Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra informačních technologií

Automatické testy v Selenium framework

Bakalářská práce

Autor: Adam Kučera
Studijní obor: Informační management

Vedoucí práce: Mgr. Josef Horálek, Ph.D.

Hradec Králové Duben 2022

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

podpis

V Hradci Králové dne 20.4.2022 Adam Kučera

Poděkování:

Tímto bych rád poděkoval vedoucímu bakalářské práce Mgr. Josefu Horálkovi, Ph.D. za metodické vedení práce, za čas, který mi věnoval a také za cenné a odborné rady, které mi během zpracování bakalářské práce poskytoval. Rád bych také poděkoval svým rodičům a přítelkyni, kteří mi byli oporou po celou dobu mého studia.

Anotace

Tato bakalářská práce se zabývá vytvářením automatických testů pro otestování uživatelského rozhraní aplikace, za použití technologie Selenium. Je popsáno, co předchází vytváření těchto testů, jejich konkrétní realizace a následné zobrazení výsledků těchto testů a jejich interpretace. Prakticky je vytvořeno pět konkrétních automatických testů. Testovací skript a jeho jednotlivé kroky jsou detailněji popsány a vysvětleny.

Práce se dále zaměřuje na základní principy testování softwaru z obecného hlediska a rozdílné druhy testování. Dále se detailněji zaměřuje na manuální a automaté testování. Jsou porovnány výhody a nevýhody těchto dvou typů. Dále je zpracován detailnější popis Selenium frameworku, kde jsou popsány jeho jednotlivé nástroje – Selenium IDE, Selenium WebDriver a Selenium Grid a příklady jejich použití.

Klíčová slova

Selenium, tester, manuální testování, automatické testování, software, webová aplikace, C#

Annotation

Title: Automation tests in Selenium framework

This bachelor thesis deals with creating automated tests for testing the user interface of an application, using Selenium technology. It describes what precedes the creation of these tests, their concrete implementation, and the subsequent display and interpretation of the results of these tests. Five specific automated tests has been created. The test script and its individual steps are described and explained in detail.

The thesis also focuses on the basic principles of software testing from a general point of view and different types of testing. It also focuses in detail on manual and automated testing. The advantages and disadvantages of these two types are compared. Furthermore, a more detailed description of the Selenium framework is elaborated, describing its individual tools - Selenium IDE, Selenium WebDriver and Selenium Grid and examples of their use.

Keywords:

Selenium, tester, manual testing, automatic testing, software, web application, C#

Obsah

Seznam obrázků

1 Úvod

Testování softwaru je v dnešní době již běžnou součástí vývojového cyklu aplikací. Není možné automaticky spoléhat na to, že bude vyvíjený software naproste bez chyb. K jejich nalezení slouží proces testování softwaru, který zabraňuje, aby se případné chyby objevily až v pozdější fázi vývoje a tím by byly náklady na jejich opravu vyšší, popř. by muselo dojít k větší úpravě kódu aplikace. Tento proces se může skládat z více druhů testování. Tyto druhy testování se dělí podle toho, v jaké fázi vývoje se aplikace nachází, testované funkcionality a zvoleného typu testování. Nejzákladnější rozdělení prováděných testů je na testy automatické a manuální. Automatické testování je hlavním tématem této práce. Jedná se o proces, kdy je sada testů spuštěna automaticky, za pomoci potřebných nástrojů.

Teoretická část práce je věnována poznatkům o testování softwaru z obecného hlediska. Důraz je kladen na automatické testování grafického rozhraní aplikace za pomoci nástroje Selenium. Tento nástroj je popsán více do detailu a zmíněny jsou jednotlivé nástroje, které nabízí. V praktické části byla vytvořena sada automatických testů, která testuje uživatelské grafické rozhraní aplikace anonymní výrobní společnosti. Tyto testy jsou napsané v programovacím jazyce C# a s využitím nástroje Selenium WebDriver. Tento nástroj umožňuje simulovat interakce reálného uživatele s webovou aplikací. Práce se také zaměřuje na reprezentaci výsledků těchto testů, ať už se jedná o lokálně spuštěné testy, nebo testy běžící v Kubernetes clusteru.

2 Důvody testování softwaru

2.1 Důvody testování

Testování softwaru se v posledních letech stává čím dál důležitějším krokem při jeho vývoji. Ať už se jedná o desktopové aplikace, či webové aplikace, u všech softwarových produktů záleží na co nejvyšší možné kvalitě a bezchybnosti. Aplikace jsou stále více složitější a s rostoucím konkurenčním tlakem se stává zajištění kvality vyvíjeného softwaru důležitým aspektem v konkurenčním boji. [1] Právě tento úkol má během vývoje softwaru na starosti testovací tým. V některých firmách je tento tým veden pod názvem Quality assurance (QA), v překladu zajištění kvality. Podle IEEE zní dvě definice softwarové kvality takto [2]:

1. Míra, do jaké systém, proces nebo komponenta splňuje stanovené požadavky.
2. Míra, do jaké systém, proces nebo komponenta splňuje zákaznické potřeby a uživatelská očekávání.

2.1.1 Chyby a selhání softwaru

Testování je proces spouštění programu se záměrem nalézt chyby. [3] Právě nalézt chyby ve vyvíjeném softwaru je jedna z hlavních činností testování. Původ selhání softwaru spočívá v jeho chybách. Ty mohou mít podobu gramatických chyb v kódu programu nebo logických chyb při převádění požadavků klienta. Není zaručené, že všechny chyby softwaru povedou k jeho nepoužitelnosti. V mnoha případech chyby zanechané v kódu nebudou ovlivňovat funkčnost softwaru jako celku. Od softwarové chyby musíme odlišit tzv. selhání softwaru. Toto selhání může nastat, pokud se splní některé podmínky nezbytné k aktivování poruchy aplikace. K softwarovému selhání nemusí nikdy dojít, protože chyba v aplikaci nemusí být aktivována a pro uživatele tak selhání aplikace nikdy nenastane. Toto však neznamená, že se v softwaru nenacházejí žádné chyby. Znamená to pouze, že měl uživatel štěstí a chybu v softwaru neaktivoval. Autor Daniel Galin ve své knize uvádí definice softwarových chyb, poruch a selhání takto [4]:

- Softwarové chyby (errors) jsou části kódu, které jsou částečně nebo zcela nesprávné v důsledku gramatické, logické, nebo jiné chyby vytvořené systémovým analytikem, programátorem, nebo jiný členem týmu vývoje softwaru.
- Softwarové chyby se stávají softwarovým selháním pouze tehdy, když dojde k jejich „aktivaci". Tj. když se uživatel pokusí použít určitou část softwaru, která je vadná. Důvodem jakéhokoliv selhání softwaru je tedy softwarová chyba.

2.1.2 Cena chyb

Podstatné je, aby se chyby v softwaru našly co nejdříve. Čím více se software nachází v pozdější fázi vývoje, tím se cena na opravení chyby zvyšuje. Z toho vyplývá, že testování softwaru by mělo být součástí všech fází jeho vývoje a ne pouze ve fázi testování.

Mezi nejdražší chyby, které se v historii lidstva objevily, můžeme zařadit například:

- Civilní raketu Arian 5, která byla jednou z pěti evropských raket používaných pro vypouštění satelitů do vesmíru. V roce 1996 tato raketa pouhých 40 sekund po startu ve Francii explodovala. Jednalo se o bezpilotní zařízení, proto nedošlo ke ztrátě na životech. Celkové ztráty jsou vyčísleny na 500 milionů dolarů. [5] Chybu v softwaru způsobil tzv. integer overflow bug. Tato chyba se vyskytne, pokud se pokusíme do celočíselné proměnné (integer) uložit hodnotu, která přesahuje maximální možnou hodnotu, kterou tento datový typ může obsahovat. [6]
- Tzv. Problém roku 2000. Mezi roky 1960 a 1980 programátoři používali pro uložení roku v softwarech pouze poslední dvě čísla. Ke zkracování roku docházelo z důvodu ušetření paměti pro uložení dat v počítači. Při blížícím se roku 2000 se začaly objevovat obavy, aby si počítače nevyložily poslední dvě nuly jako rok 1900, místo roku 2000. Amerika vynaložila miliony dolarů, aby tomuto problému předešla. Přesná částka, která musela být celosvětově

vynaložena na opravení programů není známa, z důvodu těžkého vyčíslení škod a nákladů na úpravu programů. [7]

Nenalezené chyby ale nemusí znamenat pouze vyšší finanční náklady. Pokud se jedná o systémy, které souvisí s lidskými životy, například lékařské, vojenské nebo letecké systémy, má nalezení veškerých možných nedostatků a chyb v těchto systémech mnohem důležitější prioritu. Nenalezení nedostatků v těchto systémech může mít i fatální následky.

Dřívější nalezení chyb neznamená pouze menší náklady, ale také jejich snadnější opravení. Při vývoji dochází k přidávání nových funkcionalit ke stávajícím. Pokud se na chybu přijde až v pozdějších fázích vývoje, může se stát, že oprava kódu ovlivní funkcionality, které jsou závislé právě na opravené chybě. Optimalizací těchto navazujících funkcionalit se zvyšují náklady na projekt a čas vývoje.

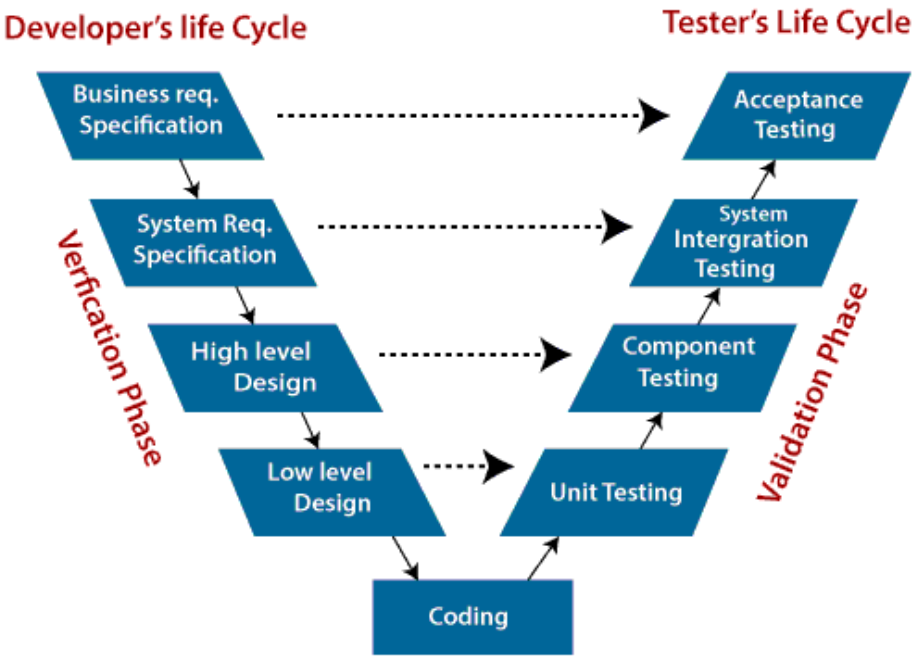
3 Druhy testování

Jaký typ testování je použit závisí na mnoha vlastnostech. Například na použitém modelu vývoje aplikace [8], dále například záleží v jaké části vývoje se testy provádějí, podle toho se zvolí požadované testovací úroveň a jí odpovídající testy.

Druhy testování se dále liší podle toho, jaká vrstva aplikace se bude právě testovat – frontend nebo backend. Jakou má tester znalost zdrojového kódu a jakou má firma nastavenou testovací strategii, zda se zaměřuje na manuální, či automatické testování.

3.1 V-Model

Způsoby testování se liší podle toho, v jaké konkrétní části vývoje se aktuálně projekt nachází. Obrázek níže zobrazuje, jaké typy testování jsou spojeny s danými částmi vývoje.



Obrázek 1: V-Model

Zdroj: Web [9]

V-model začleňuje testování softwaru do celého cyklu vývoje softwaru. Každé části vývoje softwaru patří také určitý typ testování. Již z názvu modelu je patrné, že

má tvar písmene „V", kdy se sekvenčně postupuje z levého horního rohu směrem dolů doprava a poté opět nahoru. Tak vypadá základní posloupnost vývojových a testovacích činností. Je zřejmé, že testování by mělo začít již na samém začátku vývoje a pokračovat až do konce projektu. [10]

Na obrázku se také vyskytují fráze verifikace a validace. Je důležité tyto termíny umět rozlišit. Jedná se o dva samostatné postupy, které se provádějí, aby se zajistilo, že vyvíjený software splňuje uživatelské požadavky a je bez chyb.

Verifikace, nebo-li ověření, se dělá, pokud odpovídáme na otázku: „Vytváříme správný produkt?". [11] Testuje se tedy, zda jsou splněny zákaznické požadavky na daný produkt.

Validace se provádí, pokud odpovídáme na otázku: „Vytváříme produkt správně?". [11] Testuje se zde, aby byl konečný produkt s co nejmenším množstvím chyb, a aby co nejvíce naplnil očekávání zákazníků.

3.2 Testovací úrovně

Testovací proces se skládá z velkého množství konkrétních testů. Tyto testy jsou rozděleny do tzv. úrovní testování. Celkem rozdělujeme 4 testovací úrovně. [12]

3.2.1 Component testing

Testování komponent (někdy také označováno jako jednotkové testování) má za úkol najít chyby v jednotlivých částech softwaru a ověřit jejich funkčnost. Tyto jednotky jsou samostatně testovatelné a proto mohou být otestovány izolovaně od zbytku systému. Ve většině případech je při tomto testování potřebný přístup ke zdrojovému kódu, protože se testují například jednotlivé funkce, nebo třídy v systému. Toto testování provádí sám vývojář za pomoci unit test frameworku. Při nalezení chyby dojde okamžitě k jejímu opravení. Jedná se o první testování vyvíjeného softwaru.

3.2.2 Integration testing

Integrační testování má za úkol otestovat interakce mezi jednotlivými částmi systému a rozhraní mezi komponentami systému, nebo mezi celými systémy. Například pokud dochází k integraci modulu A s modulem B, jsou testy zaměřeny

pouze na jejich vzájemnou komunikaci. Toto testování se zazývá integrační testování.

3.2.3 System testing

Testování systému zkoumá chování celého systému tak, aby odpovídalo zadání. K tomu testování by mělo být využito samostatné testovací prostředí. Toto prostředí by nejlépe mělo odpovídat konečnému produkčnímu prostředí. Díky tomu se minimalizují rizika spojená s konkrétním prostředím. Uvedené testování může například zahrnovat testování na základě splnění požadavků zákazníků, případy použití (tzv. use case), obchodní procesy, komunikaci s operačním systémem a systémovými prostředky, nebo jiný popis chování systému.

3.2.4 Acceptance testing

Za akceptační testování jsou ve většině případu zodpovědní zákazníci nebo uživatelé systému. Hlavním cílem tohoto testování není nalezení chyb, ale spíše získání zákaznické důvěry v daný produkt a zda splňuje všechny jejich požadavky stanovené na začátku vývoje. Samozřejmě i zákazník může objevit chybu a a pak musí být opravena.

