhraní API vozidla. V API rozhraní se data transformují. Jednotka je typizovaná pro několik typů aut, přičemž každý typ auta zasílá jiná data, proto je nutná transformace dat. API rozhraní tedy převádí všechna data na standardní objektový model. Transformovaná data jsou zaslána na Connected Fleet Backend (CFB). V CFB jsou uchovány všechny back-endové služby. Mezi tyto služby patří servis vozidel, řidičů, knih jízd a mnoho dalších. Jelikož máme mnoho různých zařízení a formátů, jako jsou stolní počítače, chytré telefony a tablety, jsou data dále transformována pomocí FAPI (Frontend API). Díky tomu jsou všechna data dodána standardizovaným způsobem všem zákaznickým aplikacím. V dnešní době používáme aplikace na platformách Android, IOS a webové aplikace.

Vzhledem k výše zhruba popsanému procesu sdílení informací mezi vozem a mobilním zařízením se nám jeví hned několik problémů souvisejících s testováním této komunikace.

Testování veškerých interakcí mezi vozem a zákazníkem je v reálném provozu prakticky neproveditelné a jak technicky, tak finančně náročné. Tak abychom při testování mohli simulovat chování každého potenciálního koncového zákazníka by obnášelo několik tisíc vozů a miliony kilometrů testovacích jízd. Každá z těchto testovacích jízd by musela být jiná. Styl jízdy, rychlost rozhodování, pozornost, řešení kritických situací u potenciálních řidičů, vliv hustoty a druhu provozu, povrchu vozovek, pneumatik a mnoho dalších proměnných, toto vše by se muselo neustále simulovat a testovat. (R. Steller et al., 2019)

1.3 Software pro vývojáře

IDE neboli Vývojové prostředí (Integrated Development Environment)

IDE je software usnadňující práci programátorů, většinou, ale ne nutně, zaměřené na jeden konkrétní programovací jazyk. Obsahuje obvykle editor zdrojového kódu, kompilátor, případně

interprettter a většinou také debugger. Vývojových prostředí je celá řada, namátkou například Eclipse, IntelliJ, ActiveState Komodo, NetBeans, Microsoft Visual Studio. Některé z těchto softwarů nejsou plnohodnotnými IDE, jsou podstatě pouze editory kódu. Blíže ve své práci uvedu dvě vývojová prostředí, se kterými jsem měl možnost se seznámit a mezi nimiž jsem volil ve své bakalářské práci. A to Visual Studio Code a Pycharm. Pro úplnost je nutné uvést, že ne všichni vývojáři spoléhají při své práci na IDE. Mnozí programátoři píší své programy pomocí samostatných programů, například GCC, Make nebo GDB). Mezi uživateli Linuxu je oblíbený textový editor Vim, který po doplnění pluginy schopen zastávat některé funkce IDE.

1.3.1 Visual studio code

Visual Studio Code je editor kódu s podporou vývojových operací, jako je ladění, spouštění úloh a správa verzí Jeho cílem je poskytovat nástroje, které vývojář potřebuje pro sestavení kódu a ladění, a složitější pracovní postupy přenechává plnohodnotnějším IDE, jako je například Visual Studio IDE. (Visualstudio)

Jedná o vývojové prostředí, které spravuje firma Microsoft.

Pro lepší představu viz obrázek 3 níže

Obrázek 3:Visual Studio Code
Zdroj: Vlastní

V levé části obrázku je zobrazena lišta s ikonami. Velkou část času vývojář stráví v prostředí prvního odkazu, což je prohlížeč. Tato část zobrazuje složky a soubory, které jsou obsažené v projektu. Dalším důležitým polem je pole 3, které je značeno třemi tečkami pojenými čárami. Pokud verzujeme projekt v Gitu, tak zde můžeme nahrávat změny na GitHub, či GitLab. Je tak možné projekt pohodlně průběžně nahrávat. Pokud bychom byli puristy a volili čistě „programátorský“ přístup, je možnost nahrát kód přes terminál, který se nachází úplně dole. Posledním polem, které si zde zmíníme je pole s rozšířeními. Jedná se o páté pole, které je značeno čtyřmi čtverci. Zde je možné do kódu přidat rozšíření, Těchto rozšíření je celá řada a skýtají programátorovi velký prostor ke kreativitě, případně k přesnému doladění software. Za zmínku stojí rozšíření využívající implementaci posledního výkřiku techniky – umělé inteligence (AI), kde je již aplikovaná AI, která vývojáři napovídá základní podobu kódu a výrazně tak může urychlit a usnadnit práci.

V druhé části obrazovky, která zabírá většinu plochy, vývojář vyvíjí kód. Je možné se setkat s různými podobami kódu, což je naprosto v pořádku. Záleží na cíli vývojářovy práce, někdo používá VS Code na psaní testů, někdo na vývoj webu, popřípadě na vývoj aplikací. V každém tomto případě bude pravděpodobně použit jiný programovací jazyk a kód bude velmi odlišný.

1.3.1.1 Výhody

Jednou z hlavních výhod VS Code je jeho dostupnost, jelikož VS Code je open-source software, pracující na všech známých platformách. Vývojář pak nemusí řešit, jaký má k dispozici operační systém. Další výhodou obzvláště pro začínající vývojáře je možnost získat program bezúplatně. Začínající vývojář pravděpodobně nebude mít zdroje na drahé vývojové prostředí.

Další výhodou je univerzálnost použití, VS Code podporuje velké množství programovacích jazyků. Vývojář pak může snáze vyjít vstříc zadavateli zakázky, pokud ten má striktní požadavky na použití programovacího jazyka. Programovací jazyk je možné zvolit již při instalaci software VS Code.

1.3.1.2 Nevýhody

V případě složitějších úkolů může být pro některé požadavky nedostatečný a vývojář je pak nucen použít doplňkový software. V případě debbugovacích funkcí nemusí dosahovat takové výkonosti jako jiná prostředí, která jsou zaměřena na jeden programovací jazyk. (History computer)

1.3.2 Pycharm

Pycharm je kompletní IDE, který nám umožňuje profesionální vývoj v jazyce Python. I název nám může napovídat, že toto prostředí se zaměřuje převážně na jazyk Python. Pycharm je jeden z mnoha produktů od firmy JetBrains. Tato firma má několik vývojových prostředí, pro různé typy jazyků. Prostředí je možné využít jak pro vývoj webů, tak i pro datovou analýzu. V prostředí by měl být zabudován AI asistent, který bude nápomocen při vývoji. (Jetbrains)

1.3.2.1 Výhody

Hlavní výhodou tohoto prostředí, která však pro někoho může být brána i jako nevýhoda, je zaměření na jeden určitý jazyk. Díky tomu je práce snadnější, nemusí se stahovat žádné dodatečné programy. Je tedy k dispozici kvalitní podpora a optimalizace výhradně pro jeden jazyk, z toho vyplývají lepší debuggovací nástroje atd. (History computer)

1.3.2.2 Nevýhody

Pro začínajícího vývojáře, nebo pro začínající firmu, může být toto prostředí drahé. Další nevýhodou je relativně malý přesah prostředí, sice jsou podporované jazyky jako je html, css a JavaScript aby bylo možné vyvíjet webové stránky, ale tím zde přesah končí. Jelikož se jedná o komplexní a obsáhlý software, vyžaduje výkonnější výpočetní techniku než v případě VS Code.

1.3.3 Volba vývojového prostředí

Firma, či jedinec se musí při vybírání prostředí zamyslet a rozhodnout, které prostředí je v dané situaci vhodnější. Ve firmě, se kterou jsem na vývoji spolupracoval a pro kterou byl testovací scénář vyvíjen je k dispozici jak Pycharm tak VS Code. Každé prostředí má své výhody i nevýhody. Pro svojí práci jsem zvolil VS Code. Kromě výše uvedených výhod hrála významnou roli skutečnost, že všichni kolegové používají tento software a já jsem mohl využít jejich zkušeností, rad a konzultací.

1.3.4 Gitlab

Pokud bychom chtěli přesnější definici, můžeme citovat it Muni, která popisuje Gitlab takto: „GitLab je webový nástroj pro kompletní vývojový cyklus software. Poskytuje Gitový repozitář, wiki, sledování chyb, kontinuální integraci (CI/CD), vytváření skupin, tvorbu statických webových stránek či management projektu. Výčet funkcí lze nalézt na stránkách projektu." (IT MUNI)

Jinak by bylo možné popsat Gitlab jako webovou platformu pro správu s kódem. Tato platorma slouží jako prostředí pro správu verzí, správu projektů, spolupráci týmů a automatizaci softwarového vývoje. Gitlab je komplexní nástroj, který kombinuje funkce správy verzí Gitu s různými nástroji pro plánování projektů.

GiLab je postaven na Gitu. Git je distribuovaný systém, ve kterém je možné spravovat verze kódu. Git umožňuje sledovat vývojářům změny v kódu, spravovat různé větvě kódu a usnadňuje spolupráci na projektech. Jednodušeji řečeno, Git slouží jako úložiště kódů, ke kterému mají přístup všichni vývojáři v daném týmu. (ChatGPT)

GitLab existuje od roku 2011, kdy byl vydán Dmistrijem Zaporozhetsm. Původní myšlenkou GitLabu, bylo vytvoření open-source alternativy k již existujícím službám pro správu kódu, jako byl GitHub. Od té doby se stal GitLab velice populární a oblíbenou platformou. GitLab v dnešních dnech je nabízen k užití jak zdarma, tak za poplatek. Zpoplatněná verze přináší rozšířené funkce a podporu. (ChatGPT)

1.3.5 Jenkins

Jedná se o open-source automatizační server napsaný v programovacím jazyku Java. Podporuje velkou řadu systémů například Git, CVS, Sebversion, AccuRev a mnoho dalších. Díky nástroji Jenkins může vývojář automatizovat testy.

