TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Automatizované testy webové aplikace Car Configurator

Diplomová práce

Studijní program:	N2612 Elektrotechnika a informatika
Studijní obor:	Mechatronika
Autor práce:	**Bc. Jiří Motl**
Vedoucí práce:	Ing. Mojmír Volf Ústav nových technologií a aplikované informatiky

Liberec 2022

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Zadání diplomové práce

Automatizované testy webové aplikace Car Configurator

Jméno a příjmení: **Bc. Jiří Motl**
Osobní číslo: M20000185
Studijní program: N2612 Elektrotechnika a informatika
Studijní obor: Mechatronika
Zadávající katedra: Ústav nových technologií a aplikované informatiky
Akademický rok: **2021/2022**

Zásady pro vypracování:

1. Proveďte rešerši možných způsobů automatizovaných testů pro webové aplikace a zhodnoťte jejich použitelnost.
2. Seznamte se s aktuálním řešením testování webové aplikace ve Škoda Auto a zhodnoťte výhody a nevýhody daného řešení.
3. Proveďte návrh a následně naprogramujte skript pro testování zobrazovaných dat a informací ve webové aplikaci Car Configurator.
4. Ověřte a demonstrujte správnou, spolehlivou a efektivní funkci Vámi vytvořeného skriptu pro testování webové aplikace.

Rozsah grafických prací: dle potřeby dokumentace
Rozsah pracovní zprávy: 40-50 stran
Forma zpracování práce: tištěná/elektronická
Jazyk práce: Čeština

Seznam odborné literatury:

[1] BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽEL, Peter SVOBODA, Zdeněk GRÖSSL, Martin KOMÁREK, Ondřej MACEK a Radoslav MLYNÁŘ. Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu. Praha: Grada, 2016. Profesionál. ISBN 978-80-247-5594-6.
[2] LUCCA, Giuseppe a Anna FASOLINO. Web Application Testing. MENDES, Emilia a Nile MOSLEY. Web Engineering. Berlin: Springer Berlin Heidelberg, 2006, s. 219-260. ISBN 978-3-540-28218-1.
[3] SUMMERFIELD, Mark. Python 3: výukový kurz. 2. vydámí. Přeložil Lukáš KREJČÍ. Brno: Computer Press, 2021, 584 s. ISBN 978-80-251-5030-6.

Vedoucí práce: Ing. Mojmír Volf
Ústav nových technologií a aplikované informatiky

Datum zadání práce: 12. října 2021
Předpokládaný termín odevzdání: 16. května 2022

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

Ing. Josef Novák, Ph.D.
vedoucí ústavu

V Liberci dne 19. října 2021

Prohlášení

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

Jsem si vědom toho, že na mou diplomovou 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 diplomové práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li diplomovou 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 diplomová 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.

16. května 2022 Bc. Jiří Motl

Automatizované testy pro webovou aplikaci Car Configurator

Abstrakt

Diplomová práce se zaměřuje na vývoj skriptu pro automatizované testování webové aplikace a na zhodnocení využitého řešení ve firmě Škoda Auto. V práci jsou popsány postupy vývoje softwaru a následně možnosti jejich testování. Pro automatizované testování webové aplikace je využito softwaru Robot Framework včetně knihovny SeleniumLibrary, která slouží právě pro testy na webových stránkách. Funkce vyvinutého skriptu je popsána v praktické části společně s výsledky, které bylo možné s automatizovanými testy dosáhnout. Součástí práce je i doporučení pro firmu Škoda Auto v rámci možnosti spolupráce na automatizovaných testech i mezi různými odděleními.

Klíčová slova: Agilní vývoj, Robot Framework, Python, Automatizované testy, Testy webových aplikací

Automated tests of the Car Configurator web application

Abstract

The diploma thesis focuses on the development of a script for automated testing of a web application and on the evaluation of the used solution in the company Škoda Auto. The thesis describes the procedures of software development and then the possibilities of their testing. Robot Framework software is used for automated testing of the web application, including the SeleniumLibrary, which is used for testing on websites. The function of the developed script is described in the practical part together with the results that could be achieved with automated tests. Part of the work is also a recommendation for the company Škoda Auto within the possibility of cooperation on automated tests between different departments.

Keywords: Agile Development, Robot Framework, Python, Automated Tests, Web Application Tests

Poděkování

Rád bych zde poděkoval vedoucímu práce panu Ing. Mojmíru Volfovi za vedení práce a dále bych také rád poděkoval konzultantovi práce za Škoda Auto a.s., Ing. Štěpánu Moníkovi.

Obsah

Seznam zkratek

API	Application Programming Interface - rozhraní pro programování aplikací
CC	Car Configurator
EL	Elektrický pohon, elektrická motorizace
FDD	Feature Driven Development - metodika vývoje softwaru
HTML	Hyper Text Markup Language - hypertextový značkovací jazyk
HTTP	Hypertext Transfer Protocol - internetový protokol určený pro komunikaci
IDE	Integrated Development Environment - vývojové prostředí
IT	Informační technologie
MPI	Multi Point Injection, označení pro atmosférické benzínové motory s vícebodovým vstřikováním paliva
OS	Operační systém
RF	Robot Framework - testovací nástroj
SW	Software
TDI	Turbocharged Direct Injection, označení přeplňovaného vznětového motoru s turbodmychadlem a přímým vstřikováním nafty
TDD	Test-Driven Development - vývoj řízený testy
TSI	Twincharged Stratified Injection, označení pro přeplňované benzínové motory s technologií přímého vstřikování paliva
URL	Uniform Resource Locator - jednotný lokátor zdroje
WWW	Worl Wide Web - celosvětová síť
XML	Extensible Markup Language - rozšiřitelný značkovací jazyk
XP	Extrémní programování

Seznam obrázků

Seznam tabulek

Úvod

V dnešní době, kde moderní technologie jsou všude kolem nás, jsou kladeny velké nároky na vývoj softwaru, většinou tedy aplikací. Jednou z fází vývoje softwaru je i samotné testování produktu, které přináší vývojářům důležitou zpětnou vazbu. Proto se práce zaměřuje na možné metodiky, jak ve vývoji softwaru, tak i právě v testování a to konkrétně automatizovaného testování. Automatizované testování rozsáhlých projektů přináší totiž obrovskou časovou úsporu a většinou dosahuje i lepší spolehlivosti než manuální testování.

Jednou z rozsáhlých webových aplikací firmy Škoda Auto je i aplikace Car Configurator. Tato aplikace má sloužit zákazníkovi ve smyslu možnosti konfigurace libovolného vozu dle jeho představ a to do posledního detailu. Cílem práce je tedy zhodnotit aktuální využívané metodiky testování webové aplikace Car Configurator. Dále zhodnotit i možnost využití testovacích kapacit i skrze jiná oddělení ve firmě Škoda Auto. A v neposlední řade vyvinout skripty pro automatizované testování této aplikace, především v oblasti zobrazených dat. Motivací je hlavně spokojenost zákazníků, která je pro celou firmu velmi klíčová.

V úvodní části se práce zaměřuje na popis různých metodik vývoje softwaru, jejich výhod a nevýhod. Zaměřuje se také hlavně na automatizované testy vyvíjené pomocí nástroje Robot Framework, jelikož toto řešení je ve firmě Škoda Auto využité. Rešerše pojednává i o dostupných programovacích jazycích, webových aplikacích včetně webu samotného a také o vývojovém prostředí a možnostech sdílení testovacích souborů.

V praktické části je zhodnoceno aktuální řešení a možnosti využití spolupráce skrze různá oddělení ve Škoda Auto pro efektivnější využití testovacích kapacit. Dále se praktická část zaměřuje na práci s Robot Framework, kde popisuje jakým způsobem se s tímto nástrojem pro automatizované testy pracuje. Dále je popsáno jakým způsobem byly vyvinuty testy webové aplikace pro kontrolu zobrazovaných hodnot. A v závěrečné části je uvedeno jaké chyby byly v aplikaci detekovány a jakým způsobem byla řešena jejich náprava. Poslední kapitolou je následně celé shrnutí dosažených výsledků.

1 Vývoj SW

1.1 Agilní metodiky

Agilní metodikou se rozumí v oblasti IT metody pro vyvíjení softwaru. Metodika je založena na postupném interaktivním vývoji. Tímto způsobem je umožněn rychlý vývoj softwaru a zároveň je umožněno rychle a dobře reagovat na změnu požadavků během vývojového cyklu. Software je předáván zákazníkovi a následně se upravuje dle jeho zpětné vazby. Agilní metodiky ale našly své uplatnění nejen v IT, u vývoje softwaru, ale také i v jiných oblastech jako marketing, finance, telekomunikační služby a podobně. Protikladným přístupem k agilním metodikám je vodopádový model.

Co se týče historie softwarového vývoje, tak se dříve využívaly těžké metody, které jsou kritizovány především za to, že nebyly schopné reagovat na změny požadavků zákazníka z důvodu využití již zmíněného vodopádového modelu. U tohoto modelu je totiž nutné, aby všechny konkrétní požadavky byly striktně zadány již na počátku vývoje jako požadavek. Proto se v polovině devadesátých let začaly objevovat odlehčené metody, které se navrací k vývojovým praktikám z počátků vývoje softwaru. Mezi tyto odlehčené metody patří například Scrum, Crystal Clear, Extrémní programování či Vývoj řízený vlastnostmi.

Samotný pojem agilní metodiky se začal využívat až na začátku 3. tisíciletí, kdy se tehdejší odborníci z oblasti vývoje softwaru sešli a diskutovali o možných metodách vývoje softwaru. Výsledkem tohoto setkání bylo sepsání manifestu agilního programování, ve kterém definují různé přístupy k vývoji, dnes známe jako Agilní programování. [1, 2]

1.1.1 Manifest agilního programování

Mimo jiné je v manifestu definováno 12 principů agilního programování [1].

- Nejvyšší prioritou je uspokojit zákazníka průběžnými a rychlými dodávkami kvalitního softwaru.
- Změnové požadavky jsou vítány, dokonce i v průběhu vývoje. Agilní procesy je zpracují tak, aby zákazníkovi přinášely konkurenční výhody.
- Dodávejte fungující software často, v intervalech týdnů až měsíců. Upřednostňujte kratší intervaly dodání.

- Lidé z businessu a vývojáři musí spolupracovat každý den během celého projektu.
- Pro práci na projektu vybírejte motivované jedince. Dejte jim prostředí a podporu, kterou potřebují, a důvěřujte jim, že práci dokončí.
- Nejúčinnější metoda sdílení informací vývojářskému týmu (i uvnitř tohoto týmu) je osobní setkání.
- Fungující software je hlavním měřítkem postupu vývoje.
- Agilní procesy podporují udržitelný vývoj. Sponzoři, vývojáři i uživatelé by měli být schopní dodržovat stálý výkon, dokud je třeba.
- Průběžná pozornost věnovaná technické dokonalosti a dobrému návrhu posiluje agilní přístup.
- Základem je jednoduchost – umění co nejvíce práce vůbec nedělat.
- Nejlepší architektury, požadavky a návrhy vznikají v týmech, které se samy organizují.
- Tým v pravidelných intervalech vyhodnocuje svou práci a upravuje své postupy tak, aby byl co nejefektivnější.

1.1.2 Agilní metodiky dnes

Agilní metody už dnes exponovaly do celého světa, v samotné Americe již dnes většina vývoje softwaru je právě pomocí této metodiky. Metodika přináší spoustu výhod, je vlastně bezpečnější pro obě strany, jak pro odběratele, tak pro dodavatele. Odběratel hlavně nemusí mít jasno hned na počátku vývoje, jaké mají být přesné požadavky a může je klást až v průběhu. Jelikož samotný vývoj funguje na intervaly, takzvané 14 denní „sprinty“, tak nemusí být předem jasné, kolik času vývoj zabere. Z toho pak nelze předem stanovit ani cenu, nicméně během vývoje obě strany ví, jaká je situace a dokážou toto předpovídat. Z toho lze vyvodit, že mají situaci pořád ve svých rukou a mohou včas reagovat. Takovéto představy bohužel u vodopádového modelu nejsou vůbec možné.

Každá metodika má samozřejmě i své nevýhody a úskalí a ani v tomto případě tomu není jinak. Je totiž nutné do samotného procesu vývoje aktivně zapojit klienta, od kterého jsou vyžadovány alespoň základní znalosti jak technické, tak metodické. Další nevýhodou je již zmíněná cena, která se samozřejmě odvíjí od průběhu vývoje a nelze ji předem přesně definovat. S tím souvisí i čas, stejně jako není možné odhadnout cenu, což je způsobené právě tím, že nelze odhadnout jakou dobu bude stráveno nad vývojem daného softwaru. [1, 2]

1.1.3 Příklady agilních metodik

- Extrémní programování (XP)

  Extrémní programování, neboli XP je pravděpodobně nejznámější a nejčastější metodou agilního vývoje. Je využíváno principu dodávání softwaru ve velmi krátkých časových intervalech. Programuje se v danou chvíli jen to co je opravdu nezbytné a cílem je tvořit jednoduchý a jasný kód. Vývojáři počítají s průběžnými změnami požadavků během vývoje a důležitou součásti je i neustálá komunikace mezi dodavatelem a zákazníkem. [1]

- Scrum

  Metodou Scrum se rozumí metoda založená na každodenním setkávání týmu. Při těchto setkáních jsou podány informace ohledně předchozího dne a dosažených cílů včetně plánu na den stávající. Těmto setkáním se říká „daily standups“. Opět se jedná o interaktivní vývoj ve sprintech, které trvají přibližně 2 týdny. V určitých případech, dle domluvy, mohou být sprinty delší, ale i kratší a častější. V tomto období se výsledky předvádějí stakeholderům, což jsou zákazníci, sponzoři, management, investoři a další. Jejich zpětná vazba pak poskytuje možnosti rychle reagovat na změny v jejich požadavcích. V této metodice jsou důležité pojmy jako Product Owner, jehož účelem je komunikace se zákazníkem, dále Scrum Master, který je členem vývojového týmu a zajišťuje jeho správný chod a fungování. Pak jsou zde členové týmu, neboli Scrum Team Member a posledním důležitým pojmem je Scrum Alliance, který má na starost vývoj a školení dané metodiky. [1]

- Další metody

  Další metody, které se liší svým přístupem k vývoji softwaru, jsou FDD (Feature Driven Development), Lean Development, TDD (Test Driven Development), Crystal metodiky a další. U FDD si práci programátoři nevybírají, ale je jim přidělena. Lean development je spíše souhrnem pravidel než metodou a slouží pro zrychlení vývojového procesu a efektivitu. TDD nejdříve navrhuje testy a následně se píše kód, který samotným testem projde. A u Crystal Metodik se jedná o metodiky, které jsou přizpůsobeny svému konkrétnímu účelu. [1]

1.1.4 Právní rámec

Mezi dodavatelem a zákazníkem samozřejmě musí stát smlouva. V tomto případě jsou nejčastějším případem dva typy smluv.