Ze základního popisu úrovní testování je zřejmé, že na testování se nepodílí pouze testeři. Do této činnost je v průběhu vývoje zapojeno více osob, ať už se jedná o vývojáře, systémové administrátory, nebo zákazníky a uživatele systému.

3.3 Vrstvy aplikace

Webové aplikace jsou při vývoji rozděleny na backendovou a frontendovou část. K jejich vývoji jsou použity rozdílné postupy a technologie. Aby zákazník mohl požadovanou aplikaci používat, musí obě tyto vrstvy fungovat.

3.3.1 Frontend

Testování frontendu aplikace je velice důležité. Jde totiž o část, kterou bude uživatel nejvíce použít a která je klíčová pro správně fungování aplikace. Pokud nebude fungovat správně, zákazník nebude chtít danou aplikaci používat.

Při frontend testování se kontroluje, zda vytvořené stránky odpovídají požadovanému prototypu, který odráží zákaznické požadavky. Zda obsahuje

všechny elementy, které tam mají být, zda jsou na odpovídajícím místě a plní požadovanou funkci.

Jedná se o poměrně repetetivní činnost, proto se firmy snaží tento typ testování co nejvíce zautomatizovat. K tomuto účelu se využívá například nástroj Selenium, který je použit v praktické části této práce a bude hlouběji vysvětlen dále.

3.3.2 Backend

Backendová část aplikace je většinou tvořena zdrojovým kódem, databází a serverem, na kterém je vytvořená aplikace spuštěna. Existuje množství testovacích úrovní, ve kterých se testuje vždy určitá část backendu aplikace. Podrobnější popis je v kapitole 2.2.2.

3.4 Znalost zdrojového kódu

Testovací techniky můžeme rozdělit podle znalosti zdrojového kódu na testování černé skříňky a testování bílé skříňky.

3.4.1 Black-box

Black-box, neboli černá skříňka, je způsob testování, kdy tester nemá přístup ke zdrojovému kódu aplikace. Hlavním zdrojem k vytváření testů je pouze specifikace daného produktu. Nezabýváme se tak vnitřními mechanismy systému. K testu se použije rozhraní, které testovaný objekt poskytuje. Kontroluje se výstup z objektu. který by měl odpovídat očekávanému výsledku. [13] Tento způsob se používá především u akceptačního a funkčního testování.

3.4.2 White-box

White box, neboli bílá skříňka, je testování založené na kompletní znalosti systému. To znamená, že má tester přístup ke zdrojovému kódu aplikace a rozumí vnitřním mechanismům systému. Toto testování se zaměřuje na tok dat uvnitř programu. [14] Často je tímto člověkem právě sám vývojář, který má o zdrojovém kódu největší znalosti. Z tohoto důvodu sem můžeme zařadit například unit testy, kdy tento typ testů vytváří převážně sám programátor.

3.5 Způsob provedení testů

Testování můžeme dále rozdělit podle způsobu provádění testů. Pro každou část vývoje může být vhodnější jiný způsob. Záleží také na velikosti projektu. U menších projektů je manuální testování dostatečné, ale u větších se klade důraz na co možná největší automatizaci.

3.5.1 Manuální testování

Tato technika testování se vyznačuje tím, že si tester připraví veškeré testovací případy v tzv. testovacím plánu. Jedná se o seznam kroků, které musí tester otestovat, aby zhodnotil chování systému. Testovací plán by měl obsahovat úvod, testované položky, funkce, které se mají či nemají testovat, kritéria úspěšnosti/neúspěšnosti testů, testovací přístup a další náležitosti. [15] Po provedení testů, porovná skutečné výsledky očekávanými a případně odhalí chyby v systému. Jedná se o nejstarší metodu testování softwaru. [16] Opakované manuální testování, nebo u rozsáhlých projektů je časově náročné a finančně nevýhodné. Stejně tak je nutné vynaložit úsilí na udržení aktuálního testovacího plánu. Existuje také větší možnost udělání chyby ze strany testera. Z těchto důvodů je snaha co nejvíce testování zautomatizovat.

Exploratory testing je typ manuálního testování, při kterém se nenásleduje žádný připravený testovací plán. Není proto tak časově náročné na přípravu a chyby v softwaru se tak mohou najít rychleji. Kvalita tohoto typu testování závisí na individuálních zkušenostech a dovednostech testera. [17]

Protipól manuálního testování je testovaní automatické, které v dnešní době získává na popularitě a ve firmách je stále více využíváno. Protože je tato práce zaměřená na převážně tento typ testování, je tomuto tématu věnována následující samostatná kapitola.

4 Automatické testování

V této kapitole je popsán obecný úvod týkající se automatického testování. Dále jsou uvedeny důvody, které vedou stále více firem k využívání právě této disciplíny. Na konec kapitloly jsou popsány silné a slabé stránky automatického testování.

4.1 Principy

Automatické testování softwaru je druhým typem způsovu testování softwaru na základě způsobu provádění. Prvním typem je manuální testování. Jak už název u toho typu napovídá, tento typ testování provádí tester manuálně. Automatické testy jsou prováděny počítačem, pomocí skriptů, které musí tester vymyslet a naprogramovat. Hlavní náplní testera je tedy v případě automatického testování již zmíněné vytváření testovacích skriptů, následná jejich údržba a kontrola výsledků provedených testů. Automatizace testů tedy zahrnuje kolekci testovacích skriptů, které jsou provedeny bez lidského zásahu. [18]

Tento způsob testování používá stále více firem a jeho důležitost pří vývoji softwaru se tak zvyšuje. Testování softwaru v dnešní době není pouze o prokazování kvality a absenci chyb, ale i o zajištění kvality. Nejde pouze o hledání chyb, ale je kladen velký důraz na jejich předcházení. Na tento fakt mají především vliv agilní metodiky použity při vývoji softwaru, které v dnešní době převažují při vývoji nových programů a aplikací.

Obrázek níže porovnává úspěšnost projektů používající agilní a vodopádový model. „Successful“ značí projekty úspěšně dokončené ve stanoveném čase a rozpočtu. „Challenged“ jsou dokončené projekty, ale přesahující stanovený čas a rozpočet. „Failed“ jsou projekty, které nebyly dokončeny.Je patrné, že při agilním modelu vývoje je znatelně větší úspěšnost projektům, než při vodopádovém.

Obrázek 2: Agile vs Waterfall

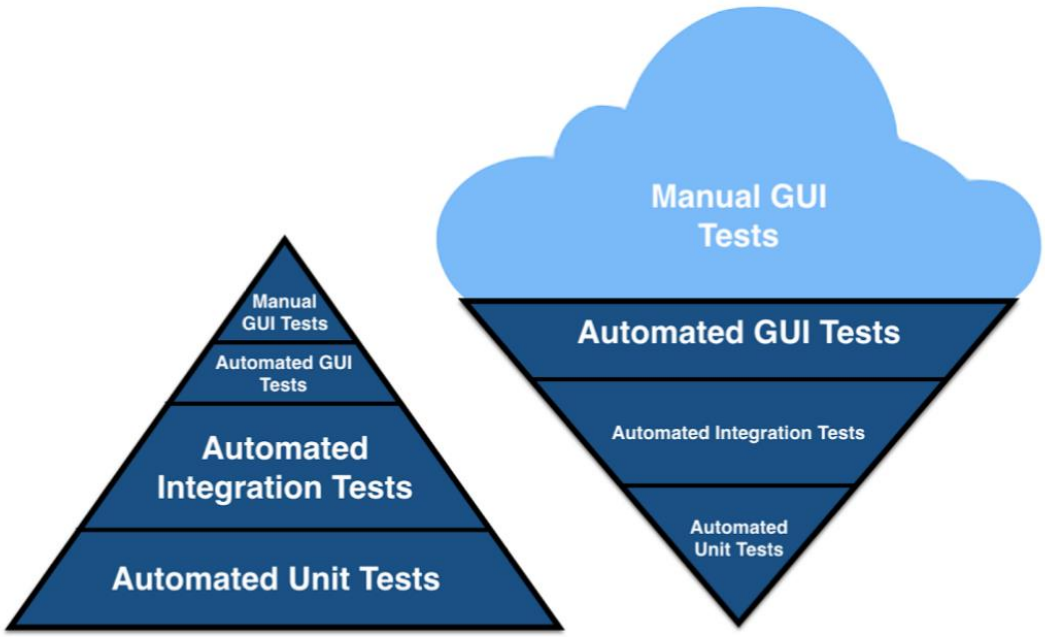
Zdroj: Web [19]

Důsledné používání automatického testování je jedinou šancí, jak zajistit kvalitu potencionálně dosažitelného produktu na konci každého sprintu. [20]

Autor Manfred Baumgartner uvádí v knize Agile Testing: The Agile Way to Quality tyto principy, které platí pro agilní testovací nástroje, a to bez ohledu na konrétní projekt [20]:

- Objektem testování je stále se pohybující cíl.
- Testy by měly být pouštěny, pokud je to možné, s každým novým buildem aplikace.
- Nástroje pro testování by měly být snadné a flexibilní.

Tzv. testovací pyramidy poskytují pohled na souvislost mezi počtem agilních metodik použitých při vývoji softwaru a počtem použitých automatických testů místo manuálních. [21] Na obrázku níže jsou zobrazeny 2 tyto pyramidy. Levá pyramida představuje agilní metodiky a pravá tradiční metodiky, díky svému vzhledu bývá označována jako „zmrzlina“. Jak je z obrázku patrné, v agilních metodikách je na automatické testování kladen důraz již od nejnižší úrovně. Díky tomu je možné potencionální chyby odhalit co nejdříve, a tím snížit náklady a čas na jejich opravu.



Obrázek 3: Testovací Pyramidy

Zdroj: Web [21]

Aby bylo automatické testování efektivní a udržitelné, musí být vyjasněno a definováno mnoho aspektů. Mezi ně patří například organizační a technická integrace do DevOps prostředí a procesů, vhodný výběr automatizačních nástrojů, návrh použitého framewotku pro testování, plánovaní činností automatizace a spouštění automatických testů v synchornizaci s vývojovými sprinty.

Další důvod proč se tento typ testování těší ve firmách velké oblibě je fakt, že ze stran zákazníků jsou na systémy kladeny stálě větší nároky, a tím se zvyšuje komplexnost vyvíjených programů. Logicky s tím rostou i nároky na testování těchto programů. Proto není možné z hlediska lidských zdrojů, časového a finančního provádět všechny testy manuálně.

4.2 Výhody a nevýhody

Automatické testování má mnoho výhod oproti manuálnímu, ale i určité limity a nevýhody. V roce 2012 byl usktučněn online dotazník, kterého se zůčastnilo celkem 115 respondentů. Skoro 55% z této skupiny tvořili lidé se zaměřením na testování, necelých 25% programátoři a zbytek byl tvořen skupinou obsahující

například systémové designéry, architekty a projektové manažery. Většina dotázaných pracovala v agilní metodice vývoje softwaru. Z předložených možností vybrali tyto jako výhody automatického testování [22]:

- Znovupoužitelnost testů
- Vyšší pokrytí aplikace testy
- Zvýšení kvality produktu
- Rychlejší oproti manuálnímu testování
- Snížení času potřebného pro testování
- Snížení nákladů v dlouhodobém měřítku
- Lepší detekce chyb

Ze stejného dotazníku vzešlo i několik bodů, které účastníci viděli jako nevýhody automatického testování:

- Náročná tvorba a údržba testovacích scriptů
- Potřeba dostatečně kvalifikovaných zaměstnanců
- Větší náklady na pořízení potřebných nástrojů a školení k jejich používání
- Nedokáže plnět nahradit manuální testování
- Vysoké náklady ze začátku zavádění procesu automatického testování
- Nereálná očekávání

4.2.1 Výhody

Manuální testování může být velice pomalé. Tester musí znát požadované chování aplikace, provést ručně jednotlivé kroky testovacího scénáře a na konec zaznamenat a vyhodnotit výsledky testu. Automatické testování umožní většinu těchto kroků provést automaticky, a díky tomu ušetřit spoustu času.

Odpadá také možnost chyby ze strany testera. Člověk může být ovlivněn mnoha faktory, jako je například únava, osobní problémy nebo nesoustředěnost. Ty mají za následek zvýšení šance udělat chybu při testování. Automatické testování těmito věcmi není ovlivněno a tak je prakticky odstraněna možnost udělat během běhu testu nechtěnou chybu.

Automatické testy lze spouštět kdykoliv bez nutnosti lidského zásahu. Mohou být napsány tak, aby se spouštěly automaticky například uprostřed noci, kdy v budově nikdo není. Tester pak může na začátku pracovní doby pouze zkontrolovat výsledky těchto testů a na jejich základě učinit další kroky.

Větší pokrytí dané aplikace je další z výhod tohoto typu testování. Zatímco při manuálním testování je tester schopen v jeden okamžik pokrýt pouze malou část aplikace, automatický test dokaže během stejného časového rozpětí pokrýt znatelně větší část aplikace.

Zatímco tester dokáže testovat v daný okamžik pouze jednu věc, automatické testy mohou být spouštěny paralelně. Toto se hodí například v microservice architekture, kdy je aplikace rozdělena na samostatné menší celky nazývané microservices. Automatické testy tak dokáží v jeden okamžik testovat víc než jednu tuto microservicu a tím znatelně ušetřit čas potřebný ke znovuotestování aplikace po nasazené úpravě kódu.

4.2.2 Nevýhody

I přes rostoucí popularitu a nesporné výhody, není automatické testování dokonalé. Asi největším limitem je čas potřebný k napsání skriptů, které tyto testy provádí. Nejvíce času zabere napsání testů pro nové systémové funcionality.

Po napsání testů je také potřeba je udržovat co nejaktuálnější. Jelikož jsou testy tvořeny kódem, je ho zapotřebí často měnit a upravovat, aby odpovídal aktuálním požadavkům a chování aplikace. Platí proto pravidlo, že by měl být testovací skript co nejkraší, aby bylo lehké ho aktualizovat. Nesmí to být ovšem na úkor kvality testů, které by měly vždy kontrolovat co největší část aplikace. O testovacím kódu lze přemýšlet spíše jako o závazku, než jako o hotové věci. [23]

Další nevýhodou je fakt, že pro správné psaní kvalitních testů musí být daný zaměstnanec dostatečně zkušený a mít potřebné znalosti pro splnění této podmínky. Pokud nejsou testy správně napsané, mohou poskytovat falešný pocit bezpečí, způsobený chybnými výsledky. Při takovýchto testech mohou nastat dva typy výsledků [24]:

1. Falešně pozitivní výsledek, anglicky False positive. To znamená, že testovací případ selže, ale testovaný software neobsahuje chybu, kterou se test pokouší zachytit. Vývojáři pak ztrácí čas hledáním a opravováním chyby, který neexistuje. Díky tomuto typu výsledků mohou někteří ztrácet důvěru v automatické testování a raději volit testování manuální.
2. Falešně negativní výsledek, anglicky False negative. Tento typ výsledku znamená, že testovací případ projde, ale testovaný software obsahuje chybu, kterou se test snaží zachytit. To má za následek výskyt chyb v produktu, které způsobují problémy zákazníkům.