Tento nástroj je ve firmě, pro kterou je aplikace v rámci bakalářské práce vyvíjena, používán jako jediný. Nemohu tak ve své práci porovnat případné alternativy, neměl jsem s čím srovnávat. (Jenkins)

1.3.6 BrowserStack

Pro testování front-end aplikace, je potřeba zařízení, na kterých budou testy prováděny. Zajistit si takové vybavení může být finančně velmi náročné, další finanční výdaje bude stát údržba těchto strojů. Pokud firma nemá prostředky na pořízení, nebo např. jde o jednorázové testování, pro které by byl nákup zařízení nerentabilní a zbytečný, nebo pouze nechce mít vybavení u sebe, existuje řada firem, které propůjčují zařízení určené k testování.

V případě BrowserStacku se jedná o cloudovou testovací platformu, jak webovou, tak i mobilní. Tato platforma umožňuje vývojářům testovat jejich software na vyžádání. Mobilní zařízení jsou fyzická, není však třeba, aby byl tester fyzicky přítomen, výhodou je možnost využití vzdáleného přístupu.

Pro aplikace i pro testování je na výběr z několika druhů telefonů, a dvou operačních systémů, IOS a Android. Pokud je vzdáleně provolán test, ve kterém je stanoveno, pro jaké je určen zařízení a verzi operačního systému, je ve velmi krátké době nalezen volný telefon a test je v reálném čase spuštěn. Výhodou tedy je, že je k dispozici log kroků, které test vykonal a případně lze snadno zjistit, na kterém kroku selhal. K tomu se na telefonu tvoří záznam, lze tedy celou sledovat dobu kroky, které test vykonává. (Browserstack)

Instalace v tomto případě není potřebná, jelikož se jedná o cloudovou službu, je možné mít otevřenou ve webovém rozhraní.

Firma si pravděpodobně před testováním samotným udělá analýzu a zhodnotí, zdali je pro ni výhodnější pořídit vlastní hardware o který se bude muset starat a někde ho skladovat, nebo využít externích služeb BrowserStacku, případně jeho konkurentů.

2 Praktická část

2.1 Oddělení firmy

Oddělení, kde byla vykonávána praxe je orientováno na zákaznický servis. Stará se o řešení problémů v komunikaci mezi automobily a mobilní aplikací. Dále udržuje informační tok dat, nebo propojuje řešitelské skupiny atd.

Oddělení je složeno z několika týmů, které spolu na denní bázi spolupracují. Každý z těchto týmů se stará o jinou problematiku. Nachází se zde například tým analytiků, kteří mají na starosti analyzování problémů a nalezení jejich příčiny. Dalším týmem je kontrolní centrum, které spravuje zákaznické tickety a předává je k analýze. Autor této práce vykonával svoji stáž v týmu monitoringu, který monitoruje funkčnost mobilní aplikace za pomoci automatizovaných testů. Na oddělení je dalších několik týmů, nicméně pro předkládanou práci jsou důležité právě tyto tři týmy, jelikož spolu úzce spolupracují.

Tým monitoring vyvíjí automatizované testy, které jsou spouštěny na pravidelné bázi. Tyto testy testují front-end i back-end mobilní aplikace. Testují různé funkce napříč aplikací. V případě, že některé testy přestanou opakovaně procházet a jsou tedy neúspěšné, spustí se automatické upozornění, které je zasíláno kolegům z kontrolního centra. Zde se provede první základní analýza problému. Díky analýze je možné určit, jakému týmu problém dál předat. Existuje mnoho řešitelských skupin, kterým se předávají různé problémy. Jedna z možností je předání problému analytikům na oddělení, jelikož je potřeba provést hlubší analýzu. Analytici na oddělení provedou hloubkovou analýzu, díky které je již možné určit, na jaký tým problém přesměrovat.

Mojí úlohou v rámci oddělení je analytická i vývojová činnost. Z počátku jsem tvořil analytické úkoly, jako vytvoření testovacích scénářů, či očišťování výsledků testu. Bylo tomu tak z důvodu rychlejšího pochopení fungování testů samotných. Po nasbírání zkušeností jsem začal plnit roli vývojáře automatizovaných testů. Díky počátečním činnostem mám přesah k analytikovi a jsem schopen tvořit si testovací scénáře sám, či posoudit výsledky proběhlých testů. Tedy mezi mou náplň práce lze zařadit tvorbu testovacích scénářů, jejich vývoj a posléze vyhodnocení výsledků.

2.2 Testovaná aplikace

Aplikace, kterou se zabývá předkládaná práce, je mobilní aplikací firmy z České republiky, která se zaměřuje na výrobu aut. Pomocí této aplikace můžeme ovládat funkce vozu, případně na dálku zapínat některé funkce auta.

Tuto aplikaci si může uživatel stáhnout na svůj mobilní telefon. Je dostupná jak pro telefony s operačním systémem android, tak i pro telefony, které mají operační systém IOS. Pokud by někdo nevlastnil mobilní telefon, nebo jej nechtěl používat, jsou k dispozici i webové stránky. Nicméně v předkládané práci bude testována mobilní aplikace. Po stažení aplikace si uživatel musí vytvořit uživatelský profil a poté „enrollovat" svůj vůz, jednodušeji řečeno propojit své auto s daným účtem. Toto propojení se liší v závislosti na modelu auta. V posledních letech bylo na trh uvedeno mnoho nových modelů. Každý model se propojuje odlišně. Některé vozy lze propojit pomocí QR kódu, jiné se musí spárovat pomocí kódu a startovacích klíčů v automobilu atp. Některé starší vozy nelze do této aplikace přidat vůbec, jelikož jejich softwarová řešení nejsou s touto aplikací kompatibilní. Jelikož je mnoho typů automobilů, bude tato práce soustředěna na jeden konkrétní vůz, který je plně elektrický a vyrábí se od roku 2020. V mobilní aplikaci je možné zapínat funkce vozu, jako jsou například: zapnutí topení/chlazení, naplánování trasy, nabíjení baterie a mnoho dalších. Mimo zapínání funkcí vozidla a přehledu stavu lze v aplikaci nalézt například manuál k vozu nebo možnost vyhledání nejbližšího servisního partnera.

Praktická část bude doprovázena několika záznamy obrazovky, které autor pořídil při své práci. Jelikož některá data mohou být předmětem průmyslového utajení, či přímo pod legislativním omezením (GDPR), budou některé části těchto obrazů cenzurovány.

2.2.1 Testovaná funkce

Tato práce bude zaměřena na test, který se jmenuje Plans. Automat bude mít za úkol nastavit pravidelný plán, který se spustí v určitém čase v určitý den. Spuštěním tohoto plánu dojde ke změně nastavení vyhřívání vozidla dle výchozího nastavení. Tento test v následujících krocích změní nastavení plánu opět do výchozího stavu a plán vypne.

Test započne na úvodní obrazovce mobilní aplikace. Zde automat vybere možnost přihlášení a provede přihlášení do aplikace účtem, který je běžně užívaný v testech, nemusí tedy vytvářet nový

účet. Po přihlášení do aplikace se zobrazí sekce vozu. V této sekci automat zkontroluje prvky, které se na stránce musí nacházet, aby bylo možné úspěšně pokračovat. Jedná se například o zobrazení stavu nabití baterie a dojezdové vzdálenosti vozu. Pokud některé z uvedených prvků nebudou uvedeny na stránce, test bude ukončen a vyhodnocen jako neúspěšný. Kontroluje se například nabití baterie. Po kontrole bude automat hledat políčko s názvem „Temperature“ (teplota). Ve chvíli, kdy toto políčko najde, tak na něj klikne. Tímto krokem se test dostane do nastavení klimatizace, či vyhřívání vozidla. Automat sjede na dolní část obrazovky, kde vyhledá políčko s názvem „Plan 1“. Pokud takové pole najde, klikne na něj. Díky tomu se automat dostane do sekce nastavení plánu.