- Fixed-Time Fixed-Price

  Jedná se o smlouvu, která definuje, že vyvinutý software má být dodán za pevně stanovenou částku v pevně stanoveném čase. Z právního hlediska se jedná o smlouvu o dílo. [1]

- Time and Material

  Time and Material je smlouva nedefinující dílo, ale pouze služby, které bude zákazníkovi poskytovat dodavatel. V případě dodávek softwaru se může jednat o programování, testování a podobně. Na základě jednotlivých objednávek jsou poskytovány jednotlivé služby. Pro agilní vývoj se tedy hodí využít tuto smlouvu. [1]

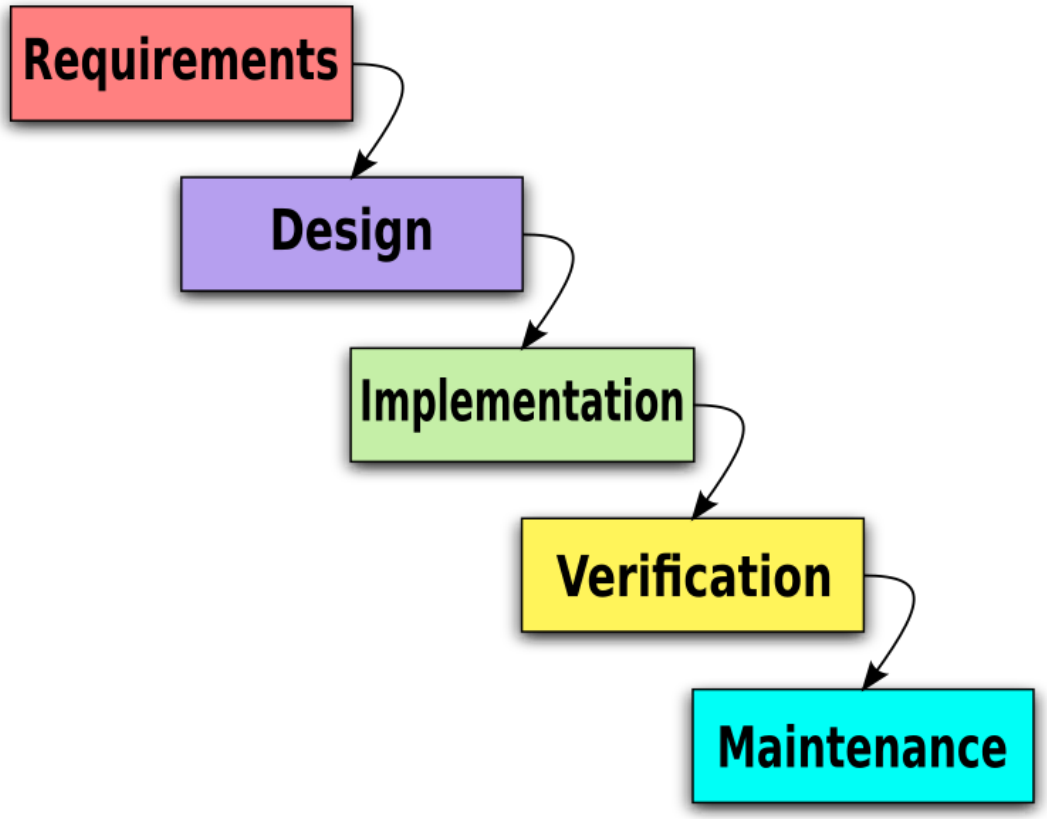
1.2 Waterfall

Vodopádový model, neboli waterfall je v podstatě předchůdcem agilních přístupů k tvorbě aplikací. Jedná se o sekvenční proces vývoje aplikace, jehož jednotlivé části na sebe plynule navazují a vycházejí ze sebe. Z toho plyne, že žádnou etapu nelze přeskočit ani přerušit. Samotný posun v procesu je závislý na úspěšnosti předchozí fáze, proto je zde větší riziko úspěšnosti než u agilních metodik. V principu zákazník vůbec nemusí zasahovat do vývoje, jeho hlavním úkolem je sestavit požadavky na aplikaci již na začátku procesu a to do co nejmenšího detailu. Model této metodiky je vizuálně zobrazen na obrázku 1.1. [3]

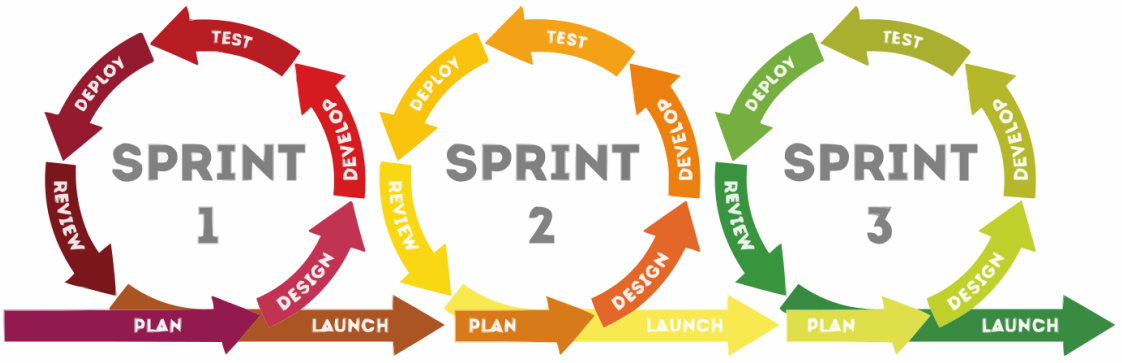
Úskalím této metody jsou již zmíněné požadavky, které musejí být specifikovány hned na začátku vývoje a to přesně. Proto pokud během vývoje nastanou změny požadavků od zákazníka, tak je mnohem těžší na tyto změny reagovat a většinou to má za důsledek celkové prodloužení vývoje a vyšší náklady. V této metodě jsou tedy fixní požadavky a proměnné jsou zde čas a náklady. Další nevýhodou tohoto přístupu je míra rizika. Jelikož musí vývoj proběhnout přes všechny fáze a to stoprocentně úspěšně, je zde možnost, že dojde během vývoje k zásadnímu problému. Aplikace se tak dostává k zákazníkovi až po průběhu všech cyklů a teprve poté má nějakou hodnotu. Do této doby roste riziko neúspěchu. [3]

Vždy je zde proto zajištěna maximální funkcionalita, avšak na úkor horšího provádění změn. Aplikace se typicky ke klientovi dostane až ve fázi testování, kdy již může být provádění určitých změn komplikovanější než v rané fázi vývoje. Největší úskalí zde proto spočívá v tom, že klient nemusí předem znát všechny své požadavky, jelikož si nedostatky v zadání uvědomí až po zhlédnutí finální verze. [4]

Přesným opakem je právě agilní vývoj, kde fixní jsou náklady a čas a variabilní je výsledek. Na začátku vývoje jsou sice stanoveny celkové požadavky na aplikaci, ale jsou rozděleny do tzv. sprintů kde jsou vyvíjeny a testovány jednotlivé dílčí funkce aplikace. S každým sprintem tedy aplikace nabývá své hodnoty a postupně jsou do aplikace implementovány další a další funkce. Proto se tímto přístupem i snižuje riziko neúspěchu. Model tohoto agilního vývoje je zobrazen na obrázku 1.2.



Obrázek 1.1: Model vodopádu (zdroj: commons.wikimedia.org)



Obrázek 1.2: Model agilního vývoje (zdroj: netmagnet.cz)

2 Programovací jazyky

2.1 C#

C# neboli C Sharp je vysokoúrovňový objektově orientovaný programovací jazyk, který byl vyvinut společností Microsoft. Tento programovací jazyk je založen na programovacích jazycích C++ a Java. Programovací jazyk C# je využíván pro tvorbu formulářových, grafických, databázových nebo webových aplikací. V tomto programovacím jazyce se nejčastěji objevují objekty a instance třídy. V informatice je objekt označení pro datovou strukturu nebo proměnnou, která je v paměti vytvořena a má tak přidělené své místo a má nějakou hodnotu. Objekt je v objektově orientovaném programování instancí třídy, myšleno nějaký konkrétní datový objekt, který byl odvozen od nějaké konkrétní třídy neboli vzoru. Každá proměnná musí mít vždy svůj datový typ. Může to být text, znak, číslo a podobně, v podstatě záleží, k čemu je proměnná využívána, dle toho se jí přiřazuje datový typ.

Jelikož programovací jazyk C# je statický typový systém, je nutné, aby každá proměnná vyskytující se v programu byla nejdříve deklarována, a to včetně datového typu. Datové typy v programovacím jazyce C# jsou celočíselné Sbyte, Byte, Short, Ushort, Int, Uint, Long, Ulong, desetinné Float a Double, ostatní hodnotové Char, Decimal a Bool a v neposlední řadě Referenční. Referenční typy jsou odlišné od hodnotových datových typů, jelikož samy nenesou hodnotu ale pouze referenci, která ukazuje na hodnotu uloženou v paměti. Proto dva různé referenční datové typy mohou ukazovat na stejnou hodnotu, pokud pomocí první reference hodnotu upravíte, druhým referenčním typem uvidíte již upravenou hodnotu. Navíc referenční datové typy mohou s sebou nést metody. Typickým příkladem je datový typ „string“. String je řetězec charakterů a má právě své metody jako například „Replace“.

V objektově orientovaném programování je základním konstrukčním prvkem třída. Třída slouží jako předpis pro generování objektu jako instance. Mnohdy odpovídá pojmům z reálného světa, typickým příkladem je třída „Zákazník“. Třída definuje atributy (Jméno, Adresa, Objednávky a podobně) a metody neboli funkce (Vložit do košíku, Zaplatit, Změnit Adresu atd.). Objektů typu třída Zákazník může vzniknout několik, ale každý tento objekt má vlastní hodnoty atributů a na každý tento objekt lze zavolat metodu definovanou v třídě Zákazník. Sada hodnot, nejčastěji sada hodnot objektů dané třídy je datový typ kolekce. Cílem kolekce je právě sloužit jako úložiště objektů. Například kolekce „Seznam zaměstnanců“ muže obsahovat všechny objekty typu „Zaměstnanec“. Poté lze přistupovat ke každému zaměstnanci z kolekce zvlášť. Kolekce má výhodu oproti poli v tom, že jeho velikost není striktně daná,

lze tak do ní přidat libovolné množství objektů a není potřeba předem definovat její velikost.

2.2 Python

Python je vysokoúrovňový programovací jazyk, jedná se o jazyk skriptovací. Podporuje programování objektové včetně procedurálního, imperativního i funkcionálního programování. Objektové programování je obdobné jako u programovacího jazyka C#, popsáno v kapitole 2.1. Jazyk Python je vyvíjen jako open source, takže je nabízen zdarma pro většinu platforem jako například Windows, Unix, macOS, Android a podobně. Postupným vývojem již vznikly tři verze, Python 1, Python 2 a Python 3. Python 1 se již nepoužívá, jeho poslední verze byla vydána v roce 2000, Python 2 je již v útlumu, jeho poslední verze byla vydána v roce 2020. Naopak Python 3 je velmi aktivní verze, která byla prvně vydána v roce 2008 a stále je ve vývoji. [5]

Mezi důležité vlastnosti programovacího jazyka Python patří jeho dynamika. Má velké možnosti a programátor si tak při tvorbě programu může vybírat z několika paradigmat, které program nabízí ačkoli některé jen v omezené míře. Oproti ostatním programovacím jazykům je kód programu v jazyce Python krátký a velmi dobře čitelný. Proto další významnou vlastností je jeho jednoduchost, zejména z hlediska učení. Proto je tento jazyk vhodný jak pro začátečníky, tak pro tvorbu rozsáhlých programů, které jsou díky jednoduchosti velice produktivní. Mezi hlavní datové typy pro programovací jazyk Python patří boolean, který může nabývat hodnot true nebo false, pak čísla, která můžou být celá (integer), reálná (float), zlomky (fraction) anebo čísla komplexní. Dalším datovým typem jsou řetězce což jsou posloupnosti znaků Unicode. Mezi ostatní hlavní datové typy poté patří seznamy, n-tice, množiny a slovníky jenž vždy reprezentují posloupnosti nebo kolekce. [6, 5]

2.3 Java

Java je vysokoúrovňový, na třídách založený, objektově orientovaný programovací jazyk, který je navržen tak, aby měl co nejméně implementačních závislostí. Jedná se o univerzální programovací jazyk určený k tomu, aby umožnil programátorům napsat jeden kód v tomto jazyce a spustit ho kdekoli. Což znamená, že zkompilovaný kód Java může běžet na všech platformách, které Javu podporují, bez nutnosti rekompilace. Java aplikace jsou obvykle kompilovány do bajtového kódu, který lze spustit na jakémkoli virtuálním stroji Java (JVM) bez ohledu na základní počítačovou architekturu. Syntaxe Javy je podobná C a C++, ale má méně nízkoúrovňových zařízení než kterýkoli z nich. Runtime Java poskytuje dynamické funkce (jako je reflexe a modifikace runtime kódu), které obvykle nejsou dostupné v tradičních kompilovaných jazycích. Od roku 2019 byla Java jedním z nejpopulárnějších programovacích jazyků používaných podle GitHubu, zejména pro webové aplikace klient-server. [7]

3 Automatizované testování

V oblasti testování softwaru se automatizovaným testováním myslí využití softwaru k porovnávání získaných výsledků s výsledky očekávanými. Automatizované testy se mohou využívat ať už k opakovaným účelům, tak i k jednoúčelovým potřebám. U opakovaných účelů se automatizace využívá především k vyšší efektivitě, než by dosáhl člověk manuálním testováním. Nepřináší to ale jen efektivitu, výhodou může být i rychlost, kterou je automatizovaný test schopen požadavky kontrolovat. Samozřejmě tato metoda s sebou nese i nějaké nevýhody, jednou z nich je udržovatelnost. Za každým testem stále musí stát vývojář, který musí řešit možné nedostatky a také případné úpravy. Proto se často i nabízí otázka zda je potřeba vytvářet automatizované testy, jestli vůbec přinášejí nějakou výhodu. Druhou zmíněnou kategorií je provedení jednoúčelového, popřípadě jednoprůchozího testu. Tato varianta se určitě nabízí ve chvíli, kdy je potřeba kontrola, která by člověku mohla dělat obrovské problémy nebo byla natolik složitá, že by člověk sám takové testy nemohl provádět. [8]

3.1 Robot framework

Robot Framework je open source automatizační framework. Může být použit pro automatizaci testování a automatizaci robotických procesů (RPA) a je podporován nadací Robot Framework Foundation. Mnoho předních společností používá tento nástroj při vývoji softwaru. Robot Framework je otevřený a rozšiřitelný, lze ho integrovat prakticky s jakýmkoli jiným nástrojem a vytvořit tak výkonná a flexibilní řešení automatizace. Je zdarma k použití bez licenčních poplatků, má snadnou syntaxi využívající lidsky čitelná klíčová slova. Jeho schopnosti lze rozšířit o knihovny implementované pomocí Pythonu, Javy nebo mnoha dalších programovacích jazyků. Robot Framework má kolem sebe bohatý ekosystém, který se skládá z knihoven a nástrojů, které jsou vyvíjeny jako samostatné projekty. Jednou z klíčových knihoven pro automatizované testování webových stránek či aplikací je SeleniumLibrary. [9]

3.1.1 BuiltIn Knihovny

Robot Framework má v sobě již zabudované některé knihovny. Zde je jejich seznam včetně stručného popisu:

- BuiltIn obsahující obecná často potřebná klíčová slova. Je importována automaticky, a proto je vždy dostupná.
- Collections, obsahuje klíčová slova pro práci se seznamy a slovníky.
- Date Time, která podporuje vytváření a ověřování hodnot data a času a také výpočty mezi nimi.
- Dialogs, podporující pozastavení provádění testu a získávání informací od uživatelů.
- Operating System umožňující provádění různých úloh souvisejících s operačním systémem.
- Process, která podporuje spouštění procesů v systému.
- Remote, neboli část rozhraní vzdálené knihovny. Nemá žádná vlastní klíčová slova.
- Screenshot, knihovna která poskytuje klíčová slova pro zachycení a uložení snímků obrazovky pracovní plochy.
- String, což je knihovna pro manipulaci s řetězci a ověřování jejich obsahu.
- Telnet, knihovna podporující připojení k serverům Telnet a sloužící k provádění příkazů na otevřených připojeních.
- XML knihovna pro ověřování a úpravy dokumentů XML.