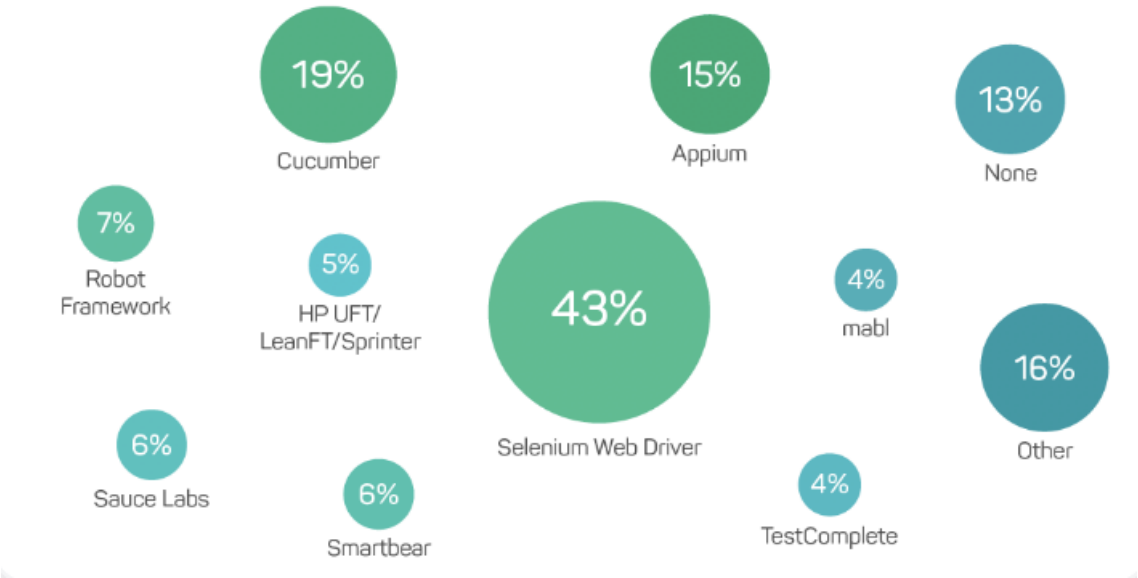
Oba tyto typy výsledků jsou považovány za škodlivé a mělo by se jim v co největší míře předcházet.

4.3 Používané technologie

Pro automatické testování dnes existuje nespočet nástrojů a technologií. Jejich použití závisí na mnoha faktorech. Určité nástroje se používají v různých testovacích úrovních – component testing, integration testing, system testing a acceptance testing. Rozdíl může také nastat podle toho, zda se testy provádějí na webové, mobilní, či desktopové aplikaci.

Rozdílně se také přistupuje k distribuci jednotlivých technologií. Některé nástroje jsou knihovnami programovacích jazyků. To je uživateli umožňuje rozšiřovat o svá řešení, a díky tomu lze tyto nástroje volně kombinovat a docílit tak požadovaného výsledku. Testovací nástroje však mohou být i vydávány jako kompletní aplikace. Příklad takové konrétní aplikace je JMeter. Jedná se o aplikaci napsanou v programovacím jazyce Java. K jejímu hlavnímu cíli patří provádění zátěžových testů a měření výkonu aplikací. [25]

Obrázek níže zobrazuje výsledky dotazníku, který se konal od Září 2019 do Března 2020. Zůčastnilo se ho necelých 1050 osob z řad testerů, programátorů a manažerů z celého světa.



Obrázek 4: Testing Tools

Zdroj: Web [26]

Z výsledků je patrné, že nejvíce procent získal nástroj Selenium. Jedná se o nástroj, který slouží převážně k testování frontendu webových aplikací, a to díky rozhraní, které slouží ke komunikaci s webových prohlížečem. Protože je tato práce zaměřena hlavně na tento nástroj, bude více popsán v další kapitole.

Na druhém místě se umístil nástroj Cucumber. Jedná se o nástroj, který podporuje tzv. Vývoj řízený chováním, anglicky Behavior Driven Development (BDD). Tento vývoj se vyznačuje tím, že před samotným psaním kódu programátory, jsou napsány scénáře, nebo akceptační testy, které popisují chování systému z pohledu zákazníka. Před začátkem vývoje si obě strany projdou tyto scénáře a potvrdí jejich správnost. Proto je někdy tento vývoj označován jako „specifikace příkladem“. [27] Klade se zde důraz na velkou spolupráci s koncovým zákazníkem.

Důležitým nástrojem při testování jsou také ty, které mají na starost component testing (unit testing). Tyto nástroje jsou dosuptné pro všechny programovací jazyky. Obecně obsahují metody, které porovnávají očekávané hodnoty s aktuálními. Často se používají v kombinaci s ostatními nástroji, například

právě se Seleniem. Mezi nejpoužívanější nástroje pro unit testy patří JUnit, NUnit, TestNG a XUnit. Poslední zmínění bude více popsán v následujících kapitolách.

5 Selenium

5.1 Představení

Jak napovídá obrázek výše, Selenium je nejpoužívanějším open source řešením pro automatické testování webových aplikací na světě. Licence open source znamená, že může být software volně používán, upravován a sdílen. [28] Zdrojový kód Selenia tak může upravit prakticky kdokoliv. Provedené změny ale musí projít procesem schválení, než jsou uvolněny k použití pro ostatní uživatele. V době psání této bakalářské práce bylo ve zdrojovém kódu provedeno necelých 28 000 změn. [29]

Velkou výhodou, a důvodem širokého používání, je vysoký počet podporovaných programovacích jazyků, ve kterých se může se Seleniem pracovat. Mezi nejznámější patří Java, Python, C#, Ruby, JavaScript, Ruby a PHP. Dalším kladem je široké pokrytí nejpoužívanějších internetových prohlížečů – Chrome, Firefox, Edge, Internet Explorer a Safari. [30]

Selenium není pouze jeden nástroj, ale nyní se pod tímto označením skrývají celkem 3 odlišné nástroje – Selenium WebDriver, Selenium IDE, Selenium Grid. [31] V minulosti tuto skupinu ještě doplňoval nástroj Selenium RC. Postupem času se ale stal nepotřebným a přestal se používat, resp. byl sloučen s nástrojem Selenium WebDriver. Nicméně je ale důležitý pro současný stav tohoto testovací nástroje, proto se i o něm v následující kapitole zmíním.

5.2 Historie

Selenium vytvořil v roce 2004 Jason Huggins. V té době pracoval jako programátor v americké společnosti ThoughtWorks a podílel se na vývoji webové aplikace, která vyžadovala časté testování. Postupem času si uvědomil, že neustálé repetetivní manuální testování aplikace se stává čím dál více neefektivní. Z tohoto důvodu vytvořil program v programovacím jazyce JavaScript, který dokázal automaticky kontrolovat činnost webového prohlížeče. Tento program pojmenoval JavaScriptTestRunner. Zanedlouho si uvědomil, jak velké přínosy by tento program mohl do světa testování přinést a uvolnil ho pro veřejnost pod licencí open source software. Program také přejmenoval z původního názvu na Selenium Core. [32]

Tento software však nebyl dokonalý. Hlavním nedostatkem byl tzv. problém "same-origin policy". Tento bezpečnostní mechanismus zakazuje kódu JavaScript přistupovat k prvkům z jiné domény, než ve které byl spuštěn. [33] Jako příklad si můžeme představit, že společnost Google má ve svém html kódu určitý program napsaný v jazyce JavaScript. Tento bezpečnostní mechanismus dovolí, aby měl daný program přístup pouze ke stránkám v rámci společnosti Google, jako je například Gmail, login, Calendar a tak dále. Nemůže však přistupovat ke stránkám z jiných webů. Díky tomuto problému museli testeři složitě na svých počítačích instalovat jak Selenium Core, tak i servery, které obsahovaly webové aplikace pro testování, tak, aby patřily do stejné domény. Tento problém vyřešil další programátor ze společnosti ThoughtWorks - Paul Hammant. Vyřešil ho vytvořením serveru, který fungoval jako HTTP proxy. Díky tomu prohlížeč věřil, že Selenium Core a testovaná aplikace pochází ze stejné domény. Tento systém pojmenoval jako Selenium Remote Control (Selenium RC), někdy také uváděno jako Selenium 1.

Japonský vývojář Shinya Kasatani vytvořil Selenium IDE. Jednalo se o rozšíření do prohlížeče Firefox, které dokáže zautomatizovat prohlížeč pomocí funkce záznamu a přehrávání. Toto rozšíření mělo za cíl urychlit vytváření testovacích případů. V roce 2006 bezplatně věnoval Selenium IDE projektu Selenium.

Simon Stewart, také programátor ve společnosti ThoughtWorks, v roce 2006 vytvořil tzv. WebDriver. Prohlížeče a webové aplikace se v této době stávaly stále výkonějšími a programy napsané v jazyku JavaScript, jako například Selenium RC, byly stále více omezovány. Místo JavaScriptu tak Simon Stewart vytvořil pro každý prohlížeč vlastního klienta (WebDriver), který byl naprogramován od začátku. Díky tomu je možné využít automatizaci s nativními možnostmi jednotlivých webových prohlížečů. K ovládání prohlížeče používá API rozhraní poskytované dodavateli prohlížečů. [34] Krátce po vytvoření začal pracovat na zpětné kompatibilitě se Selenium RC. V roce 2011 došlo ke spojení těchto dvou samostatných projektů a to vedlo ke vzniku nové verze Selenia – Selenium 2.

V roce 2008 vyvinul Patrick Lightbody Selenium Grid, další nástroj z rodiny Selenium. Původní název byl "Hosted QA". Hlavním důvodem vzniku tohoto nástroje bylo umožnit využití paralelního testování. Paralelní testování umožní provádět více

testů současně a tím tak znatelně zkrátit potřebný čas pro provedení všech automatických testů. Umožňuje spouštět testy na různých počítačích proti různým prohlížečům.

V roce 2016 bylo vydáno Selenium 3, které přineslo opravení chyb z předešlých verzí a větší stabilitu při testování. Nejnovější verze Selenium 4 byla vydaná v říjnu 2021. [35]

5.3 Nástroje

V této kapitole budou detailněji popsány konkrétní aktuální nástroje z rodiny Selenium.

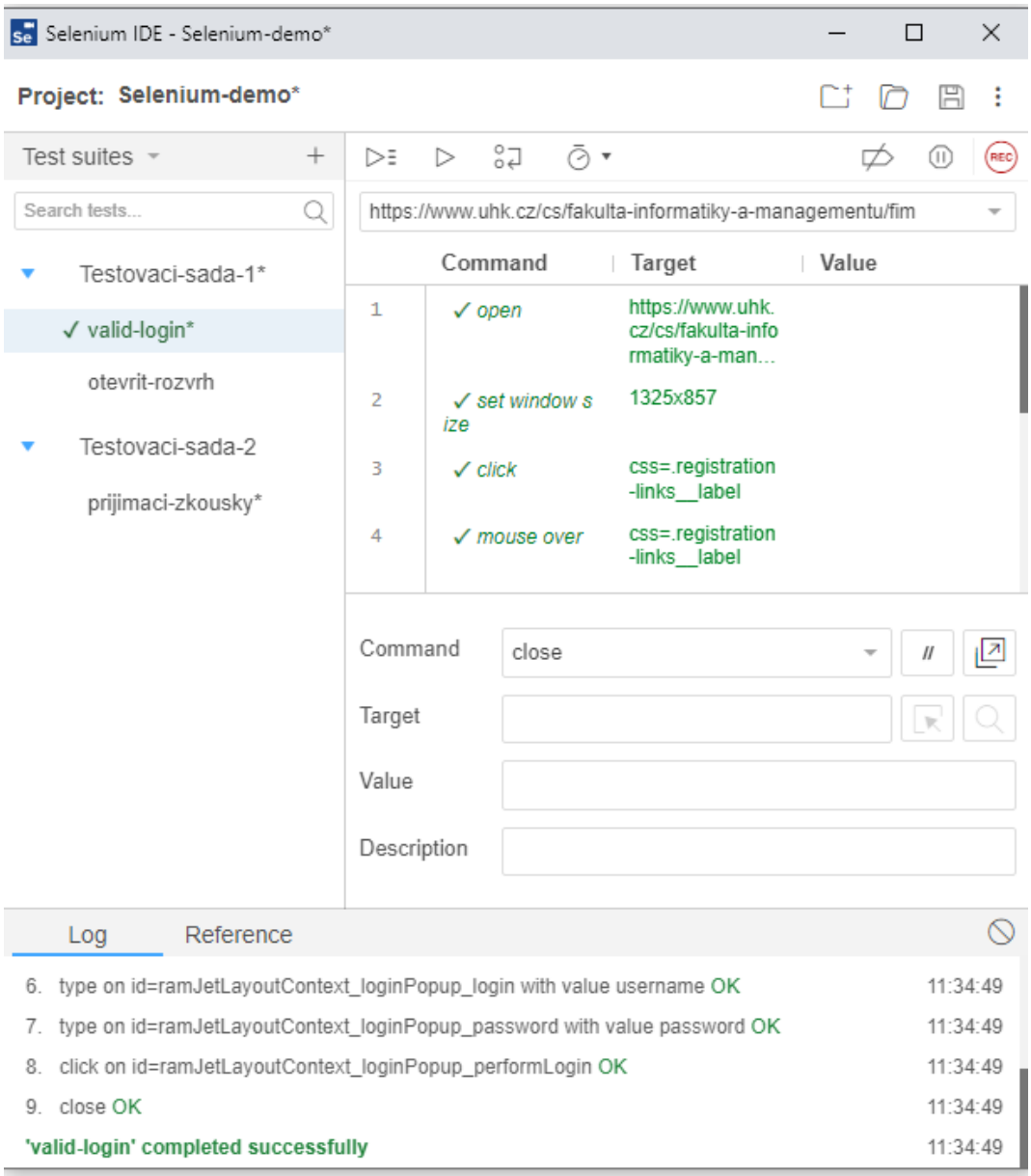
5.3.1 Selenium IDE

Selenium IDE (Integrated Development Environment) je nástroj, který primárně slouží k vývoji testovacích scénářů pro Selenium testy. K použití tohoto nástroje není zapotřebí žádné složité nastavování. Stačí stáhnout jako rozšíření do webového prohlížeče. V době psaní této práce je k dispozici pro prohlížeče Chrome a Firefox. Dříve bylo toto rozšíření dostupné pouze pro uživatele prohlížeče Firefox. V březnu roku 2020 bylo představeno i pro prohlížeč Chrome. Po stažení do prohlížeče je ihned připraven k použití. Poskytuje přehledné grafické uživatelské rozhraní pro zaznamenávání interakcí uživatele s webovou stránkou či aplikací.

Nástroj zaznamenává v prohlížeči akce uživatele, pomocí již existujících Selenium příkazů s parametry webových prvků, na které uživatel při nahrávání testu klikal. Díky snadnému použití je ideálním nástrojem i pro někoho, kdo nemá s vývojem automatiozvaných testovacích případů pro webové aplikace zkušenosti. Pro větší přehlednost je možné jednotlivé související testovací případy rozdělit do testovacích sad.