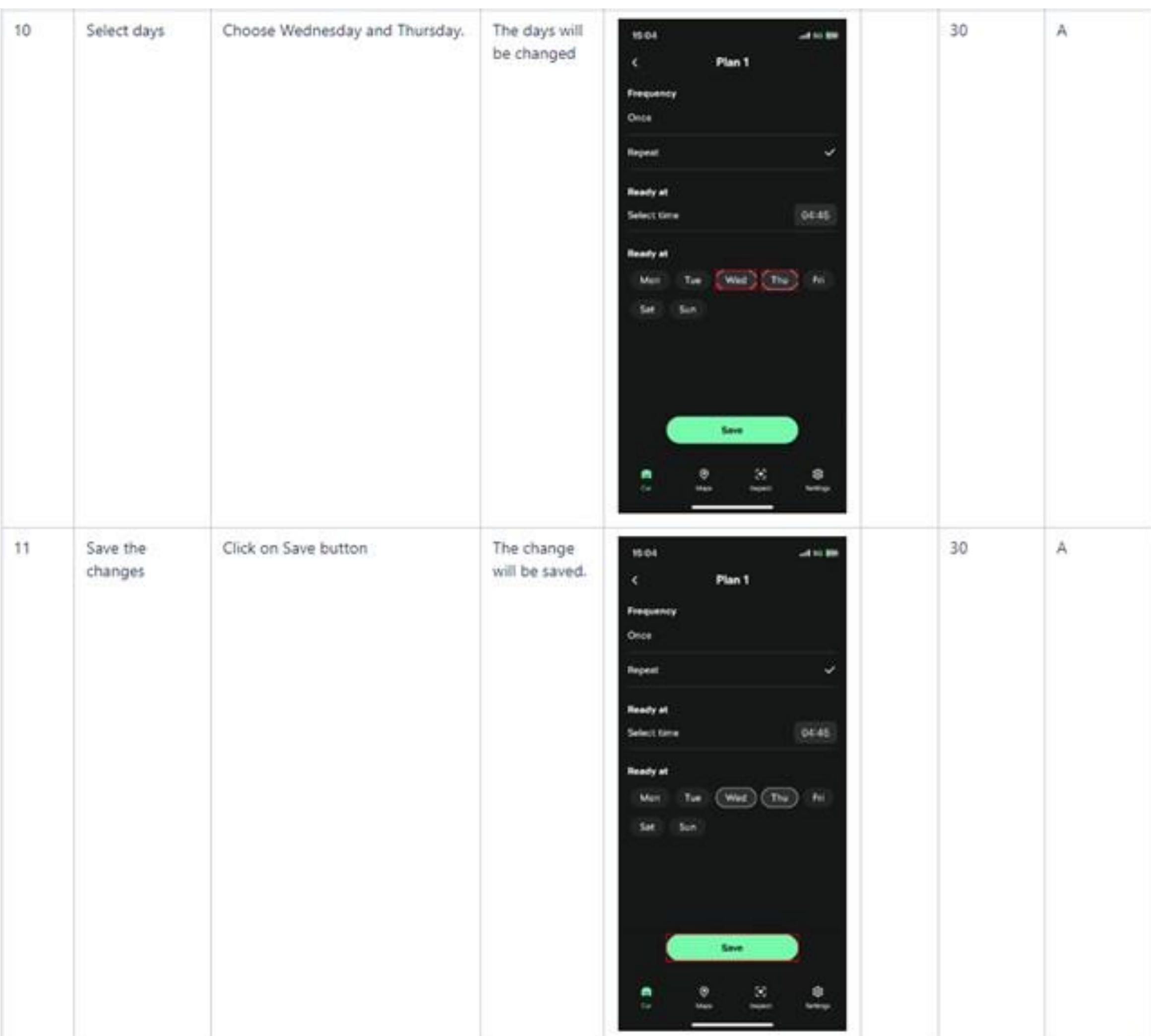
Výchozí nastavení testu by mělo být zobrazené následovně: „Frequency:Once, Time:6:20, Day: Mon“. Test ověří, jestli se shoduje zobrazené nastavení s výchozím. Pokud se bude nastavení shodovat automat jej začne měnit. Jedná se o přenastavení opakování, které automat přenastaví tak, aby se plán opakoval každou středu, čtvrtek a pátek. Po provedení změn, bude plán uložen tlačítkem „Save“. Nyní bude automat čekat na potvrzující vyskakovací okno. Ve chvíli, kdy bude zachyceno potvrzení, že byl plán úspěšně uložen opět klikne na políčko s názvem „Plan 1“. Nyní změní plán opět do výchozího nastavení a klikne na políčko s názvem „Save“. Test bude znovu čekat na potvrzení uložení. Po načtení tohoto upozornění test klikne na tlačítko, které zapíná/vypíná plán a tím jej vypne. Toto vypnutí opět kontroluje pomocí vyskakovacího potvrzení o úspěšném uložení. Tímto krokem je test ukončen a pokud dojde úspěšně až sem, je test vyhodnocen jako OK, tedy jako úspěšný.

2.3 Vytvoření analýzy

Tento krok běžně vývojář nedělá. Tato práce náleží analytikům. Nicméně jak bylo již zmíněno v práci, autor měl na praxi přesah a tvořil si testovací scénáře sám.

Tvorba analýzy začíná pokynem od nadřízeného, či jiné oprávněné osoby. Od této osoby obdrží analytik zadání, ve kterém je obsaženo několik pokynů, které musí analytik splnit. Jedná se například o to, jaká funkce má být testována, na jakém modelu automobilu má být testována, na jaké verzi aplikace má být testována a mnoho další. V našem případě se jedná tedy o testovací scénář s názvem „Plans“. Zadání pro analýzu je v podstatě popsáno v předchozí kapitole.

Po prostudování zadání a pokynů pro test bylo započato s tvorbou analýzy testovacího scénáře. Byly definovány jednotlivé kroky, které budou v budoucnosti prováděny automaticky. Jedná se v podstatě o manuální testování aplikace, kdy je každý testovací krok zaznamenán a uložen jako screenshot. Ve screenshotu byly zdůrazněny významné body a k nim byl případně přiložen detailní popis, případně vysvětlivky, aby bylo zřejmé, co má daný příkaz či krok vykonat. (viz. Obrázek 5)



10	Select days	Choose Wednesday and Thursday.	The days will be changed			30	A
11	Save the changes	Click on Save button	The change will be saved.			30	A

Obrázek 4:Analýza testovacího scénáře
Zdroj:vlastní

Pro ukládání a sdílení screenshotů se na pracovišti využívá soukromá wiki, kde je uloženo nepřeberné množství informací o celém oddělení a všech jeho týmech. V tomto rozhraní lze i nalézt složku se všemi front-endovými testy. Zde byla založena nová složka s názvem „Plans“, kam byly posléze uloženy snímky všech jednotlivých kroků.

Na obrázku jsou vidět položky, které bylo nutno vyplnit. První položka z levé strany je číslo, které uvádí o kolikátý krok testu se jedná, na uvedeném obrázku je krok 10 a 11. Dále je název kroku,

který bude později používán v testu při vývoji. V další kolonce je popsán daný krok, aby bylo zřejmé, jakou operaci má krok provést, tyto informace jsou vkládány v anglickém jazyce. V kroku číslo 10 následuje volba dnů spuštění plánu, zde středa a čtvrtek. Dále je nutné uvést, co bude následovat po daném kroku, v tomto případě změna výběru dnů. Následuje nahrání již pořízeného a upraveného screenshotu. Dále je za screenshotem uvedena číslovka. Toto číslo uvádí maximální délku trvání daného kroku, obvykle v sekundách, v tomto konkrétním případě „30“, tedy tento krok se musí vykonat do 30 sekund. V případě, že není krok vykonán do zadané doby, končí test neúspěšně. V posledním oddílu je třeba určit maximální časové trvání celého testu. Na výběr je ze tří možností, každá doba trvání je reprezentována jedním písmenem, A, B nebo C. V daném případě doba A trvá 30 s, doba B 150 s a doba C 300 s. Doba trvání testu je určována podle jeho náročnosti. Pokud má určitý krok testu za úkol kliknout na tlačítko, jehož funkce je otevření nastavení aplikace bude časový limit mnohem nižší než u kroku, který má za úkol spustit na dálku klimatizaci v automobilu, jelikož takový krok, musí cestovat přes back-endové servery do auta a zpět. V daném případě se jedná o nenáročný test, a proto byla zvolena varianta A.

V okamžiku vyplnění a popsání všech kroků testu nastává čas konzultace se Splunk specialisty.

2.3.1 Konzultace se Splunk specialisty

Tým je složen ze specialistů a vývojářů, kteří pracují s nástrojem Splunk, který slouží ke správě dat. Tento tým je koncový, přebírá a třídí data. Analytik vytvoří testovací scénář, vývojář vyvine test dle zadání a na konci musí být někdo, kdo umí výsledky třídit, porovnávat, zobrazovat a prezentovat. Splunk tým tvoří grafy a statistiky, které jsou prezentovány jiným oddělením, případně bussines manažerům. Na základě dat, tedy na výsledcích automatizovaných testů je vytvořen upozorňovací mechanismus. Pokud dojde k výpadku na některé z komponent, testy se začnou vyhodnocovat jako neúspěšné. Ve chvíli, kdy ve velkém množství začnou přicházet testy s výsledkem NOK, automaticky se zašle se upozornění kontrolnímu centru. Kontrolní centrum přepošle problém příslušné řešitelské skupině, která je zodpovědná za jednotlivé logické a infrastrukturní komponenty. Díky tomuto mechanismu je možné zjistit a opravit nefunkčnost aplikace dříve, než si tohoto problému všimne koncový zákazník.

Každý testovací scénář je unikátní a testuje určitou funkci. Z toho jasně plyne, že testy musí být v datovém prostředí rozeznatelné. Proto je při tvoření nového testu nutné se Splunk specialisty nastavit několik parametrů. Při konzultaci se specialistou odpovědným za danou oblast byl testu

přiřazen unikátní atribut, díky kterému je test identifikován a označen po celou dobu jeho vývoje a následného používání. Ve chvílí, kdy bude test vyvinut, projde laděním a bude uvolněn pro použití a nasazen na produkci, budou všechna data, která budou zasílána označená tímto atributem. Dále bylo testu přiřazeno colection name, které určuje, pro jakou generaci automobilů bude test vyvíjen. V neposlední řadě bylo testu přiřazeno bussines name, které definuje, v jaké skupině testů bude test zařazen. Díky tomu je později možné test statisticky hodnotit, určit počty úspěšných a neúspěšných běhů testu a podobně.