3.1.2 Selenium Library

SeleniumLibrary je testovací knihovna pro Robot Framework, která interně používá populární webový testovací nástroj Selenium. Poskytuje výkonnou kombinaci jednoduché syntaxe testovacích dat a podpory pro různé prohlížeče. Kromě standardního webového testování knihovna podporuje také testování aplikací Adobe Flex/Flash. SeleniumLibrary interně používá Selenium Remote Controller API, které je součástí Selenium 1. Pro možnost využívání nového rozhraní Selenium 2 WebDriver API, je třeba se podívat na Selenium2Library, což je ve většině případů náhrada za SeleniumLibrary. [10]

3.1.3 Excel Library

Tato testovací knihovna Excel Library poskytuje klíčová slova umožňující otevírání, čtení, zápis a ukládání excelových souborů z Robot Framework. V případě testování webových aplikací je tato knihovna vhodná pro čtení ze souboru, kde lze nastavit například jaké testy mají být provedeny, na jakých webových stránkách a podobně. Další využívanou funkcí je naopak zápis, pomocí kterého lze cíle dosažené testem uložit do souboru pro budoucí analýzu.

3.1.4 Nástroje

Robot Framework s sebou nese i zabudované nástroje, které lze využívat. Zde je jejich seznam s krátkým popisem:

- Rebot, nástroj pro generování protokolů a reportů na základě XML výstupů. Slouží i pro kombinování více výstupů dohromady.
- Libdoc, nástroj pro generování dokumentace klíčových slov pro testovací knihovny a zdrojové soubory.
- Testdoc, nástroj který generuje dokumentaci HTML na vysoké úrovni na základě testovacích případů.
- Tidy, nástroj pro čištění a změnu formátu testovacích datových souborů.

3.2 Git

Git je software pro sledování změn v libovolné sadě souborů, obvykle je používaný pro koordinaci práce mezi programátory, kteří spolupracují na vývoji zdrojového kódu, během vývoje softwaru. Mezi jeho cíle patří rychlost, integrita dat a podpora distribuovaných, nelineárních pracovních postupů (tisíce paralelních větví běžících na různých systémech). [11]

Git vytvořil Linus Torvalds v roce 2005 pro vývoj linuxového jádra, přičemž na jeho počátečním vývoji přispěli další vývojáři jádra. Od roku 2005 je hlavním správcem Junio Hamano. Stejně jako u většiny ostatních distribuovaných systémů správy verzí a na rozdíl od většiny systémů klient-server, je každý adresář Git na každém počítači plnohodnotným úložištěm s kompletní historií a plnými možnostmi sledování verzí. Navíc je nezávislí na síťovém přístupu nebo centrálním serveru. Git je bezplatný a open source software distribuovaný pouze pod licencí GPL-2.0. [11]

3.2.1 Github

GitHub, Inc. je poskytovatel internetového hostingu pro vývoj softwaru a správu verzí pomocí softwaru Git. Nabízí funkci distribuovaného řízení verzí a správy zdrojového kódu (SCM) systému Git a navíc vlastní funkce. Poskytuje řízení přístupu

a několik funkcí pro spolupráci, jako je sledování chyb, požadavky na funkce, správa úkolů, nepřetržitá integrace a wiki pro každý projekt. Se sídlem v Kalifornii je od roku 2018 dceřinou společností Microsoftu. Běžně se používá k hostování open source projektů. K listopadu 2021 měl GitHub více než 73 milionů vývojářů a více než 200 milionů úložišť (včetně nejméně 28 milionů veřejných úložišť). Od listopadu 2021 je to také největší hostitel zdrojového kódu. [12]

3.3 PyCharm a JetBrains

PyCharm je integrované vývojové prostředí (IDE) používané v počítačovém programování, konkrétně pro programovací jazyk Python. Vyvíjí ho česká společnost JetBrains (dříve IntelliJ). Poskytuje analýzu kódu, grafický debugger, integrovaný tester jednotek, integraci se systémy správy verzí (VCSes) a podporuje vývoj webu s Django a také datovou vědu s Anacondou.

PyCharm je multiplatformní s verzemi pro Windows, macOS a Linux. Komunitní edice je vydána pod licencí Apache a existuje také edice Professional s extra funkcemi, která je vydaná pod proprietární licencí financovanou z předplatného a existuje také vzdělávací verze. Klíčovými parametry prostředí PyCharm jsou:

- Pomoc a analýza kódování s dokončováním kódu, zvýrazněním syntaxe a chyb, integrací linter a rychlými opravami.
- Navigace v projektu a kódu, specializovaná zobrazení projektu, zobrazení struktury souborů a rychlé přeskakování mezi soubory, třídami, metodami a použitími.
- Python refactoring, zahrnující přejmenování, extrahování metody, zavedení proměnné, zavedení konstanty, vytažení nahoru, posunutí dolů a další.
- Podpora webových rámců Django, web2py a Flask.
- Integrovaný Python debugger.
- Integrované testování jednotek s pokrytím kódu řádek po řádku.
- Vývoj Google App Engine Python (pouze u profesionální edice).
- Integrace správy verzí, jednotné uživatelské rozhraní pro Mercurial, Git, Subversion, Perforce a CVS se seznamy změn a sloučením.
- Podpora vědeckých nástrojů jako matplotlib, numpy a scipy.
- Soutěží především s řadou dalších IDE orientovaných na Python, včetně PyDev od Eclipse a šířeji zaměřeného IDE Komodo.

4 Webové aplikace

Webová aplikace je aplikační software, který běží na webovém serveru, na rozdíl od počítačových softwarových programů, které jsou spouštěny lokálně v operačním systému (OS) zařízení. Webové aplikace jsou přístupné uživateli prostřednictvím webového prohlížeče s aktivním síťovým připojením. Tyto aplikace jsou programovány pomocí struktury modelované klient-server. [13]

Uživateli ("klientovi") jsou poskytovány služby prostřednictvím externího serveru, který je hostován třetí stranou. Příklady běžně používaných webových aplikací zahrnují: web-mail, online maloobchodní prodej, online bankovnictví a podobně. Webovou aplikací je i aplikace Car Configurator, která běží na serveru společnosti Škoda Auto a kde klientem jsou zákazníci, kteří si mohou konfigurovat vůz dle svých představ přímo z webového prohlížeče. [13]

4.1 WWW (world wide web)

World Wide Web (WWW), běžně známý jako web, byl původně hypertextový systém správy dokumentů přístupný přes internet. Od té doby se vyvinul ve světově dominantní softwarovou platformu. Zdroje webu jsou přenášeny prostřednictvím protokolu HTTP (Hypertext Transfer Protocol), uživatelé k nim mohou přistupovat prostřednictvím softwarové aplikace nazývané webový prohlížeč a jsou publikovány softwarovou aplikací nazývanou webový server. [14]

Webové zdroje mohou být jakýkoli typ stažených médií, ale webové stránky jsou hypertextové dokumenty formátované v jazyce HTML (Hypertext Markup Language). Speciální syntaxe HTML zobrazuje vložené hypertextové odkazy s adresami URL, což uživatelům umožňuje přejít na jiné webové zdroje. Kromě textu mohou webové stránky obsahovat odkazy na obrázky, video, audio a softwarové komponenty, které se buď zobrazují nebo interně spouštějí ve webovém prohlížeči uživatele za účelem vykreslování stránek nebo streamů multimediálního obsahu. [14]

Webové stránky tvoří více webových zdrojů se společným tématem a obvykle společným názvem domény. Webové stránky jsou uloženy v počítačích, na kterých běží webový server, což je program, který reaguje na požadavky provedené přes internet z webových prohlížečů spuštěných na počítači uživatele. Obsah webových stránek může poskytovat vydavatel nebo může být tvořen interaktivně z obsahu vytvářeného uživateli. Webové stránky jsou poskytovány z mnoha informačních, zábavních, komerčních a vládních důvodů. [14]

4.2 HTML

HyperText Markup Language neboli HTML je standardní značkovací jazyk pro dokumenty určené k zobrazení ve webovém prohlížeči. Webové prohlížeče přijímají dokumenty HTML z webového serveru nebo z místního úložiště a převádějí dokumenty na multimediální webové stránky. HTML sémanticky popisuje strukturu webové stránky a původně obsahovalo vodítka pro vzhled dokumentu. [15]

HTML elementy jsou stavební kameny HTML stránek. Pomocí konstrukcí HTML lze do vykreslené stránky vložit obrázky a další objekty, jako jsou interaktivní formuláře. HTML poskytuje prostředky k vytváření strukturovaných dokumentů tím, že označuje strukturální sémantiku textu, jako jsou nadpisy, odstavce, seznamy, odkazy, citace a další položky. Elementy HTML jsou ohraničeny značkami zapsanými pomocí lomených závorek. Tagy jako `<img/>` a `<input/>` přímo uvádějí obsah na stránku. Jiné značky, jako je <p>, obklopují a poskytují informace o textu dokumentu a mohou obsahovat další značky jako dílčí prvky. Prohlížeče nezobrazují značky HTML, ale používají je k interpretaci obsahu stránky. [15]

HTML může vkládat programy napsané ve skriptovacím jazyce, jako je JavaScript, což ovlivňuje chování a obsah webových stránek. Zahrnutí CSS definuje vzhled a rozvržení webového obsahu. [15]

5 Cíle praktické části

5.1 Skript pro automatizované testování webové aplikace Car Configurator

Hlavním cílem této práce je dosáhnout plně automatizovaného skriptu, který bude dohlížet nad webovou aplikací Car Configurator a bude reportovat nalezené chyby. Jedná se především o chyby viditelnosti kol a hodnot CO_2. Je totiž naprosto nežádoucí aby zákazník, který si chce nakonfigurovat vůz dle svých představ, nedostal možnost výběru kol. Tato chyba většinou není způsobena chybnou funkcí aplikace, ale chybou importéra, který za zobrazené data v aplikaci zodpovídá. Možným řešením je samozřejmě i manuální kontrola, nicméně možností konfigurací je velké množství a tato monotónní práce není pro člověka vhodná. Navíc při tolika možnostech se předpokládá, že i člověk udělá chybu a nemusela chyba aplikace by tak nemusel být detekována. Druhou stránkou je i samotný čas, kde se dá předpokládat, že automatizovaný test je schopen všechny konfigurace zkontrolovat mnohem rychleji. To je způsobené hlavně tím, že nemusí čekat na animace, které v aplikaci jsou. Zobrazené hodnoty může ze stránky číst ještě dříve, než jsou reálně vidět a samozřejmě i reakční doba je mnohem nižší než u člověka.

5.2 Spolupráce oddělení ve Škoda Auto

Nedílnou součástí práce je i zhodnocení, zda by kapacity testovacího týmu aplikace Car Configurator mohly být poskytnuty i jinému oddělení. Problémem zde je, že primárně kapacita tohoto týmu sloučí pro testování funkcionality. V praxi to znamená, že je důležité, aby se zobrazovaly v aplikaci takové informace, které jsou aplikaci poskytnuty. Zda jsou poskytnuté informace správné či nikoli již tým netestuje. Touto problematikou se částečně zabývá rollout tým a poskytnutí části kapacity by znamenalo snížení nákladů, které by byli nuceni vynaložit, pokud by tyto informace chtěli testovat. Bylo by tak nutné najmout další tým z externích firem. Zároveň pokud požadavky na testování nebudou nijak vysoké, myšleno mírou času, nikoli složitostí, není potřeba najímat celý nový tým. Proto se tato část zaobírá myšlenkou využití kapacity jednoho týmu pro dvě odlišné oddělení a dva odlišné koncepty.

5.2.1 Poskytnutí kapacity testovacího týmu

Jak už bylo zmíněno v předešlé kapitole, tak by rollout tým chtěl mít kontrolu nad tím, zda zobrazované informace v aplikaci jsou správné a zda se někde nenachází chyba. Prvotní myšlenkou bylo samozřejmě najmout nějaký testovací tým nebo koupit již hotové řešení pro automatické testování webových aplikací a pak sestavit tým, u kterého by nebylo třeba vysokých nároků na znalost programovacích jazyků a znalost automatizovaných testů. Tímto způsobem by se náklady snížily. Nicméně hlavní rolí jsou zde samozřejmě finance a v obou případech by náklady na jedno nebo druhé řešení byly příliš vysoké.

Jelikož testovací tým pro webovou aplikaci Car Configurator již existuje, bylo by vhodné tedy využít kapacity tohoto týmu. Nyní je tedy otázka jakým způsobem toho lze dosáhnout. Jelikož jako veškerý vývoj aplikací i tento vývoj probíhá v agilním režimu, tak jsou týmu zadávány úkoly v takzvaných PI. Tyto PI jsou pak rozděleny do jednotlivých sprintů, kde jsou vykazovány pokroky v dané problematice. Tímto způsobem tedy lze dosáhnout toho, že by rollout tým měl možnost při každém PI alokovat část kapacity testovacího týmu pro své účely. Výhodou tohoto řešení je samozřejmě i to, že není třeba se ničemu vázat na dobu neurčitou a lze využít opravdu takovou kapacitu, která je aktuálně potřeba. Protože se samozřejmě může naskytnout i situace, kdy nebude potřeba žádný vývoj testů nebo to nebude prioritou.