Na obrázku níže je možné vidět grafické rozhraní tohoto nástroje. V levém horním rohu se nachází název projektu. Pod ním jsou dvě testovací sady, kdy každá obsahuje konkrétní testovací případy. Jednotlivé kroky, ze kterých se testovací případ skládá, jsou v hlavním prostředním okně. U každého kroku je zobrazen příkaz, který Selenium vykoná a parametr, kterým je ve většině případů konkrétní element dané webové stránky. Horní navigační lišta obsahuje tlačítka pro spuštění

jednotlivých testů nebo všech testů v testovací sadě. Dále možnost zvolit rychlost vykonání testu a možnost přehrát test od konkrétního kroku. Důležité je červené tlačítko "REC" v pravé části navigační lišty, které slouží k nahrání jednotlivých kroků v testovacím případu. [36]



Obrázek 5: Selenium IDE

Zdroj: Vlastní zpracování

5.3.2 Selenium WebDriver

WebDriver pro automatizaci prohlížeče využívá rozhraní API poskytované dodavateli jednotlivých prohlížečů. Definuje jazykově neutrální rozhraní pro

ovládání chování webových prohlížečů. Tímto simuluje chování prohlížeče stejně, jako by ho ovládal reálný uživatel. Díky tomu je možné zautomatizovat hlavní prohlížeče na trhu. [37]

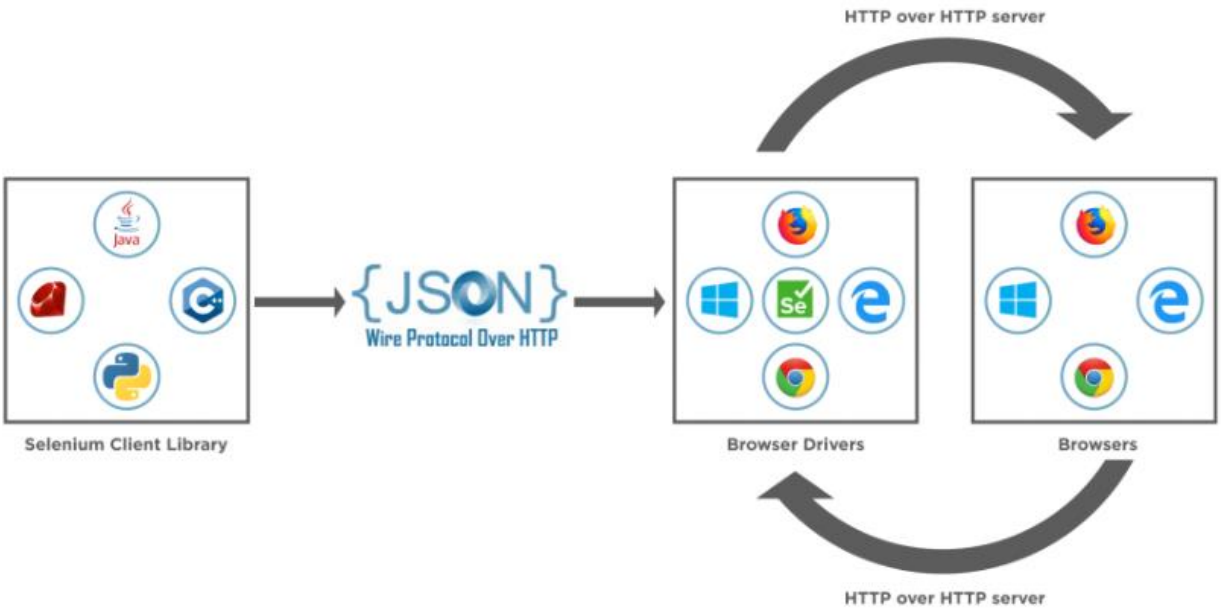
Každý prohlížeč je podporován specifickou implementací WebDriveru. Tato implementace se nazývá ovladač (driver). Jedná se o komponentu, která je odpovědná za delegování úkolů na prohlížeč a zajišťuje komunikaci mezi prohlížečem a Seleniem.

Pro zautomatizování prohlížeče potřebujeme nainstalovat dvě věci. První je instalace Selenium knihovny. Druhá je požadovaný ovladač prohlížeče. Zde je seznam aktuálně podporovaných ovladačů: [30]

- Chrome – ChromeDriver
- Firefox – GeckoDriver
- Edge – EdgeDriver
- Internet Explorer – InternetExplorerDriver

V této práci bude použit ovladač ChromeDriver. Základní možnosti tohoto ovladače budou představeny v samostatné kapitole.

Na obrázku je zobrazena zjednodušená architektura Selenium Webdriver nástroje. Architektura WebDriver podporuje řadu jazyků. Díky tomu si testeři mohou vybrat, jaký programovací jazyk chtějí použít. Stačí pouze stáhnout potřebné knihovny. JSON je zkratka pro JavaScript Object Notation. Jde o standard, který poskytuje mechanismus pro přenos dat mezi klientem a serverem. Jak již bylo zmíněno, každý prohlížeč má svoji implementaci ovladače. Tyto ovladače skrývají implementační logiku fungování prohlížeče před koncovým uživatelem. Právě JSON Wire protocol vytváří spojení mezi ovladači prohlížeče a uživatelskými knihovnami. Testy je možné spouštět pouze tehdy, jsou-li potřebné prohlížeče nainstalovány, ať lokálně, nebo na serveru, kde testy běží.



Obrázek 6: WebDriver Architecture

Zdroj: Web [38]

5.3.3 Selenium Grid

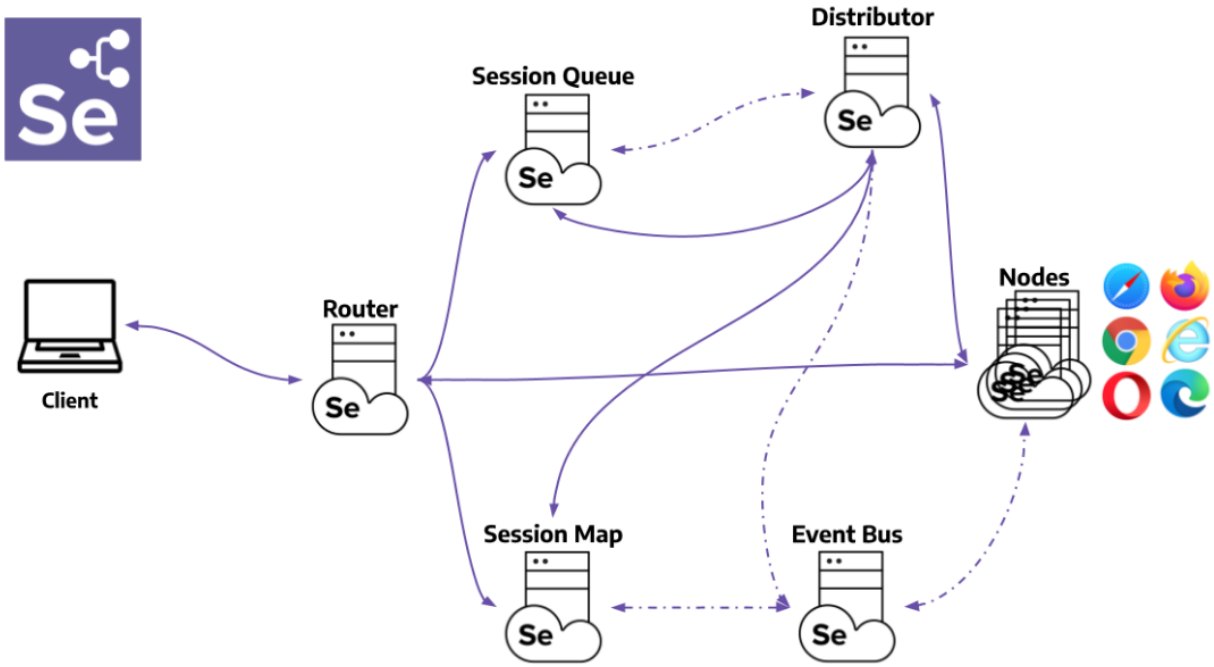
Hlavním cílem Selenium Grid je spouštět testy paralelně na více strojích. Tento nástroj umožňuje spouštět WebDriver testovací skripty na vzdálených počítačích, jak fyzických, tak i virtuálních, pomocí směrování příkazů odeslaných klientem na vzdálené instance prohlížečů. Obecně lze uvést dva hlavní důvody, proč využít právě Selenium Grid: [39]

- Chceme-li spustit testy proti více prohlížečům, s odlišnými verzemi, a na jiných operačních systémech.
- Zkrátit dobu, kterou testovací sada potřebuje k dokončení všech testů.

Díky populárním agilním vývojovým metodikám software je žádoucí, aby byly testy co nejrychlejší a tým tak měl zpětnou vazbu co nejdříve. Pokud například máme testovací sadu, která obsahuje 100 testů, můžeme nastavit Grid tak, aby tyto testy běžely na 4 různých strojích. Na každém stroji poběží paralelně 25 testů a testovací čas se tak podstatně zkrátí.

Jak bylo uvedeno výše, je také možné nastavit grid tak, aby testy běžely proti různým prohlížečům. Selenium Grid přiřadí jednotlivé testy k odpovídajícímu prohlížeči. Jedná se o velice flexibilní nástroj, u kterého ale může být docílení

požadovaného nastavení komplikovaným úkolem, který vyžaduje odbornější znalosti a zkušenosti.



Obrázek 7: Selenium Grid

Zdroj: Web [40]

Na obrázku výše je uvedeno, z jakých částí se Selenium Grid skládá. Níže jsou vysvětleny základní komponenty. [40]

Router (směrovač) se stará o předání požadavku správné komponentě. Přijímá veškeré vnější požadavky, je tak vstupním bodem do sítě. Za cíl má také vyrovnávat zátěž v síti tím, že posílá požadavky prvku, který je schopen jej nejlépe zpracovat.

Node (uzel) se může v síti vyskytovat několikrát. Každý node se stará o správu slotů pro dostupné prohlížeče na stroji, na kterém je spuštěn. Také provádí přijaté příkazy.

Distributor ví o všech uzlech a zná také jejich možnosti. Jeho hlavním úkolem je přijmout požadavek na novou relaci a najít vhodný uzel, na kterém půjde vytvořit. Po jejím vytvoření uloží do Session Map vztah mezi ID relace a uzlem, kde je relace prováděna.

New Session Queue (fronta nových relací) obsahuje všechny nové požadavky na relaci v pořadí FIFO. Router přidá nový požadavek na relaci do fronty a čeká na odpověď. Fronta nových relací pravidelně kontroluje, zda u některého požadavku nevypršel časový limit. Pokud ano, je tento požadavek odmítnut a okamžitě odstraněn. Distributor pravidelně kontroluje, zda je volný slot. Pokud ano, tak požádá frontu nových relací o první odpovídající požadavek.

Event Bus slouží jako komunikační cesta mezi ostatními komponenty. Většinu své interní komunikace provádí grid prostřednictvím zpráv, díky čemu se vyhýbá volání http. Při spuštění gridu je Event Bus první komponentou, která by měla být spuštěna.

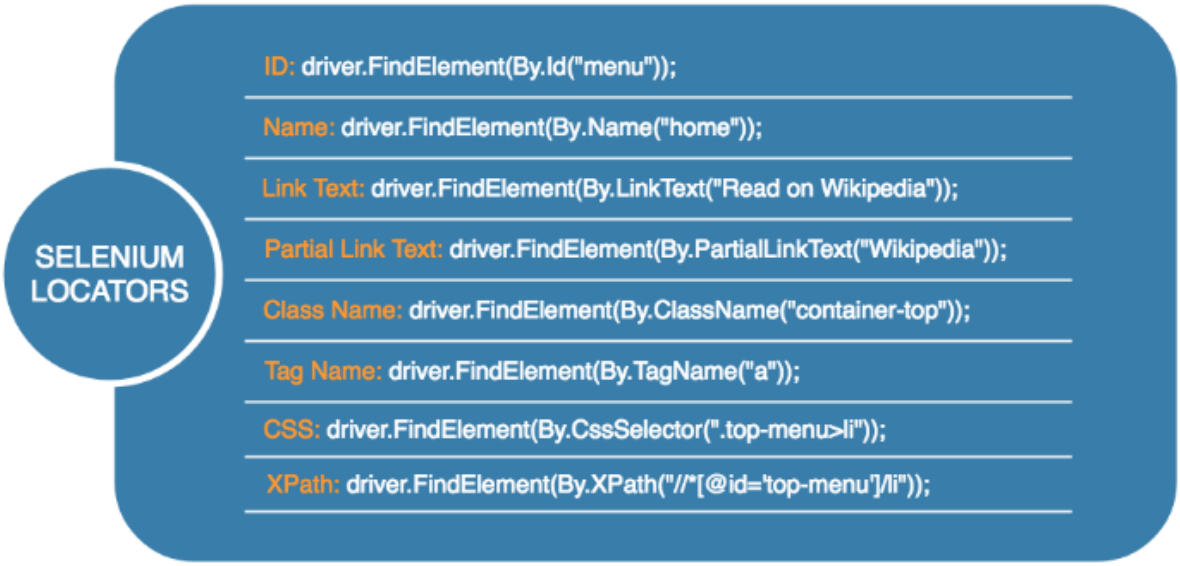
5.4 Webové elementy

Jako webový element můžeme označit cokoliv, co se nachází na webové stránce. Jedná se o prvek HTML. Jako základní předpoklad pro práci se Selenium je získat reference na webové elementy na stránce, a dále s nimi pracovat. Selenium nabízí řadu vestavěných lokátorů, jejichž pomocí můžeme prvky na stránce jednoznačně identifikovat. [41]

Mnoho lokátorů odpovídá více prvkům na stránce. Metoda findElement() vrátí první nalezený prvek, který odpovídá zadané cestě. Seznam níže zobrazuje, podle jakých lokátorů je možné elementy na stránce hledat: [42]

- Id
- CssSelector
- XPath
- ClassName
- Name
- LinkText
- PartialLinkText
- TagName

Všechny lokátory, včetně příkladů, jsou uvedeny na následujícím obrázku.



Obrázek 8: Selenium Lokátory

Zdroj: Web [43]

5.4.1 Interakce s webovými elementy

Nad prvkem, elementem, webové stránky lze provést celkem 3 základní příkazy: [44]

1. Click – zajistí kliknutí na element přesně v jeho středu. Funguje na všechny typy elementů. Aby bylo kliknutí na element úspěšné, nesmí být blokován žádným jiným prvkem.
2. SendKeys – pošle to zvoleného elementu text, který se vkládá jako parametr této funkce. Platí pouze pro textová pole a prvky s editovatelným obsahem.
3. Clear – tento příkaz resetuje obsah prvku. Stejně jako u předchozího příkazu, musí být prvek textové nebo mít editovatelný obsah.

Na dalším obrázku jsou uvedeny příklady těchto základních interakcí v programovacím jazyce C#.

```
// Klikne na odpovidajici element
driver.FindElement(By.Name("element-name")).Click();

// Vlozi  text "Selenium-text" do odpovidajici elementu
driver.FindElement(By.Name("element-name")).SendKeys("selenium-text");

// Vymaze obsah v danem elementu
driver.FindElement(By.Name("element-name")).Clear();
```

Obrázek 9: Základní příkazy nad webovými elementy

Zdroj: Vlastní zpracování

5.4.2 Viditelnost webových elementů

Viditelnost elementů na stránce patří k základním problémům, které je nutné při práci se Seleniem řešit. Hlavní problém je v tom, že jsou testy většinou vykonávané větší rychlostí, než je stránka načítána. To má za následek ukončení testů s vyjímkou *NoSuchElementException*, která značí, že test nedokázal lokalizovat požadovaný element na stránce.