2.4 Vývoj

Po obdržení testovacího scénáře započíná vývoj. Vývojář si testovací scénář prostuduje, pokud je to zapotřebí kontaktuje analytika a řeší s ním nejasnosti.

Při vývoji testu bylo využito prostředí Visual Studio Code. Toto prostředí je možné propojit s Gitem, s výhodou tak bylo možné využít kompletní knihovny testů a souborů potřebných pro vývoj testu. K této knihovně mají přístup všichni vývojáři testů, což významně usnadňuje spolupráci a sdílení poznatků a zkušeností jednotlivých vývojářů.

Vlastní práce na kódu započala zjišťováním potřebných elementů, tedy tlačítek, šipek, nadpisů atp. a jejich jednoznačné identifikace, tedy zjišťováním jednotlivých ID těmto prvkům přirazeným. K tomuto účelu byl využit program Appium Inspector vlastní proces bude popsán dále.

Zjištěné elementy a jejich ID, či XPATH byly zapsány a uloženy do souborů s názvem elements. Tyto soubory jsou uložené v podsložkách kořenové složky jejichž umístění vypadá následovně: „Kořenová_složka/platform/android/elements.py“. Soubor elements pro IOS je uložený podobně, ale ve složce iOS. Tyto soubory jsou v knihovně dva, jeden pro operační systém Android a druhý pro iOS, každý z nich je uložen v samostatné podsložce pro jednotlivé OS. Je tomu tak z důvodu, že jednotlivé ID a XPATH elementů se liší podle operačního systému. Nicméně test byl vyvíjen jak pro operační systém Android, tak i pro iOS, proto je nutné, aby byl daný element uložen pod konstantu, která bude mít stejný název v obou souborech.

Identifikace několika elementů nebylo nutno zjišťovat, jelikož byly již uloženy, protože jsou využívány i v jiných testech.

Ve chvíli, kdy byly identifikovány a uloženy všechny potřebné elementy, byla vytvořena „kostra kódu“, jednoduše řečeno, do kódu byly zapsány kritické kroky testu, viz obrázek 6.

```
with Step ('Go to target temperature', 'A'):

with Step ('Swipe to element', 'A'):


with Step ('Go to Plan 1 make change', 'A'):



with Step ('Set frequency change', 'A'):
```

Obrázek 5: Kostra kódu
Zdroj: vlastní

Byly přepsány jednotlivé kroky, jejich názvy a časové omezení.

2.5 Zjištění ID elementů aplikace

Aby bylo možné interagovat s elementy aplikace je nutné znát jejich ID nebo XPATH. V následujících kapitolách bude uvedeno, jak nastavit program a jak s ním pracovat a zjistit potřebná data.

2.5.1 Appium Inspector

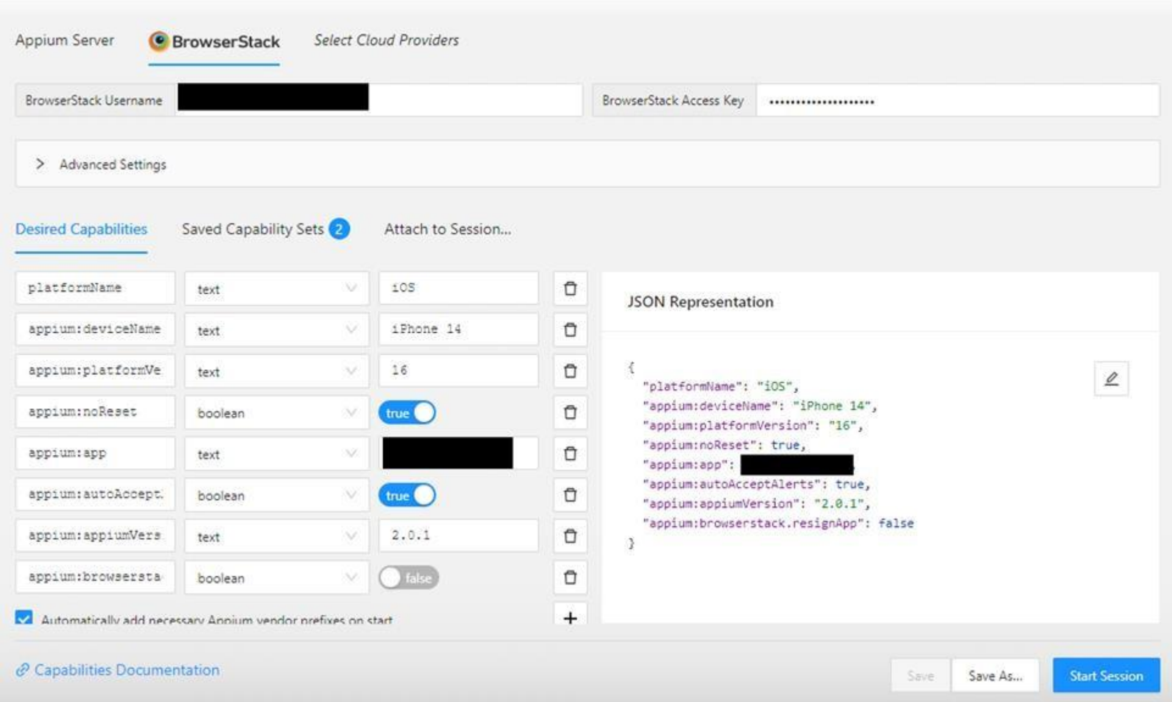
Pomocí Appium inspector je možné se vzdáleně připojit k telefonu na kterém je mobilní aplikace dle požadavků nainstalována. Po připojení je možné tuto mobilní aplikaci procházet a zjišťovat její vlastnosti.

Appium Inspector má i své nevýhody. Samotné stažení instalačního souboru může být pro běžného uživatele nezvyklé. Není zde žádná vlastní webová stránka, kde je možné za pomoci tlačítka stáhnout program. Appium Inspector nalezneme ve složce Gitu, která je dohledatelná na internetu. Appium inspector je ke stažení pro Windows, MacOS, či pro Linux. Po stažení souboru uživatel provede běžnou instalaci a program je nainstalován.

2.5.2 Nastavení Appium Inspector

Po nainstalování a spuštění aplikace si uživatel může vybrat způsob, kterým se připojí k mobilní aplikaci. Existuje mnoho způsobů připojení, nicméně na oddělení je standardně používán BrowserStack. Mobilní aplikace je v tomto prostředí nahraná a vývojáři pravidelně přidávají nové verze.

Výchozí nastavení programu se ukázalo být ne zcela vyhovujícím, proto bylo provedeno nové nastavení programu.



Obrázek 6: Nastavení Appium Inspector
Zdroj: vlastní

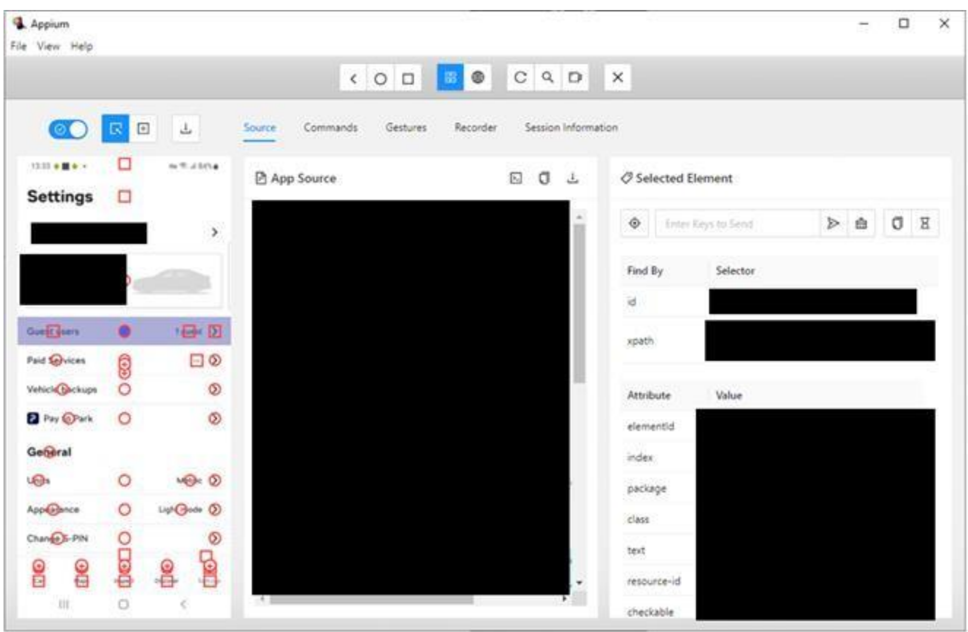
Jak je možné vidět na obrázku číslo 7, bylo změněno nastavení několika parametrů. Nejdříve byla provedena změna nastavení, kterým se program připojuje k nástroji BrowserStack, ve kterém je nadefinováno spojení k mobilním telefonům, ke kterým se Appium Inspector připojuje. Po vyplnění přihlašovacích údajů následovalo nastavení souboru JSON. V tomto souboru byl nastaven operační systém telefonu (nastavení položky platformName), a dále konkrétní model telefonu.