5.2.2 Zhodnocení a návrh

Před závěrečným zhodnocením je třeba brát v úvahu nejen možnosti, které se zde nabízí, ale i za jakou cenu by byly realizovatelné. Řešení uvedené v předchozí kapitole se zaměřuje na využití interních kapacit, pomocí čehož by se mělo dosáhnout výrazně nižších nákladů, ale stále nenulových. Pro zavedení takovéto spolupráce je tedy potřeba podložit potřebu financí konkrétními fakty. Navrhované řešení je tedy nejdříve započít spolupráci na méně náročném projektu a využít například již hotové automatizované testy s mírnou úpravou pro konkrétní potřeby. Pomocí tohoto testu následně získat data o chybách, které se vyskytují a předat požadavky na příslušná oddělení, které za vývojem projektu stojí. Po opravení těchto chyb již bude dostatek podkladových materiálů, které mohou sloužit právě jako podklady pro odůvodnění potřebných financí pro projekt automatizovaného testování.

Jelikož v případě spolupráce dvou oddělení bude pravděpodobně na testovací tým kladeno více požadavků, bylo by vhodné financovat dalšího testera do již zaběhlého týmu. Část týmu by tedy financoval produktový tým samotné aplikace Car Configurator a část týmu by financoval rollout tým. Tímto by se dosáhlo efektivnějšího způsobu testování jedné aplikace pro více účelů a zároveň by finanční zatížení pro rollout tým nemělo představovat tak vysoké náklady, jako v případě najmutí dalšího testovacího týmu nebo zakoupení hotových řešení z externích firem.

5.3 Zhodnocení využitého řešení pro automatizované testy ve Škoda Auto

Ve firmě Škoda Auto je pro automatizované testy aplikace Car Configurator využito nástroje Robot Framework. Tento nástroj je popsán v kapitole 3.1. Důvodem využití tohoto nástroje je jeho jednoduchost. Psaní automatizovaných testů je poté jednoduše pochopitelné, protože jednotlivé funkce a procedury se volají pomocí klíčových slov. Začátky programování v tomto jazyce jsou tedy pochopitelné pro jakéhokoli člověka znalého v oblasti IT a není tedy zprvu nutná perfektní znalost nějakých programovacích jazyků ani znalost jak s nimi pracovat. To samozřejmě neznamená, že Robot Framework je nástroj pro každého a že s ním lze automatizovat pouze primitivní testy aplikací. Pro Robot Framework existuje totiž nespočet knihoven a to poměrně obsáhlých, lze tedy psát jak primitivní testy, tak i daleko rozsáhlejší testy s vysokými nároky. Proto je tento nástroj naprosto vyhovují pro účely testování aplikace Car Configurator ve firmě Škoda Auto. Jelikož v testovacím týmu může docházet k častým obměnám, tak je poté následná adaptace nových členů do týmu mnohem jednodušší.

6 Car Configurator

6.1 Vize

Vize tohoto produktu, tedy nového konfigurátoru vozů je naplnění slibu značky Škoda, tedy lidskost a snadné používání aplikace. Má za cíl překvapovat zákazníky již v raných fázích zvažování koupě nového vozu a tedy i v procesu rozhodování o nákupu. Cílem je tedy dosáhnout flexibilnější konfiguraci a přizpůsobení požadavků každému zákazníkovi zvlášť. Důležité body ohledně zákaznické a obchodní nabídky jsou uvedeny v následujícím seznamu:

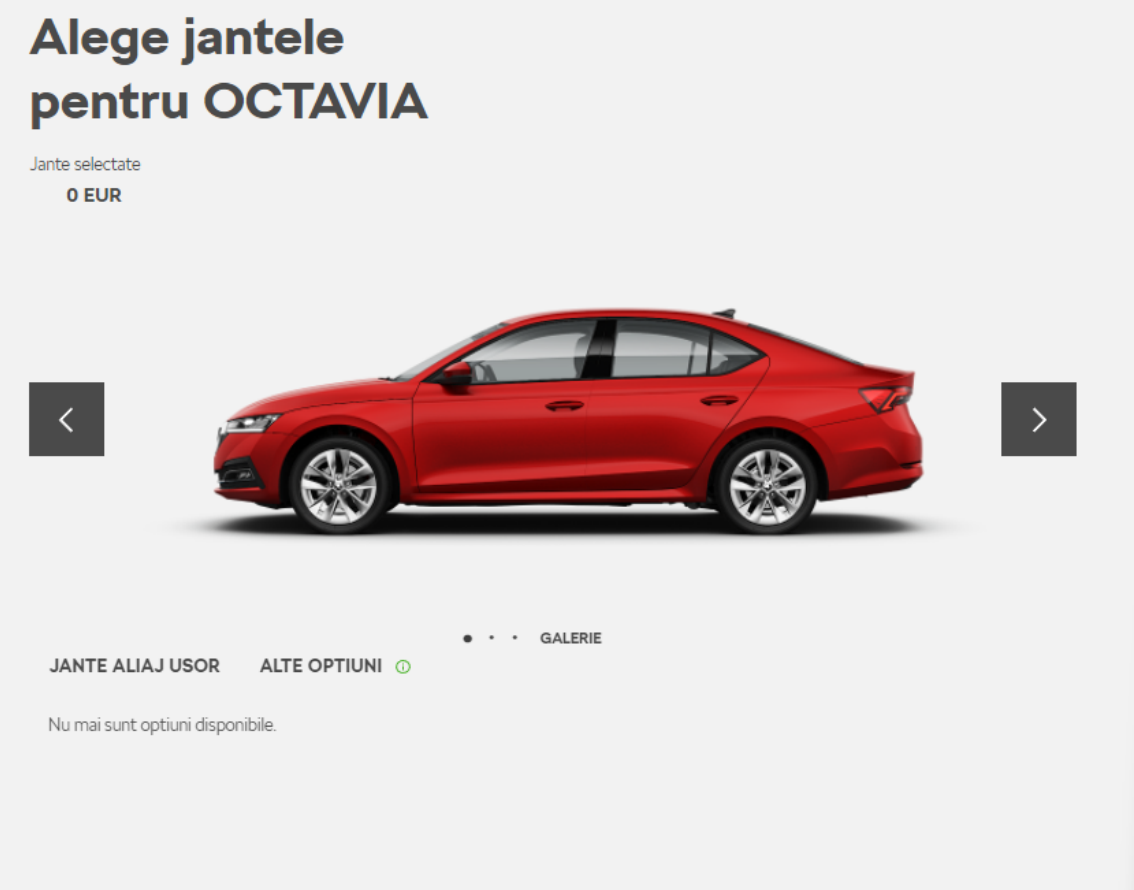
- Zjednodušení konfigurace pomocí inteligentního poradce pro výběr vozu.
- Responzivní design a srozumitelný konfigurátor hodící se ke všem dotykovým zařízením a chování zákazníků.
- Možnost uložit nakonfigurované vozy do osobní garáže pro pozdější úpravy, kontrolu či srovnání.
- Nástroj pro generování cenných potenciálních zákazníků z konfigurátoru pomocí dobře definovaných výzev akcí.
- Nástroj pro získávání dat o zákaznících a jejich zájmech.

6.2 Aktuální problémy

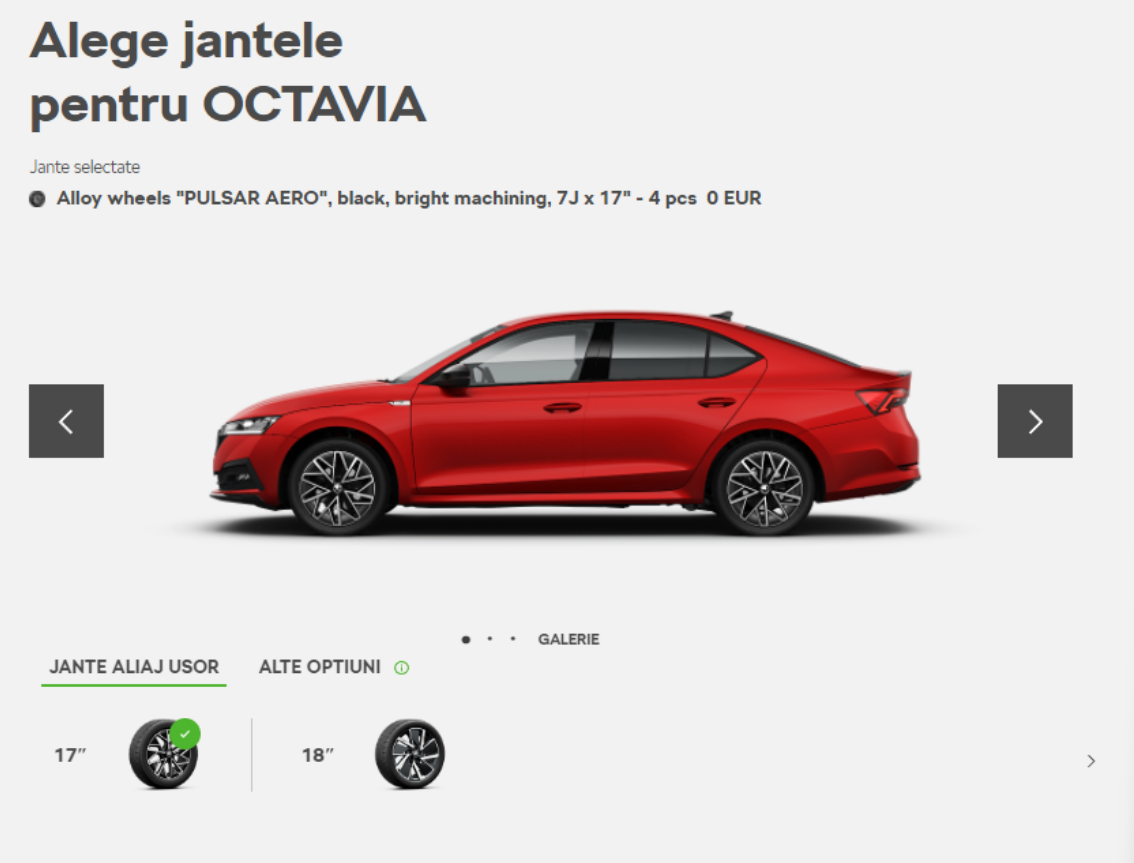
Aby mohla být naplněna vize v předešlé kapitole, předpokládá se, že aplikace bude fungovat bezchybně a že zákazník bude mít opravdu možnost nakonfigurovat si vůz dle svých představ. Bohužel v aplikaci často dochází k chybám, které nemusejí být způsobené samotnou funkcí aplikace, ale lidským faktorem. Pokud se v aplikaci nachází chyba a zákazníkovi není nabídnut některý z produktů může být příčinou nejen špatná funkce aplikace, ale například pouze špatné nastavení. Za každou zemi, ve které je aplikace spuštěna, zodpovídá tzv. importér. Úkolem tohoto importéra je nastavit v backend verzi webu vše, co se má zákazníkovi nabízet. Aktuálně aplikace trpí častým problémem s neviditelností kol (viz obrázek 6.1) a neviditelností hodnot CO_2 (viz obrázek 6.3). Správná funkce aplikace a tedy správné zobrazení konfigurace vozu je ukázáno na obrázku 6.2 (správné zobrazení kol) a na obrázku 6.4 (správné

zobrazení hodnoty CO_2). Pokud si tedy zákazník konfiguruje vůz, je možné, že nedostane na výběr žádná kola a to je rozhodně nežádoucí. Neviditelnost hodnot CO_2 není tak zásadní, protože nemění nic na tom, že si zákazník může vybrat motorizaci dle svých představ, pouze se nedozví jakou emisní hodnotou CO_2 daná konfigurace disponuje.

Aby se zabránilo těmto chybám, je nutné při každé obnově nabídku opravdu pečlivě zadávat a kontrolovat nabízené konfigurace. V každé zemi je ale tolik nabízených možností konfigurací, že sám člověk by proces této kontroly nezvládl v žádném krátkém čase. A opět by bylo možné, že by při této kontrole člověk udělal chybu a nedošlo by tak k úplné kontrole všech možných konfigurací. Proto se zde zavádí automatizované testy, které budou automaticky kontrolovat viditelnost kol pro všechny možné konfigurace ve všech dostupných zemích a které budou kontrolovat i zobrazované hodnoty CO_2 pro všechny možné motorizace.

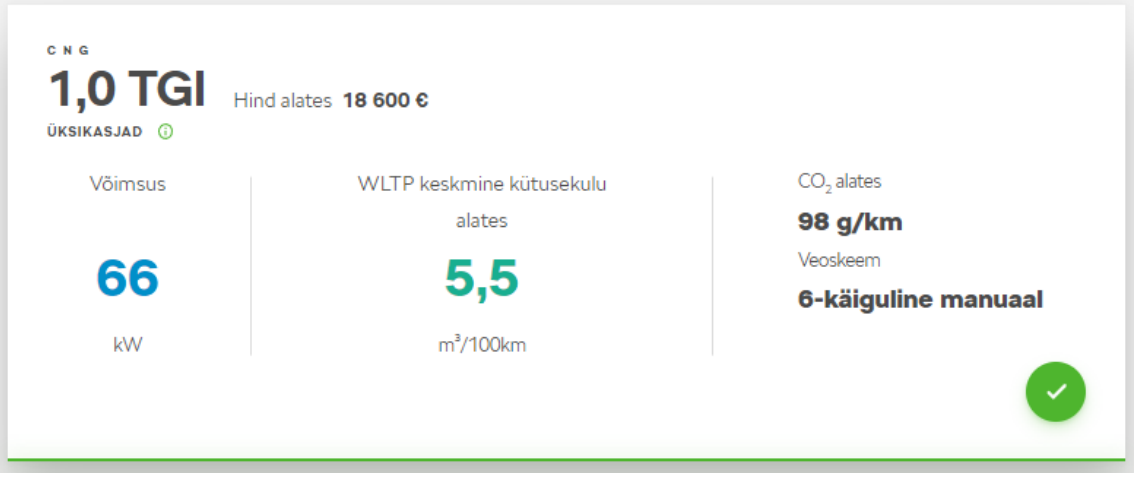


Obrázek 6.1: Konfigurace vozu bez viditelných kol (zdroj: cc.skoda-auto.com)



Obrázek 6.2: Správně zobrazená konfigurace vozu včetně kol (zdroj: cc.skoda-auto.com)

Obrázek 6.3: Chybně zobrazená motorizace vozu bez hodnoty CO_2 (zdroj: cc.skoda-auto.com)



Obrázek 6.4: Správně zobrazená motorizace vozu včetně hodnoty CO_2 (zdroj: cc.skoda-auto.com)

7 Práce s Robot Framework

7.1 Prohlížeč

Jelikož práce pojednává o automatizovaných testech webové aplikace, tak je třeba využívat k tomu nějaký z dostupných webových prohlížečů. Testovací strategie Škoda Auto předpokládá, že testy provedené v prohlížeči Google Chrome jsou dostatečně vypovídající o správné funkci aplikace a není dále nutné testy provádět i v jiných prohlížečích. Pro práci s prohlížečem je nutné disponovat i jeho ovladačem. V případě Google Chrome, který prochází častými obměnami a je vydávána nová verze několikrát za rok, je nutné tuto verzi kontrolovat a disponovat ovladačem pro tuto nejnovější verzi. Google vydává novou verzi prohlížeče vždy včetně jejího ovladače s názvem ChromeDriverXX, kde XX označuje právě pro jakou verzi je ovladač určen. Pro otevření prohlížeče v Robot Framework pak slouží klíčové slovo "Open Browser" s argumentem názvu prohlížeče, například právě "chrome". Obvykle se otevře v prohlížeči jedno minimalizované okno. Nicméně je možné pracovat jak s více prohlížeči najednou, tak i s více okny najednou.