Selenium WebDriver tento problém řeší tím, že poskytuje tzv. *wait* akce. Díky tomu je možné počkat požadovanou dobu, než bude test pokračovat v dalším příkazu. Stránka se tak stihne načíst a spolu s ní i všechny její elementy. Existují dva typy těchto akcí – explicitní a implicitní. [45]

Implicitní čekání spočívá v pevně nastaveném čase, po který bude test čekat, než bude pokračovat dále. V příkladu níže je uvedeno nastavení čekání po dobu deseti sekund, než se WebDriver pokusí element najít.

```
driver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(10);
```

Pokud se element nenačte ani po stanoveném čase, test bude opět neúspěšně ukončen.

Druhou možností je explicitní čekání. Tento typ čekání umožňuje navíc čekat pouze do doby, kdy je splněna námi zadaná podmínka (Expected Condition). Mezi tyto základní podmínky patří: [46]

- ElementExists – kontroluje, zda je prvek přítomen v modelu stránky. Nemusí to nutně znamenat, že je prvek viditelný.

- ElementIsVisible – také kontroluje, zda je prvek přítomen v modelu stránky a navíc, zda je viditelný. Viditelnost elementu znamená, že musí být na stránce zobrazen a mít šířku a výšku větší než 0.
- ElementToBeClickable(By) – kontroluje, zda je prvek viditelný, a lze na něm provést kliknutí myší.
- TitleIs – kontroluje název zobrazené stránky
- TextToBePresentInElement – kontrola, zda je daný text přítomen v zadaném elementu.

Celkem existuje přes 30 těchto podmínek. Výše uvedené jsou pouze základní z nich. Na dalším obrázku je uveden příklad explicitního čekání, kdy test bude čekat, dokud nebude námi zvolený element viditelný na stránce. Pokud tato podmínka do 10 sekund nenastane, test skončí chybou.

```
// Doba cekani maximalne 10 sekund
WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(10));
// Test bude pokracovat, pokud bude element na strance viditelny
var element = wait.Until(SeleniumExtras.WaitHelpers.ExpectedConditions.ElementIsVisible(By.Id("id-123")));
```

Obrázek 10: Explicit Wait

Zdroj: Vlastní zpracování

Díky tomu, že test v případě splnění podmínky u explicitního čekání ihned pokračuje, je doporučeno používat právě tento typ.

6 Využívané technologie pro testování

V této kapitole je proveden přehled technologií, které jsou použity při vytváření a spouštění automatických testů grafického rozhraní pro webovou aplikaci, která je ve firmě, ve které pracuji, vyvíjena. Z právních důvodů nelze uvést o jakou firmu se přesně jedná. Aby nebyl porušen Zákon o ochraně duševního vlastnictví, bylo nutné z ukázek kódu odstranit citlivé informace. Tyto informace jsou nahrazeny obecnější variantou, která se pro správné fungování testů musí vždy změnit na konkrétní data. Tyto změněné údaje jsou v textu vyznačeny kurzívou. To stejné platí pro kapitolu číslo 7.

6.1 Microsoft Visual Studio

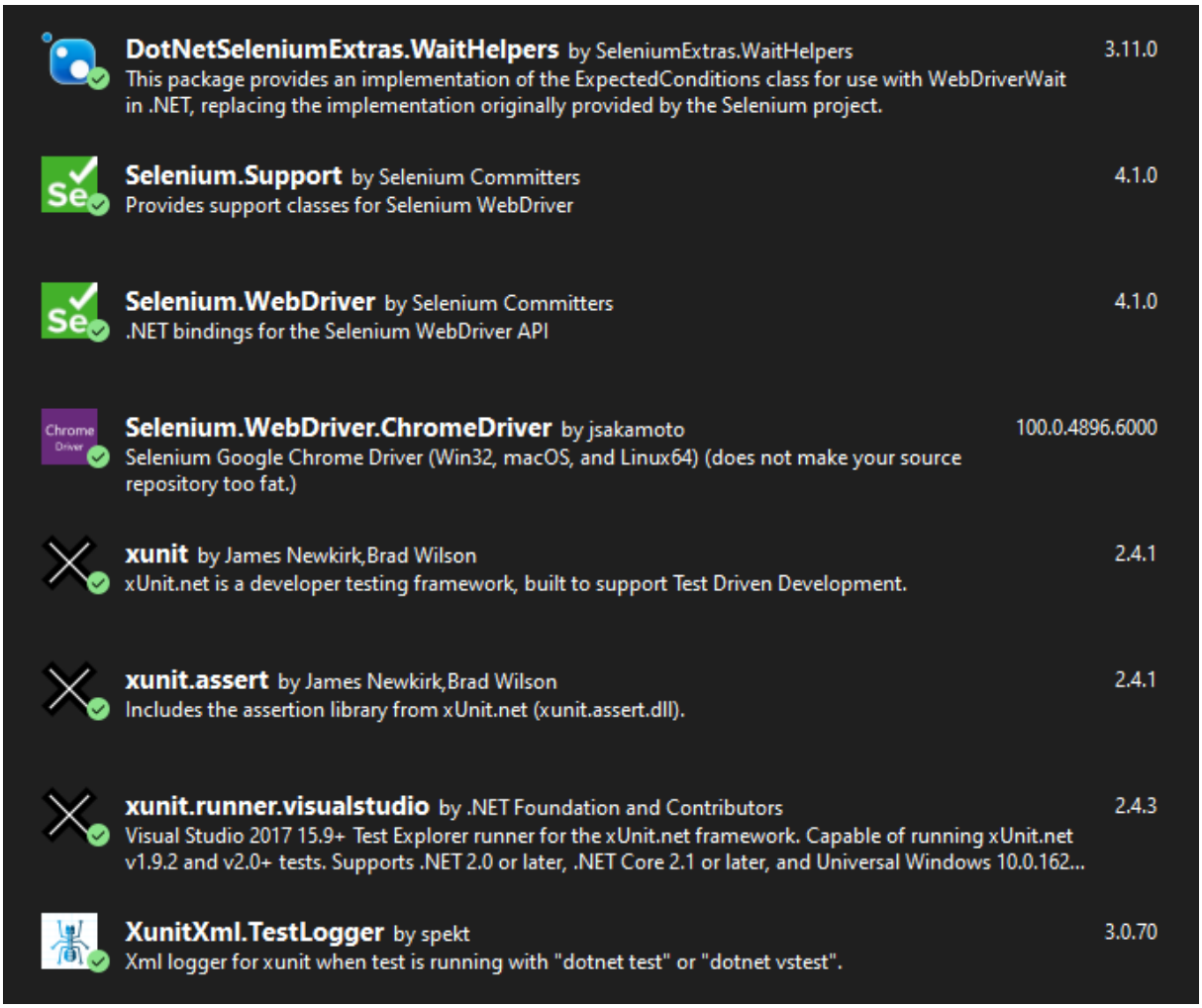
Nástroje pro vývoj testů můžeme rozdělit do dvou kategorií podle jejich zaměření – komplexní vývojová prostředí, tzv. IDE (Integrated development environment) a jednodušší editory. Hlavní rozdíly jsou v ceně, systémových požadavcích, výkonu a možnostech, které jednotlivá prostředí nabízí.

V našem případě hlavním nástrojem pro vytváření automatických testů je vývojové prostředí Visual Studio. Za jeho vývojem stojí společnost Microsoft a první verze byla vydána v roce 1997. [47] Jedná se komplexní prostředí, které slouží pro vývoj aplikací na různých platformách, zejména .NET framework. Výhoda Visual Studia je velká nabídka dostupných rozšíření a nástrojů, které se dají stáhnout z oficiálních stránek [48] a uživatel si tak může možnosti tohoto nástroje velice rozšířit.

Tento rok vyšla nová verze pojmenovaná Visual Studio 2022, která přináší mnohá vylepšení oproti starším verzím, lepší stabilitu, práci s pamětí, rychlost, vylepšené navrhování řádků kódu a další. Tato verze je nabízena ve třech edicích – Community, Professional a Enterprise. První zmíněná základní verze je k dispozici zdarma ke stažení, zbylé dvě jsou za poplatek v podobě měsíčního předplatného. Podrobnější porovnání jednotlivých verzí je k dispozici na oficiálních stránkách. [49]

Do projektu je možné ve Visual Studiu stahovat tzv. NuGet balíčky. V zásadě se jedná o soubor, který obsahuje zkompilovaný kód, který vytvořil jiný vývojář a tímto

způsobem ho zpřístupnil k použití ostatním uživatelům. [50] Díky tomu tak není nutné vše vymýšlet znovu, ale využít řešení, které někdo jiný už vyvinul a s tím související třídy a metody. Výhodou je také neustálé vydávání nových verzí jednotlivých balíčků, které přináší opravy a vylepšení předchozích verzí. Obrázek níže ukazuje všechny balíčky, které jsou v projektu pro vývoj automatických Selenium testů použity.



Obrázek 11: Visual Studio - NuGet Balíčky

Zdroj: Vlastní zpracování

6.1.1 Xunit

V rámci automatických testů je použit nástroj XUnit. Jedná se o open source testovací framework jehož autorem je tvůrce jiného testovací frameworku, Jim Newkirk. [51]

V době psaní této práce je aktuální verze 2.4.1. [52] Tento framework se snaží o minimalistický přístup. Při spuštění testů jsou procházeny všechny veřejné třídy a v nich hledány metody s atributem [Fact] nebo [Theory]. Tyto atributy označují danou metodu jako test.

Atribut [Fact] označuje metodu, která nemá žádné argumenty na vstupu. Jedná se o klasický test, kdy vše musí být uvedeno v dané metodě.

```
[Fact]
public void ValidLogin()
{
test code
}
```

Oproti tomu atribut [Theory] je u metod použit, pokud chceme do testu vložit konkrétní data. Tento atribut sám o sobě nestačí a je nutné s ním použít další atribut [InlineData]. Ten se používá k zadání podmnožiny dat, na jejichž základě budou provedeny parametrizované testy.

```
[Theory(DisplayName = "Zalozeni objednavky")]
[InlineData("Petr", "Novák", "2022-03-15", "Společnost s.r.o.")]
public void ZalozeniObjednavky(string jmenoDodavatele, string
prijmeniDodavatele, string datumObjednavky, string
nazevSpolecnosti)
        {
           test code
        }
```

V ukázce kódu je také možné vidět parametr DisplayName při definování atributu Theory. Ten umožňuje definovat přátelštější název pro testovací metodu, který je poté zobrazen v Test exploreru Visual Studia. Příklad tohoto zobrazení je možné vidět v kapitole 7.3, obrázek 16.

6.2 GitLab

Automatické testy jsou spouštěny v pipelinách GitLabu, vždy po nasazení nové verze na testovací prostředí. Na tomto prostředí jsou spouštěny všechny automatické testy, jak pro frontend, tak i pro backend. Při vývoji aplikace jsou celkem použity čtyři prostředí:

1. Develop – prostředí sloužící hlavně pro programátory k vývoji nových funkcionalit. Občas zde probíhá i manuální testování.
2. Test – zde jsou spouštěny automatické testy pro frontend a backend.
3. Stag – toto prostředí slouží pro uživatelské akceptační testování.
4. Prod – prostředí, na kterém aplikace běží k ostrému používání. Na toto prostředí by se, díky všem předchozím testům, neměly dostat žádné chyby, popř. co nejmenší množství.

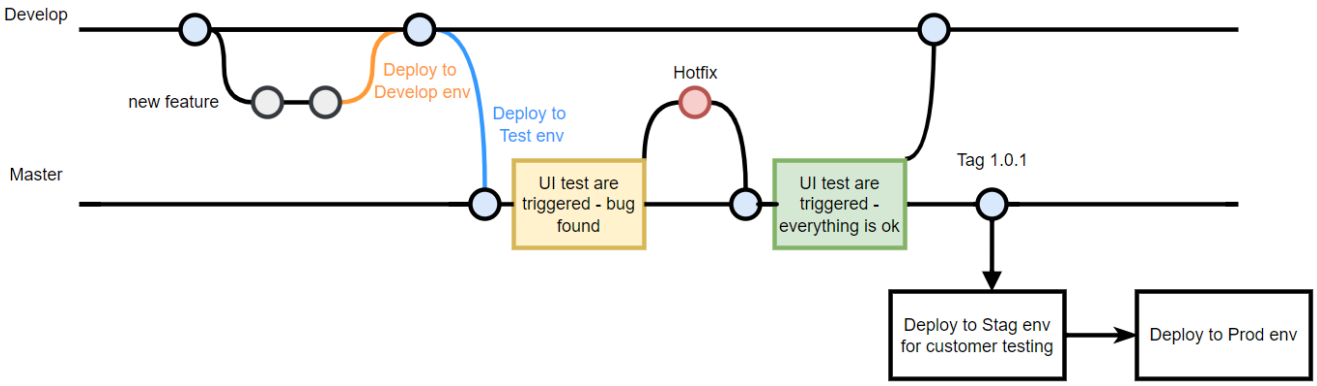
Na jakém prostředí se mají testy spouštět, je nastaveno pomocí .gitlab-ci.yml souboru. Tento soubor slouží k nastavení jednotlivých pipeline uvnitř GitLabu. Příklad tohoto konrétního souboru, pro nastavení automatického spouštění Selenium testů, je uveden níže.

```
stage: qa-test
  image: mcr.microsoft.com/dotnet/sdk
rules:
  - if: $CI_COMMIT_REF_NAME == "master"
      when: on_success
  - when: never
script:
  - dotnet test tests/uitest/Selenium_tests.csproj --logger html
artifacts:
    when: always
    paths:
       - ./tests/uitest/TestResults
    expire_in: 1 week
```

Tento soubor zaručí spuštění automatických testů, pokud je název větve, na které se provádí nasazení nové verze aplikace, master. Na této větvi se nachází testovací prostředí. U ostatních větví se tento job vůbec nespustí. Dále je možné vidět nastavení artefaktů z tohoto jobu. Artefakt je soubor, který vznikne z výstupu dané úlohy. V tom případě se jedná o html report výsledku testů, který je možné vidět v kapitole 7.3, obrázek 17.

Obecný diagram řešení, kdy jsou spouštěny automatické UI testy, je ukázán níže. Jak již bylo řečeno, testy se spouští automaticky při nasazení nové verze do master větve, kde je také nastaveno testovací prostředí. Pokud testy odhalí chybu, je vývojáři vytvořen hotfix a následně jsou testy spuštěny znovu. Pokud je vše v pořádku, je vytvořen nový tag, který zajistí automatické nasazení na stag prostředí k zákaznickému testování. Zákazníci mají určitý časový úsek, v řádu několika dnů,

na testování. Pokud potvrdí, že vše funguje, je verze ručně nasazena na produkční prostředí k oficiálnímu používání.