Nastavení bylo uloženo s názvem iOS. Totožný postup byl aplikován pro nastavení pro operační systém Android. Jediné odlišnosti byli v prvních třech řádcích a řádku s položkou „Appium:app“, kde z důvodu utajení není možné uvést podrobnosti.

Po provedení výše uvedených nastavení bylo zvoleno již uložené nastavení pro IOS a poté byla spuštěna relace klikem na tlačítko „Start Session“.

2.5.3 Používání programu Appium Inspector

Po spuštění relace je možné volně procházet mobilní aplikací. Zjišťovat podrobné informace o tlačítkách a popisech. V této práci byla převážně používána obě krajní pole okna programu.



Obrázek 7: Appium Inspector
Zdroj: vlastní

Pomocí tlačítek v pravé části horní lišty je možné vybrat způsob práce s elementy, tj. zda je cílem zjišťovat jejich podrobné informace, či na ně „kliknout“, tedy simulovat krok běžného uživatele, který by prstem „klepnul“ na tlačítko. Program byl využit zejména pro zjišťování informací o jednotlivých elementech, proto byl nastaven pro práci v tomto režimu. Jak je na obrázku vidět, některá tlačítka, či nadpisy mají kolem sebe červená ohraničení v podobě kolečka, či čtverce. Pokud došlo ke kliknutí na toto tlačítko došlo k zobrazení všech dostupných informací elementu na pravé stránce obrazovky. Pro tuto práci byly vyhledávány zejména ID či XPATH, tyto informace jsou zobrazeny vpravo nahoře.

2.6 Tvorba kódu

Po identifikaci a uložení všech elementů bylo možno přejít na samotný vývoj testu. Prvním krokem bylo zkopírování kroku login. Tento krok je používán ve všech testech, není tedy nutné jej v testu vymýšlet, ale je běžně pouze kopírován. Dále započalo vyplňování na počátku vytvořené kostry.

```
firstLogin()

with Step ('Go to target temperature', 'A'):
    temperature = getElement(elements.target_temperature)
    temperature.click()

with Step ('Swipe to element', 'A'):
    swipeToElement(elements.plan2)

with Step ('Go to Plan 1 make change', 'A'):
    sleep(1)
    planChange = getElement(elements.plan1)
    planChange.click()

with Step ('Set frequency change', 'A'):
    frequencyChange = getElement(elements.set_frequency_repeat)
    frequencyChange.click()
```

Obrázek 8:Vyplnění kostry kódu
Zdroj: vlastní

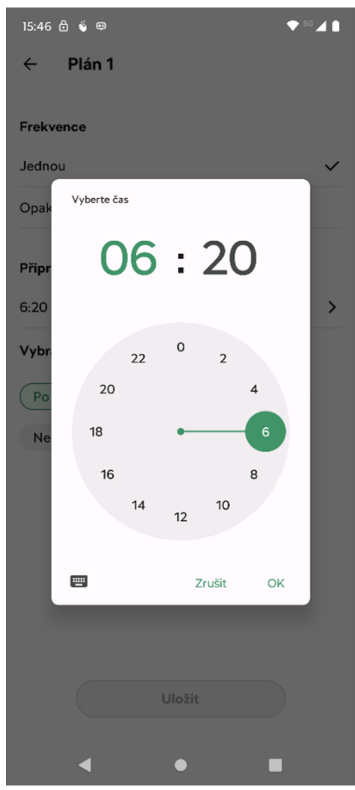
Jak je možné vidět na obrázku, po okopírování kroku login, následuje krok s názvem „Go to target temperature“. Do konstanty „temperature“ bylo nahráno ID elementu, který je uložený v souboru elements pod názvem „target_temperature“. K tomu se používá metoda „getElement“. Na dalším řádku je uvedena konstanta následovaná funkcí „click()“. Tato metoda nasimuluje krok uživatele a „klikne“ na element. Tento krok tedy klikne na políčko, které má ID nadefinované v souboru helpers.

V dalším kroku byla použita metoda „swipeToElement“. Díky této metodě bude test „swipovat“ (přejíždět po obrazovce) k elementu „plan2“. Tento krok, tedy pomocí metody přejede po stránce k elementu „plan2“

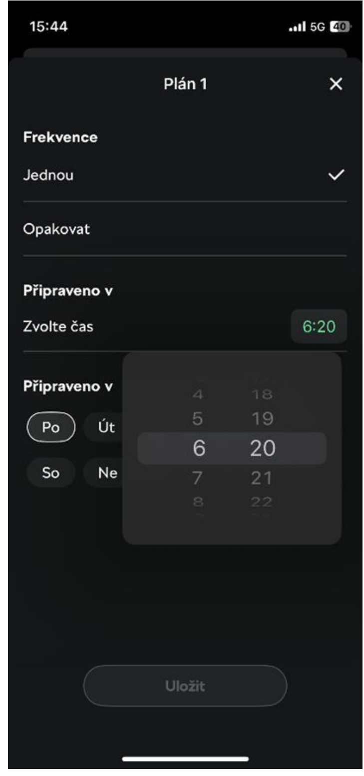
Další krok začal metodou „sleep“. Metoda byla použita, aby se předcházelo nechtěným chybám. Při zkušebním spouštění testu bylo opakovaně zjištěno, že sjíždění po obrazovce občas způsobilo dočasné zablokování běhu („zaseknutí“) aplikace. Z toho důvodu byla použita tato metoda, která na 1 sekundu zastaví (uspí) test. Díky tomu se aplikace mohla načíst a test mohl pokračovat dále.

V dalších krocích bylo postupováno totožně jako v krocích předchozích, pouze byly měněny elementy a názvy konstant.

S následujícím krokem nastaly obtíže, jelikož grafické rozhraní u operačního systému iOS a Android je velice odlišné viz. obrázky 10 a 11.



Obrázek 9: Nastavení času na operačním systému Android
Zdroj: vlastní



Obrázek 10:Nastavení času na operačním systému IOS
Zdroj: vlastní

V tomto kroku má být nastaven přesný čas spuštění plánu. Jak je možné vidět na obrázcích toto nastavení je pro každý operační systém zcela odlišné.

Tento problém byl v průběhu vývoje testu úspěšně vyřešen vytvořením podmínky typu „if/else“. Do kódu byla zanesena if/else podmínka, přičemž se v bloku if nacházela část pro operační systém iOS a v bloku else se nacházel kód pro Android.

Po spuštění testu bylo zjištěno, že podmínka bezchybně funguje pro operační systém Android, nicméně blok if, kde byl kód pro iOS nebyl funkční. Autor se řadí mezi juniorní vývojáře, i proto nebyl schopný tento problém vyřešit sám. Po rozhovoru se zkušenějším seniorním kolegou, bylo zjištěno, že zvolené řešení není špatné, ale je schované za neviditelnou vrstvou. Jak je možné vidět na obrázku 11, tak se na stránce nenachází žádné potvrzovací tlačítko, kterým by bylo možné uzavřít rozhraní ve kterém je čas nastavován. Běžný uživatel jednoduše klikne někam do prázdného prostoru a tím okno zavře. Proto v testu bylo nastaveno, aby program „klikl“ na tlačítko „jednou“, jelikož bylo stejně již vyplněné a nic by tím nebylo změněno. Jak bylo již uvedeno výše toto řešení nebylo funkční a test nikam neklikl.

Bylo tomu tak z úvodu již zmíněné neviditelné vrstvy, která překrývala toto tlačítko a test jej nemohl najít. Tato neviditelná vrstva se zobrazila ve chvíli, kdy bylo rozkliknuto nastavování času. Za pomoci seniorního vývojáře došlo k vytvoření mechanismu, který vypočítá prázdné souřadnice na stránce a do nich klikne. Tím zavře okno, ve kterém se nastavuje čas a test může pokračovat. Problém byl úspěšně vyřešen a test opakovaně prošel OK, čímž bylo prokázáno, že řešení problému s rozdílným nastavením pro různé operační systémy pomocí podmínky „if/else“ je správné a funkční.

Při vytváření tohoto kroku byl test zkušebně spouštěn lokálně. Při tomto ladění se projevil další problém, který způsoboval občasné selhání testu.

Bylo zjištěno, že pokud selže test na hlášce timeout, tak to nutně neznamená, že se nastavení plánu neuložilo. Pokud byl tedy spuštěn testovací běh, který nastavil opakování na dny: středa, čtvrtek a pátek v 4:20 a ukládání plánu překročilo maximálních 30 s, tak byl test vyhodnocen jako neúspěšný. Nicméně po pár minutách se plán uložil. Pokud se po tomto testovacím běhu spustil další běh, test nebyl schopen změnit nastavení, jelikož již bylo změněno, a proto byl opět vyhodnocen jako neúspěšný.

Jak bylo uvedeno v úvodních kapitolách úspěšný testovací běh opět přenastaví plán do výchozího nastavení. Při analýze tedy nebylo myšleno na možnost selhání testu v půlce, tedy na problém, který byl nyní odhalen.

Důvod proč další testovací běh byl neúspěšný je tedy zřejmý, plán byl již změněn, test neměl co změnit, jelikož všechny údaje byly již nastavené, v takovém případě neměl automat možnost plán uložil, jelikož mu to aplikace nedovolila.