7.2 Lokalizace elementů

Pro práci s webovými aplikacemi je nejpodstatnější porozumět lokalizaci elementů, se kterou se během automatizovaného testování lze setkat nejčetněji. Pro identifikaci konkrétního elementu se na webové stránce používá lokátor. Tento lokátor bývá unikátní pro jednotlivé elementy a obvykle bývá argumentem některé z nabízených funkcí. Příkladem může být kliknutí na tlačítko. V tomto případě není nutné přímo definovat, kde se tlačítko nachází, ale existuje mnoho dalších způsobů, jak ho správně lokalizovat. Pro práci s webovými elementy slouží samotná knihovna SeleniumLibrary, a lokalizaci lze provádět pomocí ID prvku, výrazy XPath či selektory CSS. Strategie lokalizace může být explicitně specifikována určitým prefixem, či může být strategie implicitní. Ve výchozím nastavení se předpokládá, že lokátory využívají výchozí strategii lokátoru. Veškerá klíčová slova této knihovny podporují hledání prvků na základě ID a NAME atributů, ale některá další klíčová slova podporují vyhledávání i dle dalších atributů či hodnot.

Lokalizačních strategií tedy existuje mnoho a otázkou je jakou využít pro automatizované testování webové aplikace Car Configurator. Jelikož samotná aplikace probíhá agilním vývojem neustále a cílem Škoda Auto je neustále zlepšovat její funkce, další funkce přidávat a dělat aplikaci uživatelsky přívětivější, tak dochází k časté

změně. Proto zde není vhodné elementy lokalizovat pomocí jejich polohy na stránce, například pomocí XPath. Tento argument by poté bylo nutné měnit s každou změnou a vyžadovalo by to neustálou kontrolu a tedy i zvýšení nároků na udržování aktuálnosti testu. Vhodným řešením je tedy argument nezávislý na poloze elementu s dostatečně jedinečným popisem. Proto se využívá TID, tzv. Test Identification. Každý z elementů na stránce má toto unikátní ID s tím, že pomocí něho lze tento element vždy jednoduše a naprosto přesně a spolehlivě lokalizovat. Nevýhodou tohoto řešení je vývoj aplikace, který je o to komplikovanější. Vývojáři musí při definování elementů vždy vymýšlet nové unikátní ID, které novým elementům přiřadí. Nicméně tuto operaci lze provádět i zpětně, takže pokud tester narazí pří testování na element, který tímto ID nedisponuje, tak se vývoji předá požadavek na jeho doplnění.

7.3 Klíčová slova

Další důležitou části je samotná práce s klíčovými slovy, které při automatizovaných testech opravdu klíčovou roli hrají, jelikož veškeré funkce jsou volány pomocí nich. U nástroje Robot Framework jsou definované základní funkce v již zmíněných knihovnách, ale samozřejmě v sekci "Keywords" lze definovat i vlastní funkce. Tyto funkce mohou mít jak argumenty, tak návratovou hodnotu. Argumenty se uvádí hned za definicí názvu funkce a návratovou hodnotu pak lze nastavit pomocí příkazu "return". Mohou i nastat situace, že argumentem jedné funkce je funkce další a to hlavně v případě, když chceme vědět zda funkce byla řádně vykonána. Klíčové slovo "Run keyword and return status" pak očekává jako argument právě další funkci, například "Click element". Pokud tedy bude vykonáno kliknutí na požadovaný element úspěšně, pak nám bude navrácena hodnota "True" a v opačném případě bude navrácena hodnota "False". Pro správnou syntaxi klíčového slova a jeho argumentů je nutné klíčové slovo od argumentů a i argumenty od sebe oddělit tabulátorem či dvojitou mezerou. Poslední důležitou zmínkou jsou klíčová slova "Test Setup" a "Test Teardown". Tyto klíčová slova se využívají pro definování funkcí, které mají být spuštěné před samotným testem a v případě selhání. Takže před spuštěním to může být nějaké prvotní nastavení parametrů popřípadě spuštění prohlížeče a podobně. V případě selhání testu to pak může být uložení některých informací, uzavření všech spuštěných prohlížečů či pořízení snímku obrazovky pro následnou identifikaci příčiny selhání.

8 Řešení

8.1 Jednotlivé části skriptu

Samotný skript se skládá ze čtyr základních částí. První části je nastavení s označením `*** Settings ***`. Tato část slouží pro nastavení použitých knihoven a zdrojů a popřípadě i nastavení funkcí před spuštěním, po dokončení nebo případně při selhání testů. Další částí jsou proměnné s označením `*** Variables ***`. V této části jsou důležité globální proměnné, které chce mít uživatel při ruce, slouží tedy například k nastavení režimu testování. Další části jsou klíčová slova, respektive nadefinované funkce, kde každou lze identifikovat jedním z klíčových slov, tato část nese označení `*** Keywords ***`. Část s označením `*** Test Cases ***` je poslední nedílnou částí a obsahuje daný testovací případ.

8.1.1 Nastavení

V této části jsou nahrány nezbytně nutné knihovny pro práci s webovým prohlížečem a pro práci se sešitem excel ze kterého budou nahrávány jednotlivé URL adresy testovaných aplikací a do kterého budou po skončení testu uloženy i nalezené problémy. Jsou zde i nastaveny cesty pro zdroj externích klíčových slov a pro zdroj externích proměnných. Poslední částí nastavení je i uvedení dvou funkcí, které se spustí před spuštěním testu a při selhání testu.

```
1  *** Settings ***
2  Documentation  Checks the visibility of wheels. Captures screenshots
       and returns urls in excel sheets.
3
4  Library  SeleniumLibrary
5  Library  ExcelLibrary
6
7  Resource  Resources/Vars.robot
8  Resource  Resources/CommonKeywords.robot
9
10 Test Setup  run keyword  start tests
11 Test Teardown  run keyword  close tests
```

Zdrojový kód 8.1: Nastavení

8.1.2 Proměnné

V části skriptu s názvem Variables jsou uvedeny důležité proměnné pro samotný průběh testu. Lze zvolit který prohlížeč má být pro test využit, dále lze zvolit na kterém modelu vozu a na které výbavě vozu začít testování. A posledním parametrem je, zda se mají testovat i všechny varianty motorizací nebo zda se mají testovat všechny konfigurace pouze s automaticky přednastavenou motorizací.

```
1 *** Variables ***
2 ${browser}  chrome
3 ${startHPCard}  0  #If you want to check all cars chose 0
4 ${startTrimline}  0  #If you want to check all trimlines chose 0
5 ${CheckRestEngines}  ${True}  #If you want to check all engines chose
    True (chose False for check wheels with only pre-chose one engine)
```

Zdrojový kód 8.2: Proměnné

8.1.3 Test Cases

V této sekci je uvedeno pouze jedno klíčové slovo, tím je "Check Wheels Visibility". Je to kvůli tomu, že tento test prozatím slouží pouze k jednomu účelu, ale lze vytvořit více těchto testovacích případů s využitím stejných funkcí, tyto případy pak lze spouštět v takzvané "pipeline", kde jsou testovány tyto případy postupně a je z každého generován výstup úspěšnosti. V tomto konkrétním případě test nejprve načte všechny URL, které chceme testovat, postupně vybere jednu po druhé a projde následující úkony:

- nastavení testované URL
- nastavení zkratky testované země
- nastavení cesty pro ukládání snímku obrazovky dle zkratky země
- otevření URL ve webovém prohlížeči
- zavření veškerých pop-up oken v prohlížeči
- zahájení testu viditelnosti kol s návratovým parametrem "foundurls"
- vytvoření sešitu excel a nahrání všech nalezených chyb obsažených v návratovém parametru "foundurls"
- načtení nové URL a opakování všech kroků znovu do vyčerpání všech adres

```
1 *** Test Cases ***
2 Check Wheels Visibility
3     [Tags]  Wheels
4     ${cnt}  Get FE url for test  PROD  PROD  ${wheels_check_col}
5     FOR  ${i}  IN RANGE  ${cnt}
6         ${testurl}  set variable  ${URL}[${i}]
7         ${testcountry}  set variable  ${country_code}[${i}]
```

```
8         set screenshot directory  ${testcountry}
9         go to  ${testurl}
10        set global variable  ${selectedcountrytext}  ${testcountry}
11        Log to console  ${selectedcountrytext}
12        Close pop-up  ${CookieAccept}
13        Close pop-up  ${ContinueButton}
14        Close pop-up  ${CloseRecCar}
15        ${foundurls}  Check Wheels for each tile  ${testurl}
16        Create excel for values  'wheels'  ${testcountry}  ${foundurls}
17    END
```

Zdrojový kód 8.3: Test Cases

8.1.4 Keywords

Volba URL

Počáteční fází pro start testu je samozřejmě spuštění prohlížeče, kterým je zde Google Chrome. Pro jeho správné spuštění a následně správnou funkci je potřeba i jeho ovladač. Samotný ovladač pro každou verzi prohlížeče je vydáván spolu s každou novou verzí. Pro vyhnutí se nevynucených chyb je určitě vhodné využívat stable verze a vyhnout se tak beta verzím, které by mohly vnést do samotných testů právě zmiňované nevynucené chyby. Po spuštění prohlížeče následuje jeho maximalizace, dalším krokem je načtení všech URL, které chceme testovat. Tyto URL jsou uživatelsky definované v excelovém souboru. Jakmile jsou URL načteny a uloženy do pole, tak se z pole vezme první adresa a otevře se v prohlížeči, obdobně i pro ostatní země v následujících průchodech testu. Po otevření adresy a tedy otevření webové aplikace Car Configurátor je nutné zavřít takzvaná pop-up okna, které informují uživatele. Jedním z těchto oken je i informace ohledně zpracovaní cookies souborů, které lze přijmout nebo odmítnout.

Volba HpCard

Ve chvíli kdy se prohlížeč nachází na stránce, kde už lze uživatelsky vybírat vůz pro konfiguraci a jsou zavřena všechna okna, která by v testech překážela, tak nastává detekce. Každý vůz je v aplikaci pro testovací účely označen unikátním identifikátorem ”TID”, neboli ”Test ID”. Využitím klíčového slova ”Get Tids” a parametru s názvem unikátního identifikátoru můžeme naleznout všechny vozy, které se na stránce vyskytují. Pro vysvětlení, v tomto konkrétním případě se každý vůz identifikuje identifikátorem TID jako TID = ”HPCardCarLineXXX”, kde XXX je číselné označení, které může být jednociferné i více ciferné. Proto jako parametr funkce předáme, že hledané identifikátory začínají na ”HPCardCarLine”. Funkce následně vyhledá všechny elementy na aktuální webové stránce a vrátí jejich přesné identifikátory TID jako pole.

Dalším krokem je samozřejmě výběr prvního prvku pole, tedy prvního identifikátoru elementu vozu a následné kliknutí na tento element. Funkce kliknutí je i opatřena zpětnou vazbou, aby bylo docíleno toho, že ke kliknutí došlo správně a že

se test nachází na správné stránce. V tomto případě se čeká, až bude k dispozici element webové stránky, který označuje tlačítko pro výběr jednotlivých modelů vozu. Pokud se toto tlačítko na stránce nachází, došlo ke správné funkci testu a nachází se právě na výběru modelu vozu.

Výběr Trimline

Dalším krokem v průběhu automatizovaného testu je právě výběr modelu vozu. Obdobně jako v případě výběru vozu, i zde je každý model detekovatelný pomocí specifického identifikátoru, ten je v tomto případě TID=”TrmlnCardXX”, kde XX je opět číslo odlišné pro všechny varianty. K detekci opět slouží funkce ”Get Tids”, tentokráte s tímto jiným identifikátorem. Po získání pole se všemi identifikátory modelů vozu dochází ke kliknutí na element s identifikátorem uloženým v poli jako první. I v tomto případě je zde zpětná vazba pro ověření zda došlo ke správné funkci kliknutí. Zároveň se kontroluje zda byl opravdu vybrán správný model. Tato zpětná vazba detekuje zda po kliknutí na daný element dojde k jeho zmizení, čeká tedy až se tento element na stránce nebude nacházet. Poté lze přejít na další krok a tím je výběr motorizace.

Výběr motorizace

V rámci kontroly všech možných konfigurací je nutné provést test i pro všechny motorizace daného modelu vozu. Vyvinutý skript, ale počítá i s variantou, že nebude potřeba testovat všechny motorizace a v sekci ”Variables” je parametr určující, zda se všechny motorizace mají či nemají testovat. hodnotu tohoto parametru lze měnit na True a False. V případě, že je uvedená hodnota False, tak je tento krok přeskočen a dochází k testovaní viditelnosti kol pouze pro všechny modely daného vozu, ale nikoli pro všechny dostupné motorizace. Dochází tedy k testu pouze s automaticky předdefinovanou motorizací. Ale v případě, že je uvedena hodnota parametru jako True, tak dochází ke kliknutí na element pro skok na webovou stránku, kde se nacházejí všechny možné motorizace, které pro danou konfiguraci lze zvolit. Opět i zde je zpětná vazba v podobě kontroly, zda se opravdu nacházíme na stránce s výběrem motorizace. Tato kontrola je poměrně jednoduchá a dochází pouze k dotazu, zda se na stránce nachází alespoň jeden element s identifikátorem motoru. V případě že ano, tak dochází ke stejnému procesu jako u vozů i jejich modelů, tedy k detekci všech motorů podle jejich identifikátoru.