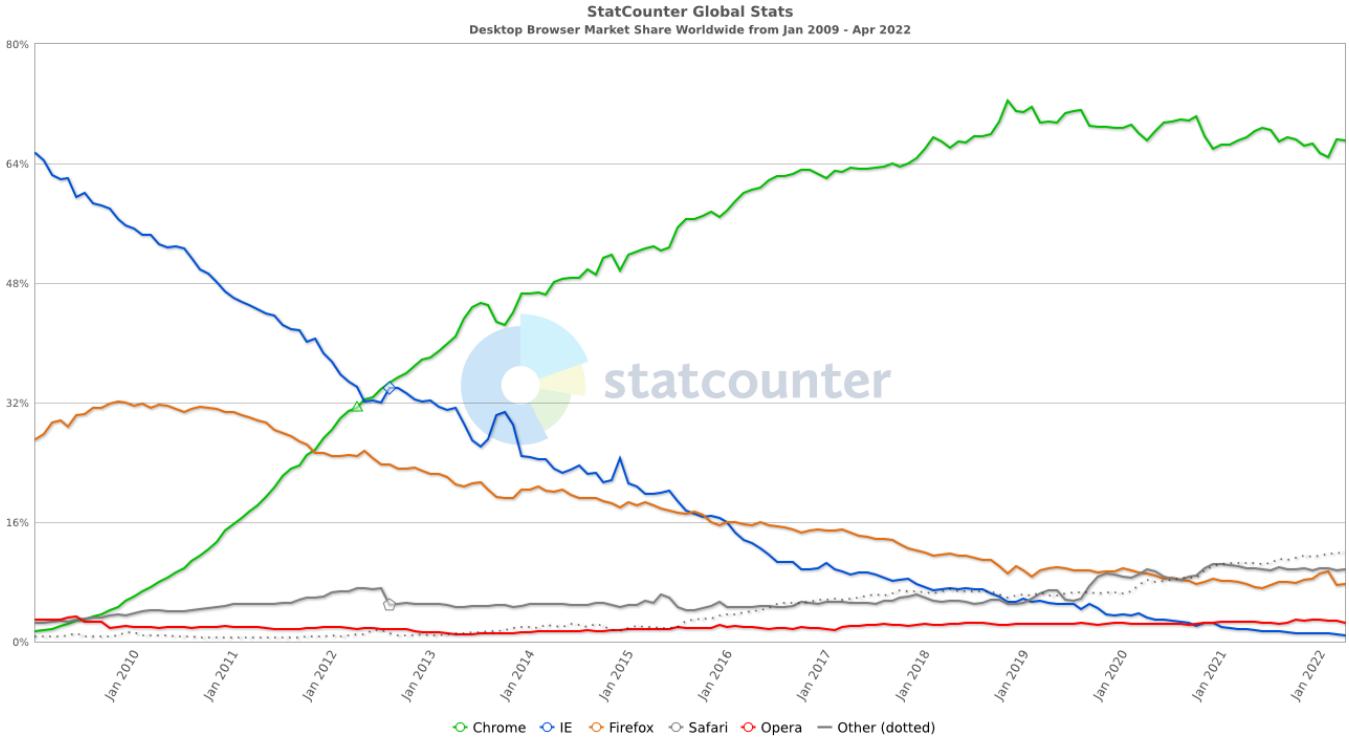
Obrázek 12: GitLab – Spouštění Automatických UI Testů

Zdroj: Vlastní zpracování

6.3 Google Chrome

Jelikož je aplikace vyvíjena převážně pro prohlížeč Google Chrome, je jeho používání každodenní činností. Může se jednat jak o manuální testování nových funkcionalit, tak i o zjišťování lokátorů pro Selenium testy u jednotlivých webových elementů.

Tento webový prohlížeč vydává společnost Google. První verze byla vydána v roce 2008. Jedná se o multiplaformní prohlížeč, který je k dispozici na Windows, Mac, Linux a Android. Jak je možné vidět na grafu níže, Chrome má největší podíl na trhu všech prohlížečů na světě na osobních počítačích. V dubnu 2022 tento podíl činí 67 %. Na druhém místě je prohlížeč Safari, který má podíl 9,74 %.



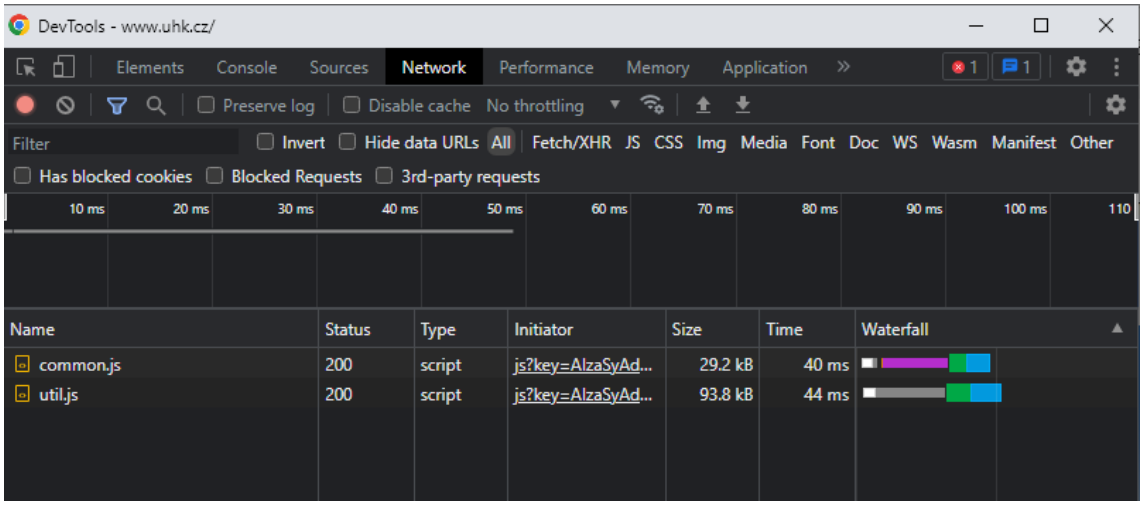
Obrázek 13: Celosvětový podíl webových prohlížečů na PC

Zdroj: [53]

6.3.1 DevTools

Pro vývoj automatických testů je nejdůležitější nástroj DevTool. Jedná se o sadu nástrojů integrovanou přímo do prohlížeče Google Chrome. Existuje několik způsobů, jak tento nástroj spustit. Nejjednodušší je použití klávesy F12. Další způsob je otevřít menu s možnostmi v pravém horním rohu prohlížeče, rozbalit nabídku More Tools a zde vybrat Developer tools. Jak vypadá spuštěný nástroj je zobrazeno na obrázku níže.

Při vytváření automatických UI testů je nejvíce využívaná záložka Elements, která zobrazuje veškeré elementy webové stránky. Zde se získávají lokátory pro tyto elementy, který jsou následně součástí testovacího skriptu.



Obrázek 14: Chrome browser - Dev tool

Zdroj: Vlastní zpracování

Na obrázku výše můžeme vidět několik záložek, kdy každá z nich je určená k jiným operacím. Důležité záložky z levé strany jsou: [54]

- Elements – zobrazení CSS a elementů stránky
- Console – zobrazení JavaScriptových zpráv
- Network – zobrazení síťové aktivity
- Performance – zobrazení rychlosti načítání webové stránky
- Application – zobrazí všechny načtené prostředky, místní úložiště, úložiště relace, cookies soubory a mezipaměť aplikací

7 Realizace testů

V této kapitole budou ukázány a popsány konkrétní příklady kódu pěti testů. Hlavní účel testované aplikace je objednávání a správa fyzického materiálu, který je použit při výrobě firemních produktů. Argumenty jednotlivých testů jsou anonymizovány obecnějšími variantami. Zároveň byly takto upraveny i některé webové elementy. Všechny takto upravené údaje jsou v ukázkách vyznačeny kurzívou a pro správné fungování testů je zapotřebí místo nich zadat konkrétní hodnoty.

7.1 Testovací případ

Před začátkem tvoření každého testu je nutné promyslet, kterou funkcionalitu webové aplikace budeme chtít novým testem pokrýt a poté naplánovat, které kroky budou v testu obsaženy, abychom docílili požadovaného stupně otestování. K tomuto účelu pomohou tzv. testovací případy. Jak již bylo v této práci uvedeno, je možné použít například nástroj Selenium IDE. Ne vždy je ale tato možnost nejrychlejší. Pokud má tester dobrou znalost testované aplikace, může testovací případ vytvořit ručně. Přesná forma testovacích případů není předepsána a v každé firmě mají rozdílnou podobu, od excelovských tabulek po textovou podobu v programu Word.

Zde uvedu příklad testovacího případu, který je dále převeden do spustitelného kódu pro automatický test. Tento testovací případ je vytvořen v programu Excel. Obsahuje unikátní ID testovacího případu, autora, datum vytvoření, krátký popis a postupné kroky, které musí uživatel provést, aby byl celý test dokončen.

Test Case ID	Test-12345
Created By	Adam Kucera
Test Created	March 25, 2022
Test Scenario	Ověřit, že lze založit novou objednávku materiálu po vyplnění povinných polí.

Step #	Step Details	Expected Results	Test Data
1	Otevřít stránku pro přihlášení	Zobrazení přihlašovacího formuláře	
2	Zadat platné uživatelské jméno a heslo	Zadány odpovídající údaje	*username, password*
3	Kliknout na tlačítko "Přihllásit se"	Úspěšné přihlášení uživatele	
4	Vyhledat požadovaný materiál	Zobrazení detailu zadaného materiálu	*material*
5	Kliknout na tlačítko "Objednat materiál"	Zobrazení formuláře pro objednání materiálu	
6	Do pole "Qty" zadat množství materiálu	Pole obsahuje požadovaný údaj	*quantity*
7	Do pole "Order ID" zadat identifikátor objednávky	Pole obsahuje požadovaný údaj	*identifier*
8	Do pole "Delivery Date" zadat požadované datum doručení	Pole obsahuje požadovaný údaj	*deliveryDate*
9	Kliknout na tlačítko "Vytvořit objednávku"	Zobrazen text "Material order created."	

Obrázek 15: Testovací případ

Zdroj: Vlastní zpracování

7.2 Testovací skript

Projekt obsahuje celkem 3 samostatné třídy - SeleniumConfig, Steps a Tests. Třída Tests.cs obsahuje všechny testy, které se budou spouštět. Tyto testy se skládají pomocí jednotlivých kroků, které jsou obsaženy ve třídě Steps.cs. To umožňuje jednodušší správu testů po menších částech a je snazší tyto změny provádět. Vždy by mělo stačit změnit pouze část kódu v jednotlivém kroku a ne celý test. Tato změna by neměla mít vliv na ostatní kroky, ze kterých jsou dané testy složeny.

Třída SeleniumConfig je popsána níže. Veškeré testy jsou zapouzdřeny v try / finally bloku. V bloku try je samotný test. V bloku finally je volána metoda Quit(), která zavře okno prohlížeče a ukončí danou testovací relaci. Je tak zaručeno, že se container vždy uvolní pro další test a nebude blokován předchozím.

Každý test se skládá z více menších kroků. Zde jako příklad vždy uvedu pouze jeden krok pro každý test. Ostatní kroky jsou okomentovány přímo v testovacím sciptu, který je přiložen jako příloha této práce.

7.2.1 Základní konfigurace testů

Základní třída, kterou pro testy potřebujeme, je třída SeleniumConfig.cs. V této třídě se provádí veškeré základní nastavení, které se týká prohlížeče použitého pro

testování. K tomu slouží vytvořená instance třídy ChromeOptions, která obsahuje vhodné metody pro specifické nastavení pro prohlížeč Chrome. Aby se tato nastavení projevila, je zapotřebí přidat objekt ChromeOptions do konstruktoru WebDriveru.

Níže jsou vidět dva parametry pro nastavení. První z nich je parametr *headless*. Díky tomu je webový prohlížeč spuštěn bez grafického rozhraní. Je vhodné ho použít, pokud se testy pouští na serveru, kde není toto grafické rozhraní vidět, tudíž ani není potřeba. Díky tomu se také ušetří omezené zdroje serveru, jako je například operační paměť, či místo na disku. Pokud se provádí lokální vývoj nových testů, je vhodnější toto nastavení nepoužívat, protože není k dispozici video o průběhu testu. To může ztížit případné hledání důvodu, proč je daný test nefunkční a tvůrce testu se musí řídit pouze podle logů.

Jako další parametr je použité nastavení *start maximized*, kdy už podle názvu je jasné, že se prohlížeč automaticky spustí v maximální velikosti okna, a ne pouze ve zmenšené podobě. Celkem existuje asi kolem 140 těchto parametrů pro nastavení. [55]

Ve skriptu je také proměnná hubAdress, která obsahuje adresu, na kterou se mají testy připojovat. Pro lokální vývoj a testování jsou přesměrovány na port 4444, na kterém je vystaven běžící lokální docker container obsahující selenium grid. Pro spuštění testů na serveru, například v Kubernetes clutestu, je nutné tuto adresu změnit na konkrétní odpovídající ip adresu.

```
 public class SeleniumConfig
    {

        String hubAdress = "http://localhost:4444";
        ChromeOptions options = new ChromeOptions();

        public Zalenium()
        {}

        public void PrepareDriver(string name = "Selenium Test")
        {
            options.AddArguments("--headless");
            options.AddArguments("--start-maximized");
            remoteDriver = new RemoteWebDriver(new
Uri(hubAdress), options);
    }
```

7.2.2 Přihlášení uživatele

Skript pro otestování úspěšného přihlášení uživatele do aplikace. V námi vytvořené třídě Steps.cs je vytvořena metoda Login, která slouží k otestování úspěšného přihlášení. Zároveň se používá ve všech ostatních testech, protože každá instance webového prohlížeče vyžaduje nové přihlášení do aplikace. Z tohoto důvodu nejsou přihlašovací jméno a heslo uvedeny jako parametry, jelikož se nepředpokládá, že se v jednotlivých testech budou používat odlišné hodnoty.

```
public void Login(RemoteWebDriver driver)
    {
      var wait = new WebDriverWait(driver, new TimeSpan(0, 0,
10));
       wait.Until(SeleniumExtras.WaitHelpers.ExpectedConditions.
ElementToBeClickablBy.XPath("//button/span[text()='    Sign    In
']"))).Click();
      driver.FindElement(By.XPath("//span[text()=Company
Users']")).Click();
      driver.FindElement(By.Id("userNameInput")).SendKeys
("username");
      driver.FindElement(By.Id("passwordInput")).SendKeys
("password");
      driver.FindElement(By.Id("submitButton")).Click();
    }
```

Pokud bychom chtěli mít jistotu, že přihlášení uživatele proběhlo úspěšně, můžeme přidat například níže uvedený kus kódu, který kontroluje podmínku, zda je správné uživatelské jméno zobrazeno v požadovaném elementu. Tento element je zobrazen pouze po úspěšném přihlášení.

```
wait.Until(SeleniumExtras.WaitHelpers.ExpectedConditions.TextToB
ePresentInElementLocated(By.XPath("//button[3]/span[1]"),
"username"));
```

7.2.3 Objednání materiálu do výroby

Test „MaterialOrder_RequiredFields“ kontroluje správné zadání údajů do jednotlivých polí formuláře pro objednání fyzického materiálu, který je po dodání použit ve výrobě. Ve formuláři se vyskytuje více polí. Tento test kontroluje pouze zadání povinných polí, aby šel materiál objednat. Test přijímá parametry material, supplier, numberOfWeeks, view, quantity, identifier, deliveryDate. Testovací

framework XUnit poskytuje atribut InlineData, který do našeho testu předá konkrétní, námi poskytnuté, hodnoty. Pořadí hodnot v tomto atributu odpovídá pořadí, v jakém jsou parametry deklarovány ve funkci, resp. testu.