Započalo vymýšlení podmínky, která by tuto problematiku ošetřila. Bylo zde zapotřebí vymyslet, podle čeho bude moci automat poznat, zdali je nastavené výchozí nastavení plánu, či již upravené. Pro tento účel byl vybrán čas, jelikož se měnil a s ním i dny. Pokud byl plán ve výchozím nastavení byl nastavený následovně: Frekvence: Jednou Čas: 6:20 Den: Po, pokud byl upravený vypadal takto: Frekvence: Opakovat, Čas: 4:20, Den: Středa, čtvrtek, pátek.

Za tímto účelem vznikla podmínka na obrázku 12.

```
        frequencyChange.click()

    with Step ('Select time change', 'A'):
        timeValue = getElement(elements.time_picker)
        timeValue = getAttribute(timeValue, platform)
        timeValue = (timeValue.split(":"))[0].strip("0")
        timeValueNew = "4" if timeValue == "6" else "6"
        print(f"{timeValue} : {timeValueNew}")

        if platform == "ios":
            timeWheelChange = getElement(elements.time_picker)
            timeWheelChange.click()

            changeTimeText = getElement(elements.change_time)
            sendText(changeTimeText, timeValueNew, platform)
            outOfWheel = findInvisibilityElement(elements.frequency)
            bounds = getDictBounds(outOfWheel)
            bod_x = bounds['x'] + int(bounds['width']*0.75)
            bod_y = bounds['y'] + int(bounds['height']*0.3)
            clickTap(bod_x, bod_y)

        else:
            timeWheelChangeAndroid = getElement(elements.time_picker)
            timeWheelChangeAndroid.click()
            if timeValueNew == "4":
                setFour = getElement(elements.change_time_four)
                setFour.click()
            else:
                setSix = getElement(elements.change_time_six)
                setSix.click()
            clickOK = getElement(elements.click_on_ok)
            clickOK.click()
```

Obrázek 11: Podmínka if/else
Zdroj: vlastní

Bylo tedy zapotřebí zjistit, jaký je nastavený čas plánu. Proto byl do konstanty „timeValue“ nahrán element obsahující čas. Z tohoto elementu byla pomocí metody „getAttribute“ zjištěna hodnota, tedy obsah elementu. Hodnota tedy byla buď „6:20“ nebo „4:20“. Pro funkčnost podmínky, nám stačí znát pouze číslovku 4 či 6. Proto byla hodnota rozdělena pomocí metody „split(:)“, která rozdělila hodnotu v místě kde se nacházela dvojtečka. Vzniklo tedy pole znaků, ze kterého bylo zapotřebí vzít pouze první číslo. Proto byl vybrán nultý prvek pole, tedy první znak. Tento znak byl ještě očištěn od nul pomocí metody „strip(„0“)“. Tento poslední krok autor doplnil až později, jelikož při několikátém spuštění testu zaznamenal, že občas se ukazuje čas jako: „06:20“ a jindy jako „6:20“. Pro běžného uživatele není rozdíl mezi 06 a 6, jelikož obojí znamená šest hodin. Nicméně při porovnávání čísel nastal problém, protože pro software hodnota 06 není totožná s hodnotou 6. Z toho důvodu byla tedy doplněna metoda strip. Těmito kroky byla zjištěna hodina, ve kterou by se plán měl spustit.

Po těchto krocích byla vytvořena podmínka. Tato podmínka je velmi jednoduchá, říká: Pokud se konstanta timeValue rovná 6, tak bude mít nová konstanta timeValueNew hodnotu 4. Pokud se šesti nerovná, bude hodnota 6.

Principiálně podobným způsobem bylo přistoupeno k programování dalších úkolů testu, jako kontrola vyskakovacích oken, kontrola vypnutí a zapnutí plánu. V průběhu psaní kódu byly průběžně řešeny drobné problémy. Podrobnosti nejsou popsány, neboť podrobným popisem by neúměrně vzrostl rozsah práce a ta by se mohla stát málo přehlednou.

2.7 Další užité soubory

Při vývoji testu bylo využito několik souborů, které zatím nebyly zmíněny. Je tomu tak z důvodu, že do daných souborů nebylo nijak zasahováno. Pro kompletnost práce jsou však důležité. Mimo soubor, kde byl vyvíjený test, který má běžně pojmenování ve tvaru JménoTestu.py a souborů elements, byly pravidelně užívány dva další soubory. Jedním z nich je soubor s názvem „helpers“.

Jak je z práce zřejmé, byly v průběhu psaní kódu používány některé metody., které samozřejmě musí být někde uložené. Všechny zmíněné metody jsou uložené v souboru helpers. Tyto metody byly vyvinuty seniorními vývojáři. Metody v tomto souboru jsou globální, a proto je možné je volat do jednotlivých testů. Příkladem může být metoda „GetAttribute“, kterou bylo možné vidět v testu. V testu má tato metoda pouze jeden řádek, ale tím se pouze volá ze souboru helpers.

```
375 def getAttribute(element, platform):
376     """
377     Metoda vraci atribut elementu
378     """
379     if platform == "android":
380         attribute = "text"
381     else:
382         attribute = "value"
383     return element.get_attribute(attribute)
384
```

Obrázek 12: Soubor helpers
Zdroj: vlastní

Jak je možné vidět na obrázku číslo 13, tato metoda vrací atribut elementu. Je v ní obsažena podmínka, která se spouští podle operačního systému. Pokud bude operační systém Android, bude do konstanty attribute nahrán text, pokud se bude jednat o iOS bude do konstanty nahrána hodnota (value).

Jedná se o stejný text, který je pouze jinak uložený pro daný operační systém.

Druhým užívaným souborem je soubor browsterstack.yml. V tomto souboru je nadefinované spojení s Browserstackem. Mimo jiné jsou v něm nadefinované přihlašovací údaje, typ mobilního zařízení, verze aplikace atd. Díky tomuto souboru bylo možné test spouštět lokálně ze zařízení. Bohužel, vzhledem k průmyslovému utajení není možné poskytnou více informací.

V poslední řadě se zaměříme na soubor elements. Jak do něj autor zanášel potřebné elementy a jich ID, či XPATH. Pro lepší vysvětlení je na obrázku 14 uveden screenshot obrazovky.

```
114    time_picker = (AppiumBy.ACCESSIBILITY_ID,                )
115    change_time = (AppiumBy.XPATH,                                     )
116    ready_at = (AppiumBy.XPATH,                                    )
```

Obrázek 13: Soubor elements
Zdroj: vlastní

Jak je možné vidět na obrázku, byly dohledány elementy jak s ID tak s XPATH. Jak bylo již avizováno je preferováno ID. Nicméně se nepodařilo u všech elementů dohledat ID, jelikož neexistovalo. Z obrázku plyne, že zadávání těchto informací se příliš neliší. Jediný rozdíl je změna v závorce za „AppiumBy“. Pokud bylo dohledáno id pokračoval kód následovně: AppiumBy.ACCESSIBLITY_ID, ‘Nalezené ID elementu‘. U zadávání XPATH se postupovalo totožně s jediným rozdílem, tedy: ApppiumBy.XPATH, ‘XPATH elementu‘.

2.8 Lokální spuštění testu

Jak již bylo zmíněno, při vývoji byl test několikrát lokálně spuštěn. Díky tomu bylo možné ověřit funkčnost již vytvořených kroků. Během vývoje byly zjištěny hned dva problémy.

```
(env) PS            Documents\Git\mobile-app-testing> browserstack-sdk python .\tests\carSection\plans.py
```

Obrázek 14: Lokální spuštění testu
Zdroj: vlastní

Z obrázku se může zdát, že stačí jeden prostý příkaz ke spuštění. Což je ve skutečnosti pravda za předpokladu, že jsou nastavené správně všechny parametry v souboru Browsterstack.yml.

Během práce se podařilo všechny potřebné parametry zadat korektně, a proto mohl být test spuštěn, jedním příkazem. Tento příkaz se skládá z příkazu „browserstack-sdk“, poté následuje určení programovacího jazyka, v našem případě „Python“ a v poslední řadě se uvádí cesta k souboru, který chceme spustit.

2.9 Nahrání na GitLab

Ve chvíli, kdy byl vývoj testu ukončen a bylo několikrát ověřeno, že test funguje bezchybně při lokálním spuštění, tak bylo na čase nahrát test na Git.

Pro tento krok byl využit terminál vývojového prostředí. První příkaz, který byl použil je „git add \cesta k souboru“ příkazem byly označeny soubory které jsou připraveny k odeslání. Takto

postupně byl přidán test samotný a poté byly přidány soubory elements. Poté následovala kontrola, zdali se všechny soubory opravdu propsaly a jsou připraveny k odeslání. K této kontrole slouží příkaz „git status“. Poté následovalo zadání dalšího příkazu a tím je git commit -b „libovolná zpráva“. Tímto příkazem došlo k označení souborů zprávou s textem „Test Plans v.1“, díky které byla později snazší orientace v Gitu. Posledním použitým příkazem, kterým došlo k odeslání na Git je „git push origin Název Větve“.