Jakmile jsou všechny motory detekovány, tak dochází k jejich postupnému výběru, tedy kliknutí na element. zpětnou vazbou je znovu čekání na to, až se element, na který bylo kliknuto, nebude na stránce nacházet. Tím způsobem je signalizováno, že tlačítko pro výběr motoru není k dispozici a motor je tedy vybrán korektně. Jakmile dojde i k tomuto korektnímu výběru, tak nastává ta nejdůležitější část, kde je již kompletně vybrána konfigurace a je potřeba zjistit zda jsou pro tuto konfiguraci nacházejí na výběru kol dostupné alespoň jedny kola.

Kontrola kol

Samotná kontrola kol je závěrečnou funkcí celého procesu testu. Zde dochází na kliknutí na tlačítko pro přesměrovaní na stránku s výběrem kol. Jakmile se test nachází na této požadované stránce, kde lze vybrat kola pro vybranou konfiguraci, dochází k detekci elementu kol. Zjišťuje se tedy, jestli se vůbec nějaké kola na stránce nachází. Stejně jako každý vůz, každý model i každá motorizace mají svůj identifikační kód, tak i každé kolo lze detekovat pomocí tohoto identifikátoru, dotaz testu je tedy, zda se na stránce nachází jakýkoli element s tímto identifikátorem. V případě že se na stránce nenachází žádný element s tímto identifikátorem, tak je potřeba tuto skutečnost zaznamenat. Pro zaznamenání je potřeba ze stránky vyčíst o jaký vůz se jedná, o jaký model se jedná a o jakou motorizaci se jedná. Tyto informace jsou poté uloženy do pole, do kterého se ukládají pouze informace o konfiguracích, které nemají viditelné ani jedno kolo. Aby bylo možné ověřit uživatelsky, zda test opravdu našel chybu a opravdu se žádné kolo na stránce nenachází, tak je proveden i snímek obrazovky dané stránky, na které je chyba viditelnosti kol nalezena.

Po detekci kol první konfigurace dochází k návratu na výběr motorizace, pro každou motorizaci se opět provede stejný test viditelnosti kol a po vyčerpání všech možností motorizací se test vrátí na výběr modelu. Tento proces se opět celý zopakuje pro všechny z dostupných modelů a pro dokončení viditelnosti kol pro všechny konfigurace daného vozu se test vrátí na domovskou stránku, kde dojde k výběru dalšího vozu v pořadí a opět se celý cyklus opakuje, dokud nejsou vyčerpány i všechny možnosti vozů i jejich jednotlivých modelů. Úplným závěrem je vytvoření excelového souboru, do kterého se zapíší všechny nalezené konfigurace s problémem viditelnosti kol. Tento excel se vytváří pro každou testovanou zemi, respektive URL dané země zvlášť. Pokud je tedy vybráno několik zemí k otestování viditelnosti kol, tak vznikne stejný počet výstupních souborů.

Kontrola výběru konfigurací

Během celého procesu testování viditelnosti kol jsou ukládány i informace ohledně toho, jaká konfigurace je aktuálně zvolena. V případě, že některá ze zpětných vazeb zahlásí problém, tak test do konzole daný problém vypíše. Například pokud z nějakého důvodu nelze vybrat některou z motorizací, tak test do konzole vypíše, že nebylo možné vybrat některou z motorizací, včetně informace u kterého vozu k tomu došlo. Takto se docílí i odhalení nesrovnalostí jednotlivých konfigurací, které se nastavují v backend verzi webové aplikace. Mimo jiné je při každém testu viditelnosti kol inkrementována hodnota vypovídající o počtu otestovaných konfigurací, která je následně součástí výstupního souboru a lze tak ověřit, zda byly otestovány všechny konfigurace. Pomocí této hodnoty lze vyčíslit i procentuální chybovost.

Kontrola CO_2

Automatizovaná kontrola zobrazovaných hodnot CO_2 je v samostatném testovacím skriptu, nicméně princip funkčnosti je velmi podobný kontrole viditelnosti kol. Nejprve je vhodné vysvětlit, kde se hodnoty CO_2 v aplikaci objevují. Tyto CO_2

hodnoty jsou parametrem každé motorizace a úzce souvisí s hodnotou průměrné spotřeby dané motorizace. Ve většině případů tedy tyto hodnoty korelují a pokud není zobrazena korektně hodnota spotřeby, tak není zobrazena korektně ani hodnota CO_2. Emisní hodnota CO_2 je tedy zobrazena u každé motorizace a dále se propisuje do vyšších vrstev. Jelikož každá výbava daného vozu obsahuje v převážné většině několik možností motorizace, tak je u každé výbavy i informace o tom, jaké nejnižší emisní hodnoty může dosahovat při výběru jednoho z nabízených motorů, v podstatě toho nejšetrnějšího k životnímu prostředí, jelikož se tato emisní hodnota udává v gramech CO_2 na jeden ujetý kilometr. Každý vůz má ale většinou i několik různých výbav, obdobně to tedy platí i zde. U každého vozu (na domovské stránce aplikace) je také zobrazena nejnižší emisní hodnota, zde je uváděna tedy ta hodnota, která je nejnižší skrze všechny možné výbavy daného vozu. V případě, že je ale některá z motorizací špatně nastavena a nezobrazuje se u ní korektní hodnota CO_2, v tomto případě myšleno, že se zobrazí N/A či jiné obdobné označení pro chybějící hodnotu, tak se tento údaj propíše až do té nejvyšší úrovně. Poté lze tyto chybné informace pozorovat jak na úrovni vozu, tak na úrovni výbavy a i na úrovni motorizace. Z těchto úvah vyplývá, že je tedy potřebné hledat chybně zobrazenou hodnotu CO_2 až na úrovni motorizace.

Tento test tedy provede stejné kroky jako kontrola viditelnosti kol, tedy výběr vozu, výběr výbavy a následně otevření dostupných motorizací pro danou výbavu. V této části pak hledá všechny zobrazené informace o motoru, jako je výkon, převodovka, objem a samozřejmě i emise. Tyto informace jsou poté uloženy do seznamu. Po kontrole všech motorů ve všech vozech a ve všech možných výbavách se tento seznam ukládá do excelového souboru jako tabulka hodnot. Tento soubor je pak nutné kontrolovat. Test nevypisuje pouze chybně zobrazené hodnoty CO_2 ve smyslu, že nejsou doplněny, protože pak lze kontrolovat, zda číselné hodnoty těchto motorizací opravdu souhlasí se skutečnými. Tak lze odhalit jak chyby viditelnosti hodnot, tak chyby zobrazených hodnot. V dnešní době se emise CO_2 u vozů se spalovacím motorem pohybují okolo hodnoty 100g/km. Je tedy nesmyslné, aby některý ze spalovacích motorů dosahoval hodnot malých desítek, případně dokonce jednotek g/km a tomu se chce samozřejmě předejít. Obecně je nežádoucí, aby jakákoli hodnota motoru byla nezobrazena, ať už výkon, spotřeba, objem či zmíněná hodnota CO_2, proto se ukládají všechny tyto hodnoty a lze je zpětně kontrolovat.

Posledním důležitým bodem jsou kódy motorů. I přes širokou nabídku motorů není pro každou konfiguraci motor jedinečný. Existuje tedy určitá sada motorů, které jsou poté v aplikaci přiděleny jednotlivým konfiguracím. Proto je naprosto zřejmé, že i přesto, že zvolíme jiný vůz, či jinou výbavu, tak motorizace bude naprosto stejná. Pokud tedy budou nějaké hodnoty u motorizace chybět budou chybět u každé konfigurace vozu, která bude disponovat touto danou motorizací. Proto v rámci testů je žádoucí kontrolovat specifický kód motoru, který je jedinečný pro danou motorizaci. Je tedy třeba ukládat kódy již otestovaných motorů, aby při kontrole jiné konfigurace se stejným motorem již nebyl duplicitní záznam tohoto motoru. Je to jak z důvodu výpočetního, kde dosáhneme rychlejších průběhů testů, tak i z hlediska následující zpětné kontroly. Jelikož při vynechání stejných motorů klesne celkový počet záznamů a lze tak jednodušeji v listu hledat chybné hodnoty.

9 Výsledky prvních testů

9.1 Viditelnost kol

Jak již bylo zmíněno v předešlých kapitolách, tak výsledky automatizovaných testů na viditelnost kol jsou zapisovány do výstupního excelového souboru pro každou zemi zvlášť. Každý řádek představuje jednu detekovanou chybu. Dále soubor obsahuje informace o jakou konfiguraci se jedná, tedy o jaký vůz, model vozu a motorizaci. Pro přehlednost a lepší demonstrovatelnost výsledků jsou tyto výsledky zobrazeny do tabulek a grafů. Obecné informace o provedených testech jsou zde:

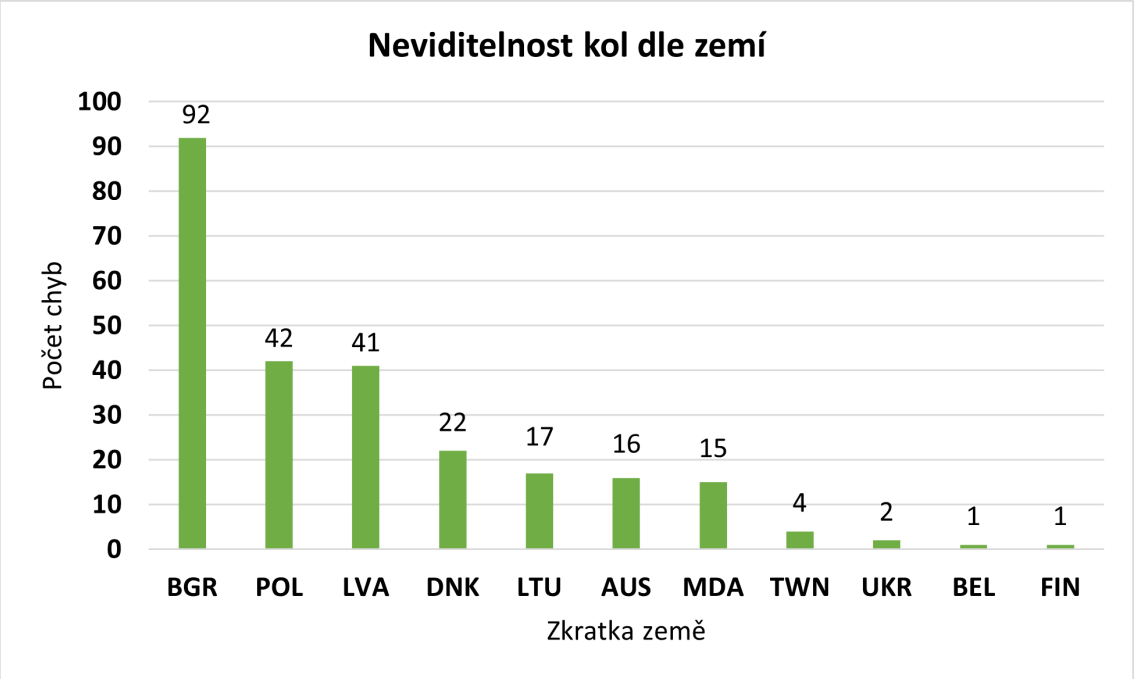
- Počet testovaných zemí: 28
- Počet testovaných konfigurací: více než 1900
- Počet nalezených chyb ve viditelnosti kol: 253 (13 %, téměř každá 8. konfigurace)
- Počet zemí, ve kterých byl nalezen alespoň jeden problém: 11 z 28

Hlavní statistiky chybovosti v jednotlivých zemích jsou uvedeny v tabulce 9.1. Pro přehlednost jsou v tabulce uvedeny pouze ty země, ve kterých byla nalezena alespoň jedna chyba. Zbylé země, které v tabulce nejsou uvedeny jsou země ve kterých nebyla detekována žádná chyba. Z výsledků vyplývá, že nejhorší zemí, myšleno země s nejvyšším výskytem chybovosti viditelnosti kol je Bulharsko, kde celkem 92 různých konfigurací nenabízí zákazníkovu volbu jakýchkoli kol. Z celkových 28 zemí, byla chyba nalezena v 11 a to znamená, že chybovost je ve 39 % případů, ačkoli v některých zemích se chyby vyskytovaly pouze v jednotkách případů.

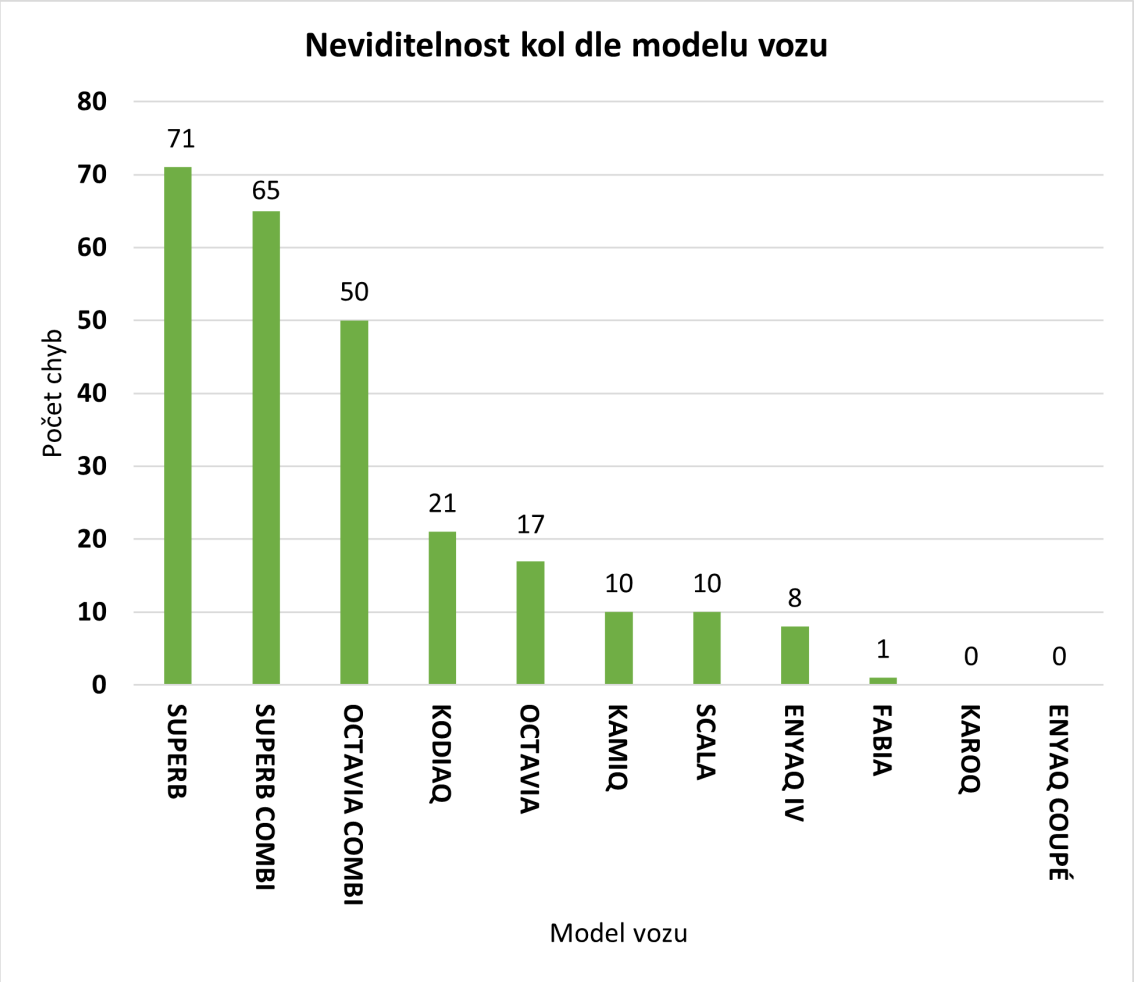
Jelikož jsou zaznamenávány veškeré údaje o konfiguraci u které byla detekována chyba viditelnosti kol, lze vést statistiku i napříč těmito parametry. Proto jsou v této kapitole zobrazeny grafy, které ukazují právě na chybovost v závislosti na těchto parametrech, jako například motorizace. Graf 9.1 znázorňuje chybovost v závislosti na testované zemi. Graf 9.2 znázorňuje chybovost v závislosti na vybraném modelu, kde lze vidět, že nejproblémovějšími modely jsou Superb, Superb Combi a Octavia Combi. Dále je graf 9.3 znázorňující chybovost v závislosti na vybrané převodovce, kde je vidno, že problémovější je výrazně více automatická převodovka. Poslední dva grafy ukazují chybovost v závislosti na motorizaci, kde graf 9.4 zobrazuje závislost na typu motorizace, kde se ukazuje být nejproblémovější motorizace typu TSI. A graf 9.5, který zobrazuje závislost chybovosti na objemu motorizace, kde je nejvíce zastoupený objem 2,0 litru.