Dále deklarujeme instanci z námi vytvořené třídy Steps. Díky tomu můžeme přistupovat k metodám této třídy, které tvoří jednotlivé kroky testu. Jak je patrné z ukázky níže, je zde celkem pět kroků – Login, Dashboard_SearchMaterial, EnterMaterialOrder_RequiredFields, SearchForMaterialOrder a DeleteMaterialOrder. Z názvu kroků by mělo na první pohled být jasné, co který dělá.

```
[Theory(DisplayName = "Material order (Required fields)")]
[InlineData("material", "supplier", "numberOfWeeks", "view",
"quantity", "identifier", "deliveryDate")]
public void MaterialOrder_RequiredFields(string material, string
supplier, string numberOfWeeks, string view, string quantity,
string identifier, string deliveryDate)
        {
            try
            {
                Steps step = new Steps();
                fix.remoteDriver.Navigate().GoToUrl(url);
                step.Login(fix.remoteDriver);
                step.Dashboard_SearchMaterial(fix.remoteDriver,
material, supplier, numberOfWeeks, view);
                step.EnterMaterialOrder_RequiredFields(fix.remot
eDriver, quantity, identifier, deliveryDate);
                step.SearchForMaterialOrder(fix.remoteDriver,
identifier);
                step.DeleteMaterialOrder(fix.remoteDriver);
            }
            finally
            {
                fix.remoteDriver.Quit();
            }
        }
```

Na příkladu níže je ukázka jednoho kroku testu, a to konkrétně kroku „EnterMaterialOrder_RequiredFields“. V první fázi testu deklarujeme instanci třídy WebDriverWait. Konstruktor této třídy vyžaduje dva parametry - na jaký driver se má čekání použít a jak dlouho se bude na jednotlivé elementy čekat. V tomto případě se bude vyčkávat maximálně 10 sekund. Pokud do této doby nebude splněna konkrétní podmínka, test skončí chybou. Zde musí být splněna podmínka, aby bylo možné během této doby kliknout na vybraný element.

Postupně se do jednotlivých elementů formuláře posílají hodnoty. Po zadání hodnot je kliknuto na tlačítko pro založení objednávky materiálu. Na konci tohoto kroku je provedena kontrola úspěšného založení objednávky pomocí zobrazeného textu, který musí být „Material order created.“.

```
public void EnterMaterialOrder_RequiredFields(RemoteWebDriver
driver, string quantity, string identifier, string deliveryDate)
        {
            var wait = new WebDriverWait(driver, new TimeSpan(0,
0, 10));
            wait.Until(SeleniumExtras.WaitHelpers.ExpectedCondit
ions.ElementToBeClickable(By.CssSelector("mat-icon[aria-
label='Open new order form']"))).Click();
            wait.Until(SeleniumExtras.WaitHelpers.ExpectedCondit
ions.ElementToBeClickable(By.XPath("//input[@placeholder='Qty']"
))).Click();
            wait.Until(SeleniumExtras.WaitHelpers.ExpectedCondit
ions.ElementToBeClickable(By.XPath("//input[@placeholder='Qty']"
))).SendKeys(quantity);
            wait.Until(SeleniumExtras.WaitHelpers.ExpectedCondit
ions.ElementToBeClickable(By.XPath("//input[@placeholder=Order
id']"))).SendKeys(identifier);
            wait.Until(SeleniumExtras.WaitHelpers.ExpectedCondit
ions.ElementToBeClickable(By.XPath("//input[@placeholder='Delive
ry date']"))).Click();
            Actions action = new Actions(driver);
            action.SendKeys(Keys.Escape).Perform();
            wait.Until(SeleniumExtras.WaitHelpers.ExpectedCondit
ions.ElementToBeClickable(By.XPath("//input[@placeholder='Delive
ry date']"))).SendKeys(deliveryDate);
            wait.Until(SeleniumExtras.WaitHelpers.ExpectedCondit
ions.ElementToBeClickable(By.Id("submit"))).Click();

            // kontrola zobrazene hlasky po vytvoreni nove
objednavky
            wait.Until(SeleniumExtras.WaitHelpers.ExpectedCondit
ions.ElementIsVisible(By.XPath("//span[contains(text(),'Material
order created.')]")));
        }
```

7.2.4 Vzorec pro výpočet počtu kusů materiálu

Test „CreateCalculationFormula“ má za cíl otestovat správné založení vzorce pro výpočet počtu kusů materiálu na skladě v jednotlivých týdnech. Tento vzorec se poté používá v aplikaci na více místech. Nicméně to není předmětem tohoto testu, zde se ověřuje pouze jeho základní založení a následné smazání.

```
[Theory(DisplayName = "Calculation formula - Create, Search,
Delete")]
[InlineData("view", "Selenium - Calculation formula")]
public void CreateCalculationFormula(string view, string
nameOfCalculationFormula)
        {
            try
            {
                Steps step = new Steps();
                fix.remoteDriver.Navigate().GoToUrl(url);
                step.Login(fix.remoteDriver);
                step.Dashboard_SelectViewModule(fix.remoteDriver
);
                step.OpenConcreteView(fix.remoteDriver, view);
                step.View_CreateCalculationFormula(fix.remoteDri
ver, nameOfCalculationFormula);
                step.View_SearchForSpecificCalculationFormula(fi
x.remoteDriver, nameOfCalculationFormula);
                step.View_DeleteCalculationFormula(fix.remoteDri
ver, nameOfCalculationFormula);
            }
            finally
            {
                fix.remoteDriver.Quit();
            }}
```

Tento test se skládá ze šesti kroků. Prvním je krok „Login“, který je u všech testů stejný. Zajišťuje přihlášení uživatele do aplikace. Druhý krok „Dashboard_SelectViewModule“ zajišťuje vybrání odpovídající modulu aplikace, který slouží k nastavení vzorců. Třetí krok „OpenConcreteView“ otevře požadovaný pohled. Jaký pohled chceme otevřít, specifikujeme zadáním jeho jména do parametru funkce. Čtvrtý krok založí konkrétní vzorec pro výpočet. V pátém kroků se založený záznam vyhledá podle zadaného jména, tento krok je uveden jako příklad níže. V posledním, šestém kroku, dojde ke smazání tohoto záznamu a kontrole úspěšného smazání.

```
public void
View_SearchForSpecificCalculationFormula(RemoteWebDriver driver,
string name)
        {
            var wait = new WebDriverWait(driver, new TimeSpan(0,
0, 10));
```

```
            // search
            wait.Until(SeleniumExtras.WaitHelpers.ExpectedCondit
ions.ElementToBeClickable(By.Id("Grid_searchbar"))).SendKeys(nam
e);
            // resi problem s obcasnym mizenim vlozeneho textu
do vyhledavani
            for (int i = 0; i < 10; i++)
            {
                var enteredTextSearch =
wait.Until(SeleniumExtras.WaitHelpers.ExpectedConditions.Element
IsVisible(By.Id("Grid_searchbar"))).GetAttribute("value");

                if (enteredTextSearch != name)
                {
                    wait.Until(SeleniumExtras.WaitHelpers.Expect
edConditions.ElementToBeClickable(By.Id("Grid_searchbar")))
.Clear();
                    wait.Until(SeleniumExtras.WaitHelpers.Expect
edConditions.ElementToBeClickable(By.Id("Grid_searchbar"))).Send
Keys(name);
                    continue;
                }
                else
                {
                    break;
                }
            }
            wait.Until(SeleniumExtras.WaitHelpers.ExpectedCondit
ions.ElementIsVisible(By.CssSelector("td[aria-label= Selenium -
Calculation formula]")));
        }
```

Uvedený For cyklus řeší občasný problém s mizením vloženého textu do vyhledávacího pole, který je způsoben příliš rychlým vložením textu testovacím skriptem. Funkce GetAttribute získá text, který je vložen do searchbaru. Pokud se tento text neshoduje s názvem, který chceme vyhledat, skript pošle text to vyhledávacího pole znovu a provede znovu kontrolu, zda se nyní texty shodují. Pokud ano, pokračuje se v testu dále, pokud ne, pokusí se poslat text do pole znovu. Pokud se hledaný text nepodaří vložit během deseti pokusů, test skončí chybou. Poslední řádek kontroluje, zda byl skutečně vyhledán námi požadovaný vzorec podle jména. Pokud se jméno shoduje, dalším krokem dojde k jeho smazání.

7.2.5 Virtuální skupina materiálu

„VirtualGroup“ test má za úkol otestovat správné založení a smazání tzv. virtální skupiny. Do této skupiny se poté mohou vkládat další jednotlivé

konkrétní materiály. Prvním krokem je přihlášení do aplikace, druhým vybrání odpovídající modulu pro založení virtuální skupiny, třetím založení této skopiny a posledním krokem je její smazání.

```
[Theory(DisplayName = "Virtual group - Create, Delete")]
[InlineData("Selenium Virtual Group")]
public void VirtualGroup(string virtualGroupName)
      {
          try
          {
              Steps step = new Steps();
              fix.remoteDriver.Navigate().GoToUrl(url);
              step.Login(fix.remoteDriver);
              step.Dashboard_SelectVirtualGroupModule(fix.remote
Driver);
              step.CreateVirtualGroup(fix.remoteDriver,
virtualGroupName);
              step.DeleteVirtualGroup(fix.remoteDriver);
          }
          finally
          {
              fix.remoteDriver.Quit();
          }
      }
```

Níže je uveden příklad kroku „CreateVirtualGroup“. Skript nejdříve klikne na tlačítko pro přidání virtuální skupiny a poté do příslušného pole zadá jméno této skupiny. Po vytvoření porovná, zda se jméno skupiny shoduje s tím, které bylo při vytváření zadáno. Pokud ano, pokračuje se v testu dalším krokem.

```
public void CreateVirtualGroup(RemoteWebDriver driver, string
nameOfGroup)
        {
            var wait = new WebDriverWait(driver, new TimeSpan(0,
0, 10));
            // klikne na tlacitko pro pridani
            wait.Until(SeleniumExtras.WaitHelpers.ExpectedCondit
ions.ElementToBeClickable(By.XPath("//mat-
icon[text()='add']"))).Click();
            // jmeno virtualni skupiny
            wait.Until(SeleniumExtras.WaitHelpers.ExpectedCondit
ions.ElementToBeClickable(By.XPath("//input[@placeholder='Virtal
Group add']"))).Click();
            wait.Until(SeleniumExtras.WaitHelpers.ExpectedCondit
ions.ElementToBeClickable(By.XPath("//input[@placeholder='Insert
Name']"))).SendKeys(nameOfGroup);
```

```
            wait.Until(SeleniumExtras.WaitHelpers.ExpectedCondit
ions.ElementToBeClickable(By.XPath("/html/body/div[2]/div[2]/div
/mat-dialog-container/app-platform-add-group/form/mat-dialog-
actions/button[2]"))).Click();
            // porovnani
            wait.Until(SeleniumExtras.WaitHelpers.ExpectedCondit
ions.TextToBePresentInElementLocated(By.XPath("//div[contains
(text(),'Selenium Virtual Group')]"), nameOfGroup));
        }
```

7.2.6 Přiřazení uživatelských práv

Poslední test „UserMaterialsRights“ testuje správné přiřazení práv pro uživatele. Uživateli mohou být přiřazena práva, k jakému materiálu má přístup. K tomuto účelu slouží speciální modul, který je zde testován. Test je složen ze čtyř kroků. Jako u ostatních testů, je prvním krokem „Login“, tedy přihlášení uživatele do aplikace. Druhým krokem je vybrání modulu pro přiřazování práv. Třetím krokem dojde k založení práv pro uživatele na odpovídající materiál a posledním, čtvrtým krokem, dojde k jejich smazání.

```
[Theory(DisplayName = "User material rights - Create, Delete")]
[InlineData("userEmail", "material")]
public void UserMaterialsRights(string userEmail, string
material)
       {
           try
            {
                Steps step = new Steps();
                fix.remoteDriver.Navigate().GoToUrl(url);
                step.Login(fix.remoteDriver);
                step.Dashboard_SelectUserMaterialRightsModule
(fix.remoteDriver);
                step.UserMaterialRights_AddRights(fix.remoteDriv
er, userEmail, material);
                step.UserMaterialRights_DeleteRights(fix.remoteD
river);
            }
            finally
            {
                    fix.remoteDriver.Quit();
            }
}
```

Krok „UserMaterialRights_AddRights“ přidá práva mezi uživatelem a potřebným materiálem. Nejdříve skript klikne na tlačítko pro přidání záznamu. Po