Větve jsou části knihovny. Nejčastěji využívanou větví je větev „master“ se kterou pracují všichni vývojáři. Jako začínající vývojář jsem během vypracování bakalářské práce byl a stále jsem juniorem. Z toho důvodu mi byla vytvořena nová větev, která v podstatě kopírovala master větev. Bylo tak učiněno z důvodu bezpečnosti, jelikož junior snadno udělá při práci chybu, kterou může ovlivnit fungování větvě, či může dojít ke ztrátě dat. Pokud by došlo ke smazání dat z větve master, byl by to velký problém pro vývojový tým. Pokud bych jakkoliv poškodil svou vytvořenou větev, jednoduše by došlo k jejímu smazání a byla by mi vytvořena větev nová. Tedy v tomto posledním případě by mohl příkaz vypadat následovně: „git push origin Plans“. Došlo by tedy k odeslání vybraných dat na větev s názvem Plans.

Soubor helpers nebyl odeslán na Git, jelikož nebyl během vývoje testu jakkoliv změněn.

2.10 Automatizace testu

Automatizace probíhá za pomoci serveru Jenkins. Zde se provádí několik nastavení. Prvním je nastavení spouštění v čase. Je možné nastavit test tak aby byl spuštěn každou druhou hodinu, či každý druhý den. Nastavení vypadá následovně: „** ** ** **“. První dvě hvězdy slouží k nastavení minut, další dvě slouží k nastavení hodin, další dvě k nastavení dne a poslední dvě k nastavení měsíce. Pro lepší pochopení bude tato problematika vysvětlena na příkladu: „** 0-23/2 ** **“ Zde jsou vyplněné pouze hodiny, je zde uvedený časový úsek celého dne lomeno dvěma. Tedy test poběží každou druhou hodinu dne, neboli poběží dvanáctkrát za den. Pokud by došlo k úpravě 0-23/4, tak by test běžel po 4 hodinách tedy například v 00:00,04:00,08:00 atd. Hodinové nastavení je využívané nejčastěji, jelikož spouštění testu jednou za den, či jednou za měsíc nemá pro tým vypovídající hodnotu.

Dalším nastavením může být řetězové spouštění. Některé testy mohou mít určitou návaznost. Je tedy možné nastavit jeden test tak aby běžel každé tři hodiny a další test nastavit, aby se spustil pokaždé, kdy se spustil test číslo 1.

Toto se provádí u náročných testů, které například zapínají topení v autě. Tyto testy jsou občas vyhodnoceny jako neúspěšné s hláškou „timeout“ tedy nastavování trvalo déle, než bylo maximálně povoleno. Jak bylo již avizováno výše, to že test skončil neúspěšně s hláškou „timeout“ nemusí nutně znamenat, že se nastavení neuloží za pár minut po jeho skončení. Tedy například po testu, který zapíná a poté vypíná topení v automobilu, je pouštěn další test, který kontroluje, zdali je topení opravdu vypnuté. Je to z důvodu ochrany baterie, která se při dlouhém vyhřívání vozu velice rychle vybíjí.

Jelikož toto nastavení je pro všechny testy stejné a není nutné jej měnit a zároveň tato problematika není určena pro juniorní vývojáře vzhledem k důležitosti, nebude tato problematika popsána podrobněji. Vyhotovený test byl pouze předán seniornímu vývojáři, který jej přidal do správné složky, ve které později docházelo k automatizaci.

2.11 Spuštění na testovacím serveru

Dříve než byl test přidán do balíčku testů na produkčním prostředí, tak musel projít testováním. Test byl spuštěn na testovacím serveru, ze kterého nejsou zasílány výsledky nikam dál, jsou pouze k zobrazení. Zde byl test spouštěn po dobu jednoho dne v intervalu dvou hodin. Tedy se zopakoval 12krát.

Po uplynutí 24 hodin bylo zapotřebí prohlédnout všechny jednotlivé běhy a zkontrolovat, zda-li všechno proběhlo v pořádku a že nebyla objevena žádná nová chyba. Jelikož test fungoval v pořádku byl test schválen a nasazen na produkční prostředí. V případě, že by byla objevena nová chyba, byl by test stažen, dokud by chyba nebyla opravena.

2.12 Spuštění testu

Spouštění testu se odvíjí od potřeb zákazníka. Tedy pro jaké oddělení byl test vyvíjen. Tento konkrétní test měl sloužit pro průběžné testování aplikace a pro regresní testy.

Test byl tedy spuštěn v (ne)pravidelných intervalech, aby bylo možné zkontrolovat dostupnost a funkčnost mobilní aplikace. V jiném případě byl test spouštěn v případě vydání nové verze aplikace, aby bylo zjištěno, zdali je testovaná funkce dostupná a funkční i na nové verzi.

V případě regresních testů jsou testy spouštěny minimálně na dobu 24 hodin. Během této doby se odhalují problémy jak aplikace, tak testů. Pokud dojde k odhalení chyby aplikace, předává se tento problém dál k vývojářům mobilní aplikace, kteří by měli poskytnout opravu. Na druhou stranu může docházet ke změnám v aplikaci, které můžou daný test udělat nefunkčním. Může dojít ke změně textu, smazání tlačítka atp. Každá taková změna způsobí 100% neúspěšnost testu, jelikož test bude hledat něco co již v aplikaci není. V takovém případě musí dojít k upravení testu, tak aby splňoval nové požadavky.

Průběžné testování nepřináší práci pro vývojáře testů, jelikož se aplikace nemění. Toto testování slouží ke statistice funkčnosti mobilní aplikace, která je prezentována jiným oddělením. Dále toto testování pomáhá odhalit případné výpadky na back-endových komponentech. Jak již bylo zmíněno tým monitoring má díky těmto výsledkům vytvořený systém upozornění v případě velkého poklesu dostupnosti.

2.13 Očištění výsledků

Výsledky testů musí být pravidelně kontrolovány a analyzovány. V průběhu testů může docházet k neočekávaným chybám. Zde je důležité pochopit rozdíl mezi skutečnou chybou a falešnou chybou. Skutečnou chybu je nutné najít, jelikož se jedná o chybu aplikace, či výpadek na některém z komponent. Takové chyby jsou hlášeny a zaznamenávány, jelikož jejich oprava je důležitá.

Naopak falešně chyby jsou nechtěné. Zde se jedná o chyby, které jsou zapříčiněny neočekávaným chováním aplikace. Může se jednat například o zobrazení varovného okna, které se běžně nezobrazuje. Pro příklad lze uvést, že uprostřed testu na telefonu vyskočí upozornění „Baterie je téměř vybitá". Běžný uživatel takové upozornění odklikne a případně dá nabíjet svůj mobilní telefon. Automat však s takovým scénářem nepočítá, ani počítat nemůže. Proto kdyby došlo k této události, test bude vyhodnocen jako neúspěšný s chybovou hláškou „timeout". Důvodem bylo, jak bylo v práci ukázáno, že test má určené jednotlivé kroky, které jsou často typu klikni na tohle, sjeď k tomuhle atp. Ve chvíli, kdy by vyskočilo takovéto okno, test by mohl jen marně hledat tlačítko

Plan 1. Jednodušeji řečeno, jedná se o chyby, které mohou zapříčinit neúspěšnost testu, ačkoliv aplikace a testovaná funkce jsou funkční.

2.14 Vyhodnocení výsledků

Test byl již několikrát spuštěn a navrátil několik výsledků. Pokud se podíváme na poslední testovací cyklus, tak testovací scénář byl spuštěn po dobu jednoho dne každé dvě hodiny, tedy celkově 12krát. Na zařízení s operačním systémem Android proběhl 7 úspěšně a 3 neúspěšně. Na IOS test proběhl 6 úspěšně a 4 neúspěšně. Dva testovací běhy na každém operačním systému chybí. Je tomu tak z důvodů nečekaných problémů telefonu, kdy tyto testy byly vyhodnoceny jako falešné, a tedy se do celkové statistiky nezapočítávají. Testy na obou operačních systémech se zdají být přibližně rovnocenné. Z daného množství dat není zcela možné vyvozovat statisticky podložené závěry, na to je dat velmi málo a časová dotace bakalářské práce neumožnila další, časově náročné, pokusy. Je možné, že by další testy potvrdily vyšší stabilitu systému Android, nicméně vyvozovat tento závěr z daného množství dat by nebylo statisticky zcela korektní.

Neúspěšné testy byly jako takové vyhodnoceny vesměs z důvodu překročení času běhu testu – systému trvala reakce déle, než bylo předpokládáno a než je pro praktické použití únosné. To značí obtíže v komunikaci mezi mobilním zařízením, back-endovými servery a automobilem, na nichž budou muset vývojáři firmy ještě zapracovat.

Firma ušetřila i nemalé množství finančních prostředků tím, že se rozhodla tento testovací scénář automatizovat. Pokud by manuální tester vykonával tento test 12krát, strávil by vykonáváním této činnosti přibližně 5 hodin. Mimo testování funkce samotné musí tester vkládat data do vhodného prostředí. Automatizace tohoto testu by trvala mediornímu vývojáři přibližně 2-3 pracovní dny. Nicméně je nutné vzít v potaz, že po vyvinutí testu není nutný další zásah vývojáře, mimo nutnou údržbu, která není příliš častá. Dle webových stránek indeed je průměrný měsíční plat juniorního softwarového testera 36 971 Kč. Pokud přepočítáme plat na hodinový a poté vynásobíme 100 hodinami dostaneme výsledek 23 107 Kč. Pokud by bylo zapotřebí testovat funkci každý pracovní den 12krát celý měsíc, tak by firma za takové testování zaplatila 23 106 korun českých. Průměrný plat vývojáře automatizovaných testů stanoví webová stránka jooble na 49 999 Kč. Nicméně, jak bylo již zmíněno vývoj takového testu by trval 3 pracovní dny, tedy 22,5 hodiny. Pokud přepočítáme průměrný měsíční plat na hodiny a poté vynásobíme 22,5 hodinami dostaneme částku

7 031 Kč. Měsíční údržbu a spouštění testu můžeme stanovit na další 3 pracovní dny, tedy dalších 7 031 Kč, což je v součtu 14 062 Kč. (Jooble)(Indeed)

Firma tedy na měsíčním testování takové funkce ušetřila 9045 Kč. Je nutné vzít v potaz, že tyto dny byly stanoveny odhadem. Není možné přesně říci, kolik dní v měsíci bude potřeba upravovat test, a proto je potřeba brát tyto částky pouze orientačně.