Tabulka 9.1: Výsledky prvních testů viditelnosti kol

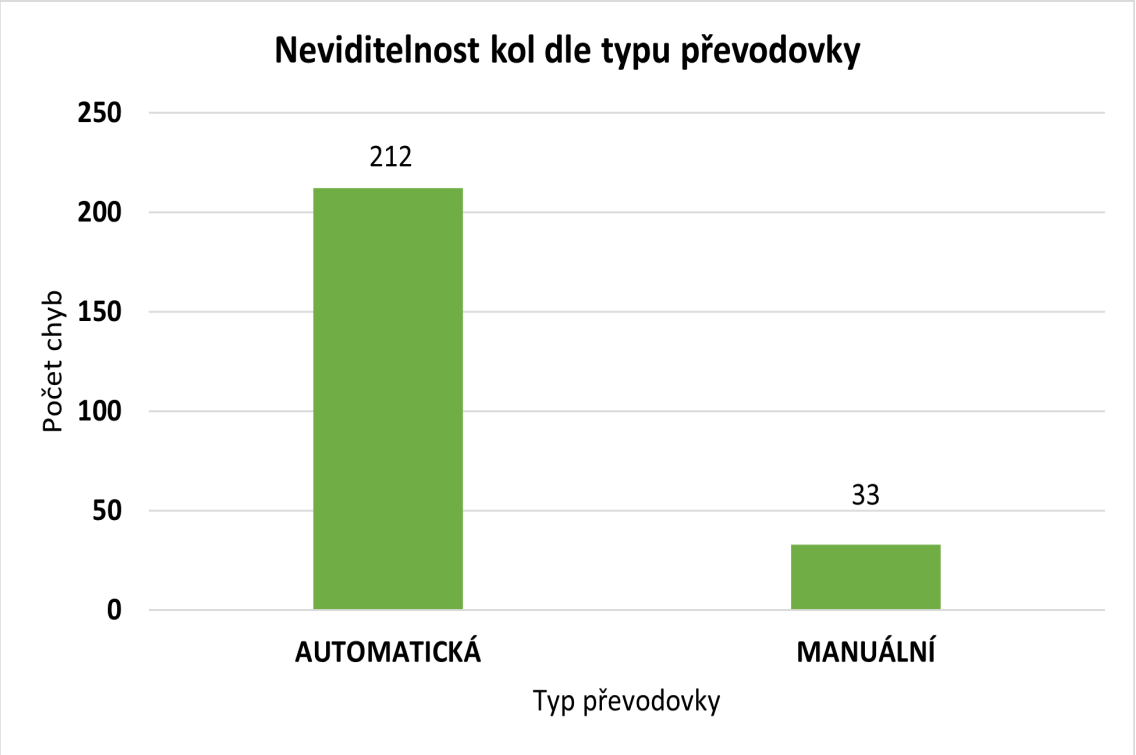
Zkratka	Název země	Počet nalezených problémů
BGR	Bulharsko	92
LVA	Litva	41
DNK	Dánsko	22
POL	Posko	22
LTU	Lotyšsko	17
AUS	Austrálie	16
MDA	Moldavsko	15
TWN	Taiwan	4
UKR	Ukrajina	2
BEL	Belgie	1
FIN	Finsko	1



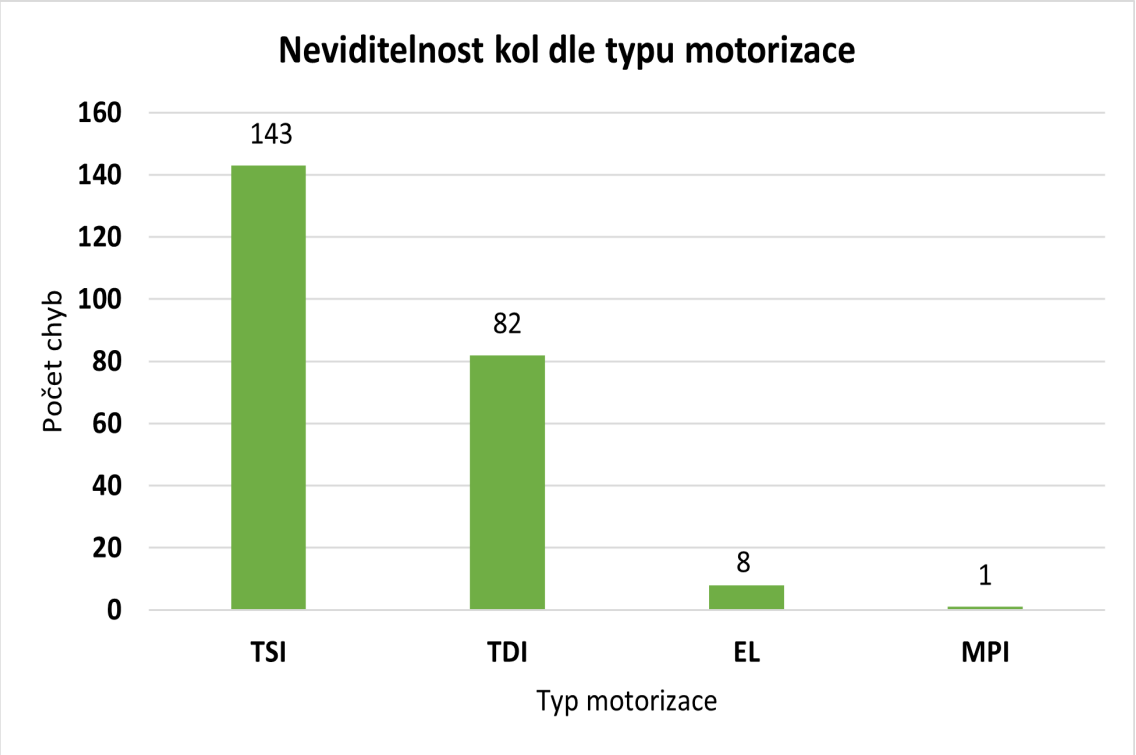
Obrázek 9.1: Graf počtu chyb v jednotlivých zemích



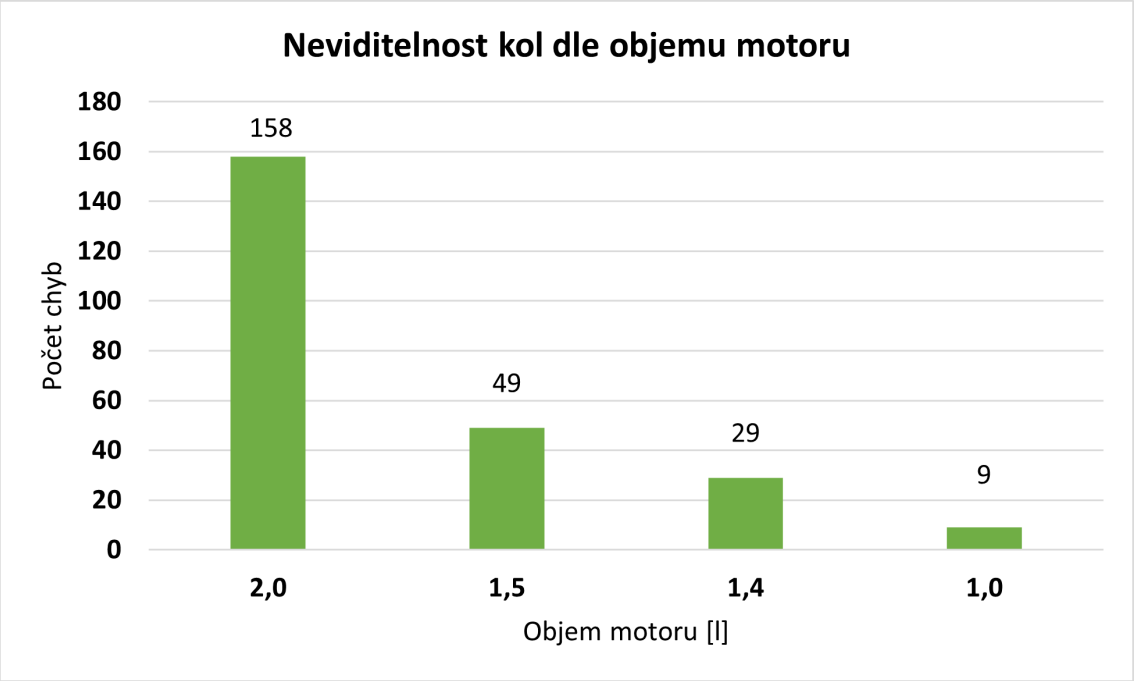
Obrázek 9.2: Graf počtu chyb v závislosti na modelu vozu



Obrázek 9.3: Graf počtu chyb v závislosti na typu převodovky



Obrázek 9.4: Graf počtu chyb v závislosti na motorizaci



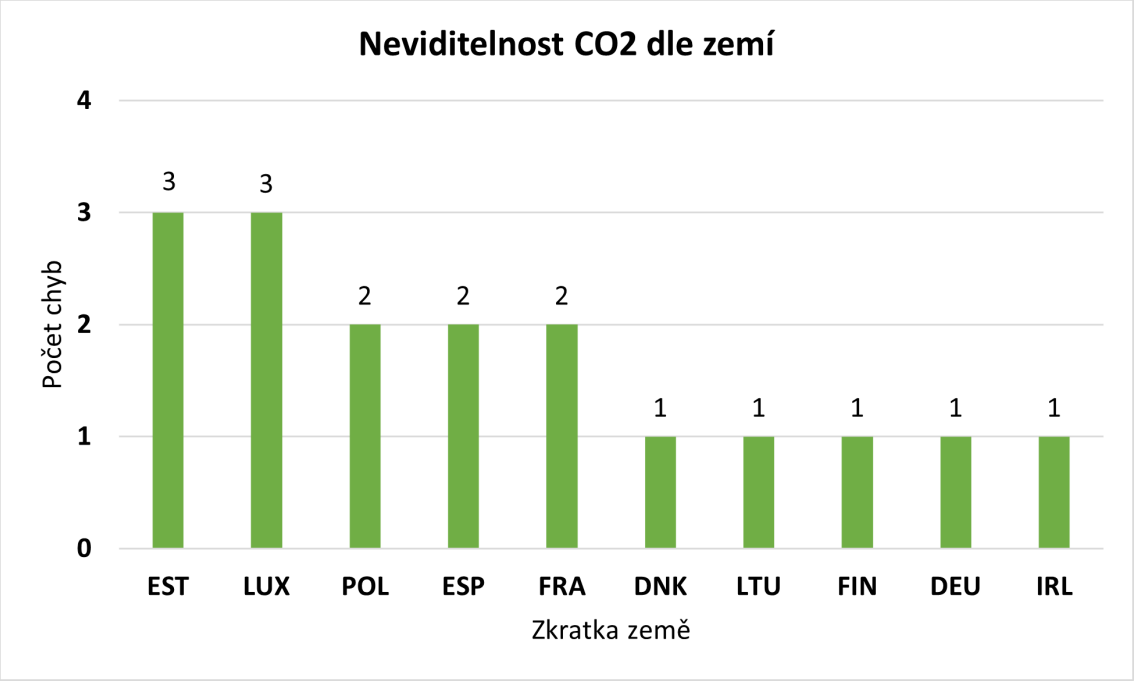
Obrázek 9.5: Graf počtu chyb v závislosti na objemu motoru

9.2 Viditelnost CO_2

Druhou sadou testů jsou testy viditelnosti hodnoty CO_2. Výsledky těchto testů jsou zapisovány do excelového souboru, obdobně jako testy kol. Nicméně tentokrát výsledek nezávisí na konkrétní konfiguraci, ale pouze na motorizaci. Přestože jsou nabízeny zákazníkovi desítky modelů vozů, tak často motorizace pro různé modely jsou naprosto totožné a jelikož tato motorizace nese právě onu hodnotu CO_2, stačí kontrolovat pouze tyto zmíněné motorizace. Výsledek tedy nese informace, o jakou motorizaci se jedná, ale i informaci u kterého vozu a modelu jí lze nalézt, pro případnou následnou kontrolu. Výsledky prvních testů jsou opět zobrazeny v tabulce, konkrétně v tabulce 9.2. Obdobně jako u výsledků viditelnosti kol, i v této tabulce jsou uvedeny pouze ty země, ve kterých byla nalezena alespoň jedna chyba v zobrazení hodnoty CO_2. Z výsledků je pozorovatelné, že počet chyb viditelnosti hodnot CO_2 je výrazně nižší než počet chyb viditelnosti kol. V testu všech 28 zemí a všech možných motorizací bylo nalezeno celkově pouze 17 chyb v 10 různých zemích. Pro grafické znázornění jsou výsledky zobrazeny i pomocí grafu 9.6.