Závěr

Na základě zadání byla úspěšně vytvořena analýza a byl sestaven příslušný testovací scénář.

Na základě této analýzy bylo započato s tvorbou testovacího software. Byly úspěšně dohledány, identifikovány a případně přiřazeny jednotlivé ID a XPATH požadovaným elementům. Byly identifikovány kritické body testu.

Při psaní kódu byly úspěšně vyřešeny potíže s blokováním běhu aplikace použitím metody sleep.

Následně byl postupně sepsán kód testovacího programu a postupně průběžně řešeny vyskytující se potíže.

Úspěšně byl vyřešen problém principiálně odlišné definice nastavení časů spuštění u operačních systémů Android a iOS. Problém byl vyřešen použitím podmínky typu if/else, přičemž v každém bloku byl vepsán kód pro jiný operační systém.

Byl úspěšně vyřešen i problém s neviditelnou vrstvou blokující možnost uzavření rozhraní ve kterém byl nastavován čas. Problém byl vyřešen vytvořením mechanismu, který vypočítá prázdné souřadnice na stránce a následně do nich klikne, čímž se dané rozhraní uzavře.

Další vyřešený problém se týká selhání testu na hlášce timeout, které bylo způsobováno pozdním uložením testu. Tento problém byl vyřešen využitím časové podmínky, kdy musel být nejprve rozdělen údaj o zadaném času metodou split a následně, po očištění hodnoty metodou „strip(„0“)“ byl údaj vložen do podmínky typu if/else.

Po vyřešení problémů byl test úspěšně automatizován za pomoci serveru Jenkins.

Test byl spuštěn i v provozu. Většina testů proběhla úspěšně. Neúspěšné běhy testu byly způsobeny vesměs problémovými externalitami, na které v postavení diplomanta a juniorního vývojáře nemám žádný vliv ani možnost je jakkoli ovlivnit.

Na závěr byl proveden odhad úspor automatizovaného spouštění daného testu ve srovnání s manuálním testováním. Vypočtená částka se může jevit pro zadavatele malá, nicméně ve výhledů častých pravidelných dlouhodobých opakování je významná i pro tak velký podnik, jakým zadavatel je.

Seznam použité literatury

LIU, Kun & JIANG, Jinmin & DING, Xiaohan & SUN, Hui. (2017). Design and Development of Management Information System for Research Project Process Based on Front-End and Back-End Separation. 338-342. 10.1109/CIIS.2017.55.

MOZILLA 1. Client-side tooling overview. Online. ©2024. In: mozilla. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Understanding_client-side_tools/Overview. [citováno: 2023-12-04].

MOZILLA 2. Introduction to the server side. Online. ©2024. In: mozilla. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Introduction. [citováno: 2023-12-04].

DATABÁZE NÁRODNÍ KNIHOVNY ČR. KTD – Úplné zobrazení záznamu. Online. ©2014. In: Databáze Národní knihovny ČR. Dostupné z: https://aleph.nkp.cz/F/?func=direct&doc_number=000000568&local_base=KTD. [citováno: 2024-02-23].

MARTINEK, David, 2012. Ladění a testování programů. Online. In: Fakulta informačních technologií v Brně. Dostupné z: https://www.fit.vutbr.cz/~martinek/clang/debug.html. [citováno: 2023-11-22].

ALI Zulfiqar, 2019. Mobile Application Testing Using Behavior-Driven Development Based Specifications for Android Platform. Graz. Doktorská diplomová práce. Graz University of Technology.

Dostupné z: https://diglib.tugraz.at/download.php?id=5f9018e86b037&location=browse. [citováno: 2023-12-29] .

PAN Jiantao: Software testing, Carnegie Mellon University, 18-849b Dependable Embedded Systems, Spring 1999.

ABRAM A. (editor) a kol.: Guide to software engineering Body of Knoeldge, SWEBOK, A Project of Softaware Engineering Coordinating Comitee, Chapter V: Software testing (autor Antonia Bertolino). str. 5-1, trial version, Dostupné z: https://cmapspublic.ihmc.us/rid=1K1K7VJ2J-1QMZQ0M-2SXJ/SWEBOK.pdf#page=88. [citováno: 2024-01-27].

STELLER Robin and STESS Marek, "Test Automation Architecture for Automotive Online-Services," Journal of Automation and Control Engineering, Vol. 7, No. 1, pp. 1-7, June, 2019. doi: 10.18178/joace.7.1.1-7.

THUMMALAPENTA Suresh & SINHA Saurabh & SINGHANIA Nimit & CHANDRA Satish. (2012). Automating test automation. Proceedings - International Conference on Software Engineering. 881-891. 10.1109/ICSE.2012.6227131.

BUREŠ, Miroslav; RENDA, Miroslav; DOLEŽEL, Michal; SVOBODA, Peter; GRÖSSL, Zdeněk et al., 2016. Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu. Profesionál. Praha: Grada. ISBN 978-80-247-5594-6.

DAKA Ermira; FRASER Gordon et al.: A Survey on Unit Testing Practices and Problems, 2014 IEEE 25th International Symposium on Software Reliability Engineering, 03-06 November 2014, Electronic ISBN:978-1-4799-6033-0, DOI: 10.1109/ISSRE.2014.11.

WHITTAKER J.A., “What is Software Testing? And Why Is It So Hard?” IEEE Software, vol. 17, no. 1, 2000, str. 70–79, ISSN 1937-4194.

ITNETWORK. Unit testy v Javě a JUnit. Online. ©2024. In: Itnetwork. Dostupné z: https://www.itnetwork.cz/java/testovani/java-unit-testy-v-junit. [citováno: 2023-12-19].

SWTESTOVÁNÍ. Druhy testování v procesu vývoje SW. Online. In: Sw testování. Dostupné z: http://test.swtestovani.cz/index.php?option=com_content&view=article&id=18:druhy-testovani-v-procesu-vyvoje-sw&catid=3:zaklady&Itemid=11. [citováno: 2023-12-19].

ISTWB GLOSSARY. Regresní testování. Online. In: ISTQB Glossary. Dostupné z: https://istqb-glossary.page/cz/regresni-testovani/. [citováno: 2023-12-21].

IT SLOVNÍK. Co je to Smoke testing. Online. ©2024. In: It slovnik. Dostupné z: https://it-slovnik.cz/pojem/smoke-testing. [citováno: 2023-12-17].

GUPTA Varuna, Vivek Sen Saxena: Software Testing: Smoke and Sanity, International Journal of Engineering Research & Technology (IJERT), Vol. 2 Issue 10, October – 2013, ISSN: 2278-0181.

VISUAL STUDIO CODE. Visual Studio Code FAQ. Online. ©2024. In: Visual studio code. Dostupné z: https://code.visualstudio.com/docs/supporting/faq. [citováno: 2023-12-19].

HISTORY COMPUTER. Pycharm vs VS code, which python ide wins. Online. ©2024. In: History computer. Dostupné z: https://history-computer.com/pycharm-vs-vs-code-which-python-ide-wins/. [citováno: 2023-12-19].

JETBRAINS. Pycharm. Online. In: Jetbrains. Dostupné z: https://www.jetbrains.com/pycharm/use-cases/. [citováno: 2023-12-19].

IT MUNI. GitLab. Online. ©2024. In: It Muni. Dostupné z: https://it.muni.cz/sluzby/gitlab. [citováno: 2024-04-08].

OPENAI. ChatGPT-3,5. May 12 Version. 2023-12-21. Dostupné z https://chat.openai.com/. [citováno: 2023-12-21].

JENKINS. Documentation. Online. In: Jenkins. Dostupné z: https://www.jenkins.io/doc/. [citováno: 2023-12-19].

BROWSERSTACK. Interactive mobile app testing. Online. ©2024. In: Browserstack. Dostupné z: https://www.browserstack.com/app-live?utm_source=google&utm_medium=cpc&utm_platform=paidads&utm_content=668760067900&utm_campaign=Search-Brand-EMEA-Navigational&utm_campaigncode=Core+1003799&utm_term=e+browserstack. [citováno: 2023-12-19] .

INDEED. Junior tester plat. Online. ©2024. In: Indeed. Dostupné z: https://cz.indeed.com/career/junior-tester/salaries. [citováno: 2024-04-10].

JOOBLE. Vývojář automatizovaných testů platy. Online. ©2024. In: Jooble. Dostupné z: https://cz.jooble.org/salary/v%C3%BDvoj%C3%A1%C5%99-automatizovan%C3%BDch-test%C5%AF. [citováno: 2024-04-10].