Tabulka 9.2: Výsledky prvních testů viditelnosti CO_2

Zkratka	Název země	Počet nalezených problémů
EST	Estonsko	3
LUX	Lucembursko	3
ESP	Španělsko	2
FRA	Francie	2
POL	Posko	2
DEU	Německo	1
DNK	Dánsko	1
FIN	Finsko	1
IRL	Irsko	1
LTU	Lotyšsko	1



Obrázek 9.6: Graf neviditelosti hodnot CO_2 v jednotlivých zemích

9.3 Předání výsledků na importéry

Jelikož aplikace je spravována v každé zemi zvlášť, tak za každou zemi odpovídá takzvaný importér. Tím je myšlena osoba, která zodpovídá za nastavené informace v aplikaci. Tyto informace se nastavují v backend verzi webu a pokud dojde k nějaké chybě, ať už procesní nebo se informace zadá špatně či se úplně zapomene něco vyplnit, tak dochází k chybám zobrazení na frontend webu. Tento nedostatek je ale velkým problémem, protože vznikají potíže zákazníkovi, který si pak nemůže zvolit konfiguraci dle svých představ. Proto je nutné vzít výsledky testů pro jednotlivé země a předat je všem těmto importérům u kterých je nalezena chyba.

U tohoto předání informací se nabízí, aby informace automaticky odešly bez nutnosti dalšího zásahu. Nicméně po každém testu je nutné výsledky zkontrolovat, jestli opravdu nalezené chyby jsou detekovány správně. Proto se výsledky neposílají ihned automaticky, ale je potřeba, aby je testovací tým prověřil. Pro tento konkrétní případ slouží tzv. screenshoty, jedná se o snímky obrazovky, které jsou pořízeny vždy když je nalezena chyba zobrazení. Testovací tým pak jednoduše může ověřit zda detekovaná chyba zobrazení je opravdu chybou v aplikaci a nebyla detekována chybně. Po ověření těchto chyb se jednotlivé reporty z testu rozešlou na importéry, kteří dostávají za úkol detekované chyb opravit a dosahuje se tak lepší funkcionality aplikace.

10 Výsledky druhých testů

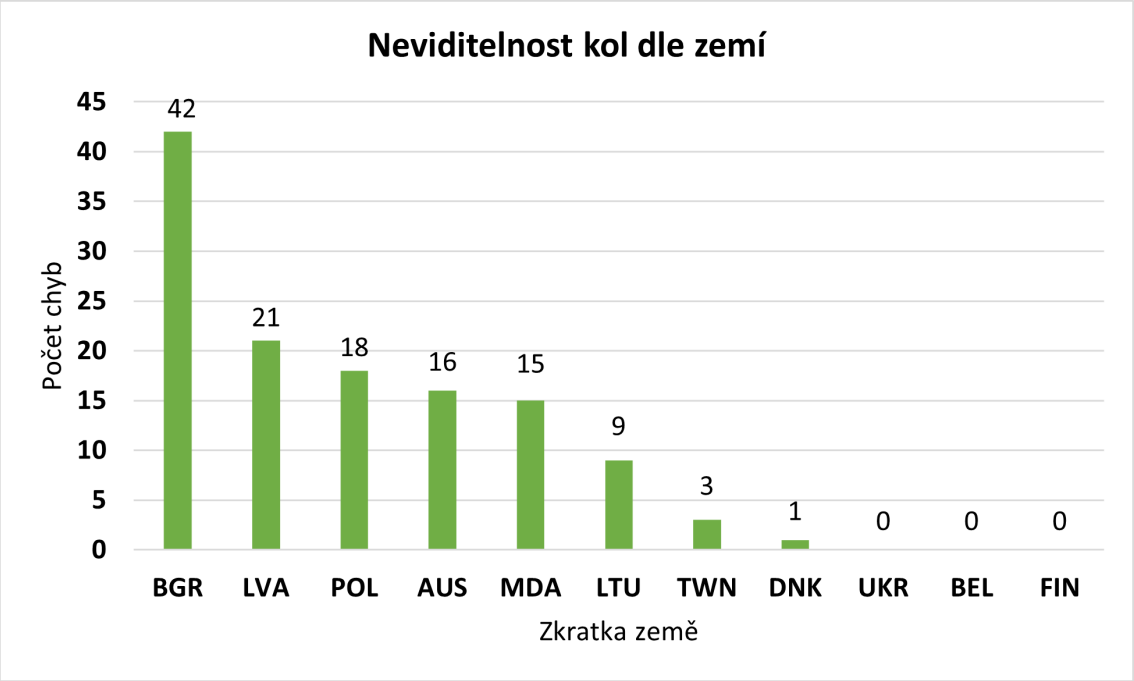
V rámci toho, že výsledky prvních testů byly předány na importéry ve smyslu požadavku na opravení těchto chyb, tak se očekává toto opravení v rámci několika sprintů. Pro demonstraci funkce předávání informací mezi různými odděleními ve firmě Škoda Auto byly tedy otestovány všechny země znovu. Očekávaný výsledek byl, že budou veškeré chyby opraveny, nebo alespoň převážná většina.

Výsledky nalezených chyb v problematice viditelnosti kol jsou vypsány v tabulce 10.1. I přestože nejvíce chyb bylo opraveno v Bulharsku, tak počet těchto chyb je natolik vysoké číslo, že bude jejich oprava trvat větší množství sprintů, než si vývojové týmy vyhradí dostatek času na opravu. V zemích jako je Ukrajina, Finsko a Belgie již nebyla shledána oproti prvním testům žádná chyba. Výsledky jsou zobrazeny i graficky v grafu 10.1.

U výsledků druhých testů v problematice viditelnosti CO_2 je situace zcela odlišná. Tyto chyby bylo opraveny neprodleně a v druhých testech již nebyla odhalena žádná chyba. Proto nejsou výsledky nijak v tabulce ani graficky zobrazeny.

Tabulka 10.1: Výsledky druhých testů viditelnosti kol

Zkratka	Název země	Počet nalezených problémů
BGR	Bulharsko	42
LVA	Litva	21
POL	Posko	18
AUS	Austrálie	16
MDA	Moldavsko	15
LTU	Lotyšsko	9
TWN	Taiwan	3
DNK	Dánsko	1
UKR	Ukrajina	0
FIN	Finsko	0
BEL	Belgie	0



Obrázek 10.1: Graf neviditelosti kol po opravách

11 Zhodnocení výsledků

Pro zhodnocení slouží porovnání prvních a druhých testů chyb, které od sebe odděluje časový interval jednoho měsíce. V tabulce 11.1 je zobrazeno porovnání výsledků testů viditelnosti kol. Z porovnání lze zpozorovat, že nejvíce chyb bylo odstraněno v Bulharsku, kde byl ale výskyt chyb největší. K úplné opravě došlo pouze u Ukrajiny, Taiwanu a Finska, kde byly nalezeny pouze jednotky chyb. Naopak k žádné nápravě nedošlo v Austrálii a Moldavsku. Nicméně za pár sprintů, z celkové počtu 253 nalezených chyb, bylo odstraněno 122 chyb a došlo tedy k nápravě téměř 50 % chyb ve viditelnosti kol. Tento krok byl v této problematice nejobtížnější a předpokládá se, že ostatní chyby budou také opraveny v několika dalších sprintech. Jakmile budou chyby opraveny a budou automatizované testy dále podávat zpětnou vazbu, nebude již téměř vůbec možné, aby se chyb mohlo akumulovat velké množství. Proto bude poté odstranění těchto chyb méně časově náročné a bude k němu moci dojít mnohem rychleji.

V problematice viditelnosti emisních hodnot CO_2 u všech motorizací je situace zcela odlišná. Tyto chyby byly během pár sprintů odstraněny naprosto všechny. To je také způsobeno tím, že chyb v této problematice bylo v každé zemi pouze v řádu jednotek. Porovnání mezi počtem chyb viditelnosti emisních hodnot CO_2 v prvních a druhých testech je v tabulce 11.2.

Z celkových výsledků lze tedy usoudit, že reporty testů dosáhly cíle opravení chyb v obou problematikách. Proto se počítá s jejich funkcí a častým reportem chyb v problematice viditelnosti kol a emisních hodnot CO_2 i nadále. Navíc testy mohou být do budoucna i rozšířeny a mohou sloužit i pro jiné testovací účely.

Tabulka 11.1: Porovnání výsledků testů viditelnosti kol

Zkratka	Název země	1. výsledky	2. výsledky	rozdíl
BGR	Bulharsko	92	42	-50
POL	Posko	42	18	-24
DNK	Dánsko	22	1	-21
LVA	Litva	41	21	-14
LTU	Lotyšsko	17	9	-8
UKR	Ukrajina	2	0	-2
TWN	Taiwan	4	3	-1
BEL	Belgie	1	0	-1
FIN	Finsko	1	0	-1
AUS	Austrálie	16	16	0
MDA	Moldavsko	15	15	0

Tabulka 11.2: Porovnání výsledků testů viditelnosti CO_2

Zkratka	Název země	1. výsledky	2. výsledky	rozdíl
EST	Estonsko	3	0	-3
LUX	Lucembursko	3	0	-3
ESP	Španělsko	2	0	-2
FRA	Francie	2	0	-2
POL	Posko	2	0	-2
DEU	Německo	1	0	-1
DNK	Dánsko	1	0	-1
FIN	Finsko	1	0	-1
IRL	Irsko	1	0	-1
LTU	Lotyšsko	1	0	-1

12 Závěr

Cílem práce bylo zhodnotit využité řešení pro automatizované testy ve firmě Škoda Auto. V této firmě je pro automatizované testy webové aplikace Car Configurator využito nástroje Robot Framework. Výhodou tohoto nástroje je jeho jednoduchost. Psaní automatizovaných testů je poté jednoduše pochopitelné, protože jednotlivé funkce a procedury se volají pomocí klíčových slov. Z tohoto důvodu je tento nástroj nejvhodnější, jelikož testovací tým není součástí firmy, ale poptávají se kapacity z externích firem. Proto při změně části týmu či změně dodavatele testovacích kapacit je adaptace na toto vývojové prostřední velmi rychlá.

Nedílnou součástí práce bylo i zhodnocení, zda by kapacity testovacího týmu aplikace Car Configurator mohly být poskytnuty i jinému oddělení. Problémem zde je, že primárně kapacita tohoto týmu slouží pro testování funkcionality. Nicméně ve Škoda Auto existuje i tým, který se zabývá problematikou naimportovaných dat, tedy zda importéři v jednotlivých zemí nahráli do aplikace správná data. Pokud totiž nejsou data nahrána korektně, tak dochází k chybám v možnosti konfigurace. Zákazník je poté omezen možnostmi konfigurace nebo se mu nedostávají určité informace. Jako doporučení je tedy možnost využití části kapacity testovacího týmu aplikace Car Configurator i pro tuto problematiku. Za následek by to mělo mít větší efektivitu testovacího týmu a lepší komplexní pokrytí detekce chyb v aplikaci.

Hlavním cílem práce bylo dosáhnout plně automatizovaného skriptu, který bude dohlížet nad webovou aplikací Car Configurator a bude reportovat nalezené chyby. Primárně tedy chyby způsobené importem dat a to konkrétně viditelnost kol a viditelnost hodnot CO_2. Byly vyvinuty tedy 2 skripty pro detekci těchto chyb. Skripty mají jako vstupní data veškeré URL adresy, na kterých je požadován test. Výstupními daty ze skriptů jsou veškeré nalezené chyby, které jsou ukládány do excelových souborů s názvy testovaných zemích. Pomocí těchto testů bylo odhaleno celkem 270 chyb, z toho 253 chyb ve viditelnosti kol a 17 chyb ve viditelnosti emisních hodnot CO_2. Z testovaných 28 zemí byla chyba nalezena v celkem 17 zemích. Informace o nalezených chybách byly pomocí rollout týmu předány na importéry s požadavkem na jejich opravu. Po reportu těchto chyb bylo během dvou sprintů opraveno více než 51 % chyb a předpokládá se, že během dalších sprintů budou opraveny veškeré tyto nalezené chyby. Prodleva opravy je v tomto případě pravděpodobně z důvodu velkého množství nalezených chyb. Nicméně až budou automatizované testy tyto chyby reportovat v krátkých časových intervalech, tak nebude akumulováno tolik chyb a jejich oprava bude tedy mnohem rychlejší a efektivnější.

Použitá literatura

[1] MYSLÍN, Josef. *Scrum*. Albatros Media a.s., 2016. ISBN 8025146561.

[2] ŠOCHOVÁ, Zuzana a Eduard KUNCE. *Agilní metody řízení projektů*. 2. vyd. Brno: Computer Press, 2019. ISBN 978-80-251-4961-4.

[3] MCCONNELL, Steve. *Dokonalý kód: umění programování a techniky tvorby software*. Brno: 80-251-0849-X, 2005. ISBN 80-251-0849-X.

[4] KOĎOUSKOVÁ, Barbora. *Co je agilní vývoj aplikací a kdy ho využívaT* [online]. 2022 [cit. 2022-04-22]. Dostupné z: www.rascasone.com/cs/blog/co-je-agilni-vyvoj.

[5] SUMMERFIELD, Mark. *Python 3: výukový kurz*. 2. vydání. Brno: Computer Press, 2021. ISBN 978-80-251-5030-6.

[6] PILGRIM, Mark. *Ponořme se do Python(u) 3: Dive into Python 3*. 1. vyd. Praha: CZ.NIC, 2010. ISBN 978-80-904248-2-1.

[7] ORACLE. *What is Java technology and why do I need it?* [Online]. 2021 [cit. 2022-02-02]. Dostupné z: www.java.com/en/download/help/whatis_java.html.

[8] DUSTIN, Elfriede, Thom GARRETT a Bernie GAUF. *Implementing Automated Software Testing*. Verlag, 2009. ISBN 978-0-321-58051-1.

[9] ROBOT FRAMEWORK, ry. *Robot Framework* [online]. 2022 [cit. 2022-02-16]. Dostupné z: www.robotframework.org/.

[10] GITHUB, Inc. *SeleniumLibrary* [online]. 2022 [cit. 2022-02-16]. Dostupné z: www.github.com/robotframework/SeleniumLibrary/.

[11] CHACON, Scott a Jason LONG. *Git* [online]. 2022 [cit. 2022-02-16]. Dostupné z: www.git-scm.com/.

[12] GITHUB. *Where the world builds software* [online]. 2022 [cit. 2022-03-16]. Dostupné z: https://github.com/about.

[13] LUCCA, Giuseppe A. a Anna Rita FASOLINO. *Web Application Testing*. Springer, Berlin, Heidelberg, 2006. ISBN 978-3-540-28196-2.

[14] DOCS, MDN Web. *World Wide Web* [online]. 2022 [cit. 2022-04-22]. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/World_Wide_Web.

[15] ŠTRÁFELDA, Jan. *HTML* [online]. 2022 [cit. 2022-04-22]. Dostupné z: www.strafelda.cz/html.