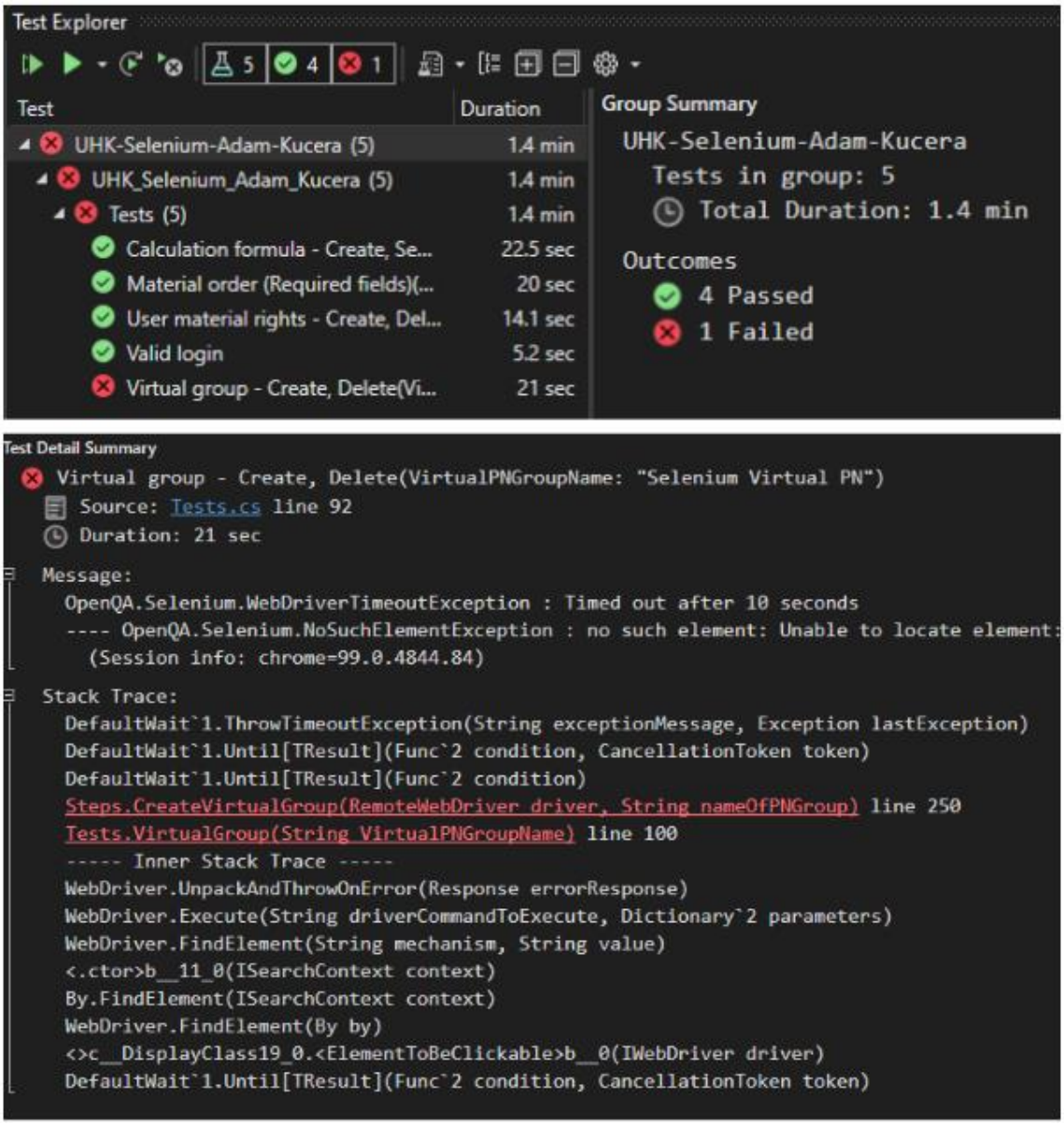
otevření okna zadá požadovaný email uživatele a označení materiálu, na který má právo. Poté se provede stisk tlačítka pro uložení a pokračuje se dalším krokem v testu.

```
public void UserMaterialRights_AddRights(RemoteWebDriver driver,
string userEmail, string material)
        {
            var wait = new WebDriverWait(driver, new TimeSpan(0,
0, 10));
            // klikne na tlacitko pro pridani
            wait.Until(SeleniumExtras.WaitHelpers.ExpectedCondit
ions.ElementToBeClickable(By.XPath("//mat-
icon[text()='add']"))).Click();
            // zadani emailu a povolenych zaznamu k videni
            wait.Until(SeleniumExtras.WaitHelpers.ExpectedCondit
ions.ElementToBeClickable(By.XPath("//input[@placeholder='User
Email']"))).Click();
            wait.Until(SeleniumExtras.WaitHelpers.ExpectedCondit
ions.ElementToBeClickable(By.XPath("//input[@placeholder='User
Email']"))).SendKeys(userEmail);
            wait.Until(SeleniumExtras.WaitHelpers.ExpectedCondit
ions.ElementToBeClickable(By.XPath("//textarea[@placeholder=Mate
rial]"))).Click();
            wait.Until(SeleniumExtras.WaitHelpers.ExpectedCondit
ions.ElementToBeClickable(By.XPath("//textarea[@placeholder='Mat
erial']"))).SendKeys(material);
            // tlacitko ulozit
            wait.Until(SeleniumExtras.WaitHelpers.ExpectedCondit
ions.ElementToBeClickable(By.Id("submit"))).Click();
        }
```

7.3 Vyhodnocení

Pro samotné testy je také důležité zobrazení jejich výsledků. Zde záleží, zda chceme vidět výsledky pro lokálně spuštěné testy, nebo pro testy běžící pouze na serveru.

Pro lokální vývoj je nejlepší použít přímo Text Explorer uvnitř Visual Studia. Ten poskytne přehledný souhrn všech testů a dodatečné informace o nich, jako je například délka běhu každého testu, nebo použité argumenty. Pokud test neproběhne v pořádku, program zobrazí konkrétní výjimku, kvůli které test nedoběhl a pokud je to možné, ukáže i přesný řádek v kódu, kde se daná validace, popř. chyba, vyskytuje.



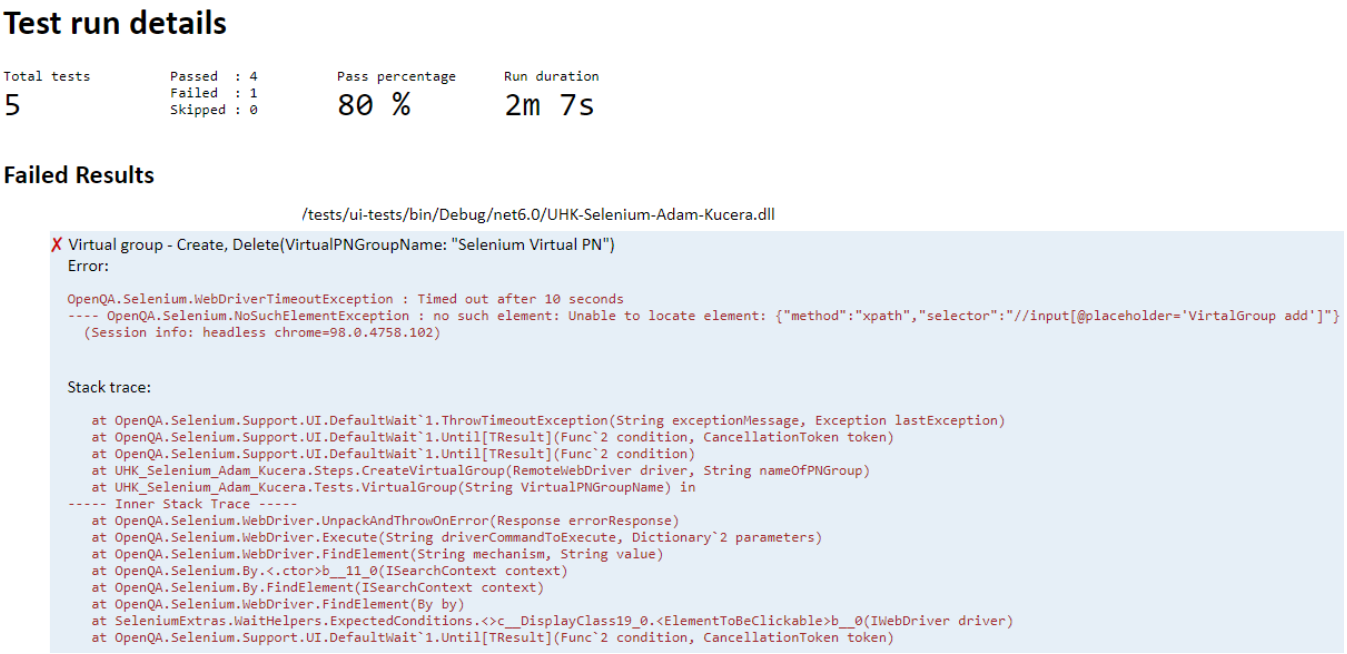
Obrázek 16: Microsoft Visual Studio – Test Explorer

Zdroj: Vlastní zpracování

Pokud jsou automatické testy spouštěny na serveru, je pro zobrazení výsledků nutné použít jiné prostředky. K tomuto účelu slouží protokolovací nástroj logger, který poskytuje testovací ovladač dotnet test. Tento příkaz slouží k provádění testů v daném projektu. Testy se provedou s pomocí vybraného testovacího frameworku, v našem případě XUnit. Pomocí parametru –logger určíme protokolovací nástroj pro výsledky testu. Konkrétní příklad je uveden níže. Je vidět, že v našem případě má mít výsledný soubor formát html.

```
dotnet test tests/uitest/UHK-Selenium-Adam-Kucera.csproj --logger html
```

Obrázek níže zobrazuje vygenerovaný report po dokončení pěti testů. V levém horním rohu jsou základní informace jako celkový počet provedených testů, počet úspěšných, neúspěšných a přeskočených. Dále procentuální vyjádření úspěšnosti a celkový čas běhu všech testů. Dále jsou detailněji popsány testy, které byly neúspěšné. Díky tomuto popisu, v jaké části kódu došlo k chybě, je možné testy zkontrolovat ručně, popřípadě pustit znovu, či nahlásit objevenou chybu v aplikaci k opravě.



Obrázek 17: HTML Logger

Zdroj: Vlastní zpracování

8 Závěry a doporučení

Cílem práce bylo vytvořit testy pro automatické testování uživatelského grafického rozhraní aplikace v anonymní společnosti. Výstupem je sada těchto testů napsaných v jazyce C# za pomocí nástroje Selenium WebDriver. Testy pokrývají základní funkcionality aplikace, jako je přihlášení aplikace, objednání materiálu, vytvoření vzorce pro výpočet, virtuální skupina materiálu a přiřazení práv uživateli, jaký materiál může vidět. Dále jsou ukázány dva reporty výsledků těchto testů. První je přímo nástroj z Visual Studia, který je nápomocný při lokálním vytváření testů. Druhým z nich je html report, který je vygenerován po dokončení běhu testů v prostředí GitLab. Testy jsou vytvořeny pro aktuální verzi firemní aplikace. Pro testování budoucích verzí aplikace bude nutné tyto automatické testy aktualizovat, aby odpovídaly nejnovějšímu vývoji.

Teoretická část práce shrnuje poznatky o testování softwaru. Na začátku práce jsou uvedeny důvody, proč je testování softwaru důležité a konkrétní příklady z historie, kdy na testování nebyl kladen dostatečný důraz, popř. došlo k přehlédnutí chyb v softwaru. V další části práce jsou představeny různé druhy testování softwaru a pomocí V modelu je ukázáno, do jaké části vývojového procesu jednotlivé druhy patří. Jsou představeny testovací úrovně, rozdíly v testování na frontend a backend části aplikace a je popsáno testování podle znalosti zdrojového kódu aplikace. Práce pokračuje popsáním manuálního a automatického způsobu testování. Jelikož je tato práce věnována vytváření automatických testů, je druhému zmíněnému typu testování vyhrazena samostatná kapitola. V té jsou uvedeny principy automatického testování, jeho výhody, nevýhody a ke konci jsou stručně popsány technologie, které se pro automatické testování používají.

Další kapitola je věnována frameworku Selenium. Je provedeno představení tohoto frameworku, popsána jeho historie a uvedeny jednotlivé nástroje, které nabízí. Každý z těchto nástrojů je podrobněji popsán. Kapitola pokračuje popsáním jednotlivých lokátorů, které slouží k identifikaci webových elementů použitých v testech. Popsány jsou také základní operace, které Selenium pro interakci s

elementy nabízí. Kapitola číslo 6 obsahuje stručný popis programů a technologií, které jsou použity při vytváření automatický testů pro uživatelské grafické rozhraní aplikace. Poslední kapitola se věnuje konkrétní implementaci automatických testů.

Nelze s určitostí říci, zda je pro vývoj softwaru výhodnější použít automatické, či manuální testování. Každý typ má své klady a zápory. Jako optimální řešení se jeví používat především automatické testování, doplněné testováním manuálním. Automatické testy přináší velkou časovou úsporu při regresním testování, kdy jeho potřebu prakticky eliminují.

9 Seznam použité literatury

[1] JAMIL, Muhammad, Muhammad ARIF, Normi ABUBAKAR a Akhlaq AHMAD. Software Testing Techniques: A Literature Review. In: *2016 6th International Conference on Information and Communication Technology for The Muslim World (ICT4M)*. Jakarta: IEEE, 2016, s. 177-182. Dostupné z: doi:10.1109/ICT4M.2016.045

[2] IEEE STANDARDS ASSOCIATION. *IEEE 610.12-1990*. 1990. Dostupné z: doi:10.1109/IEEESTD.1990.101064

[3] MYERS, Glenford, Tom BADGETT a Corey SANDLER. *The art of software testing*. 3rd ed. Hoboken, New Jersey: Wiley, 2012. ISBN 1118031962.

[4] GALIN, Daniel. *Software Quality Assurance: From Theory to Implementation*. 1st edition. England: Pearson College Div, 2003. ISBN 978-0201709452.

[5] Top 10 famous computer bugs that cost millions of dollars. In: *Techworm* [online]. 2020 [cit. 2021-04-15]. Dostupné z: https://www.techworm.net/2016/12/top-10-famous-computer-bugs-cost-millions-dollars.html

[6] PERLA, Enrico, Oldani MASSIMILIANO a Graham SPEAKE. *A guide to kernel exploitation: attacking the core*. 1st Edition. Amsterdam: Elsevier, 2011. ISBN 978-1-59749-486-1.

[7] Y2K bug. In: *National Geographic* [online]. 2011 [cit. 2021-04-18]. Dostupné z: https://www.nationalgeographic.org/encyclopedia/Y2K-bug/

[8] PATTON, Ron. *Software testing*. 2nd ed. Indianapolis: Sams Publishing, 2006. ISBN 0672327988.

[9] V-Model. In: *Javatpoint* [online]. [cit. 2021-04-20]. Dostupné z: https://www.javatpoint.com/software-engineering-v-model

[10] MATHUR, Sonali a Shaily MALIK. Advancements in the V-Model. *International Journal of Computer Applications* [online]. 2010, **1**(12), 29-34 [cit. 2021-04-20]. ISSN 09758887. Dostupné z: doi:10.5120/266-425

[11] DASSO, Aristides. *Verification, Validation and Testing in Software Engineering*. 1st Edition. IGI Global, 2006. ISBN 978-1591408512.

[12] Certified Tester - Foundation Level Syllabus: Version 2018 V3.1. In: *ISTQB* [online]. ISTQB, 2018 [cit. 2021-04-22]. Dostupné z: https://www.istqb.org/downloads/syllabi/foundation-level-syllabus.html

[13] MITRA, Porshia, Shreya CHATTERJEE a Nikita ALI. Graphical analysis of MC/DC using automated software testing. In: *2011 3rd International Conference on Electronics Computer Technology* [online]. IEEE, 2011, s. 145-149 [cit. 2021-04-25]. ISBN 978-1-4244-8678-6. Dostupné z: doi:10.1109/ICECTECH.2011.5941819

[14] SHAO, Danhua, Sarfraz KHURSHID a Dewayne PERRY. A Case for White-box Testing Using Declarative Specifications Poster Abstract. In: *Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION (TAICPART-MUTATION 2007)* [online]. Windsor: IEEE, 2007, s. 137-137 [cit. 2021-05-01]. ISBN 0-7695-2984-4. Dostupné z: doi:10.1109/TAIC.PART.2007.36

[15] IEEE STANDARDS ASSOCIATION. *IEEE 829-2008 - IEEE Standard for Software and System Test Documentation*. 1. IEEE, 2008. Dostupné také z: 10.1109/IEEESTD.2008.4578383

[16] KUMAR, RAJENDER. Analytical Study on Manual vs. Automated Testing Using with Simplistic Cost Model. In: *International Journal of Electronics and Electrical Engineering* [online]. 2012, s. 23-35 [cit. 2021-05-10]. ISSN 2277-7040. Dostupné z: https://vixra.org/pdf/1208.0216v1.pdf

[17] ITKONEN, Juha, Mika MANTYLA a Casper LASSENIUS. Defect Detection Efficiency: Test Case Based vs. Exploratory Testing. In: *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)* [online]. IEEE, 2007, s. 61-70 [cit. 2021-05-15]. ISBN 978-0-7695-2886-1. Dostupné z: doi:10.1109/ESEM.2007.56

[18] HENRY, Pierre. *The Testing Network: An Integral Approach to Test Activities in Large Software Projects*. 1. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. ISBN 978-3-540-78504-0.

[19] MERSINO, Anthony. Why Agile is Better than Waterfall. In: *VitalityChicago* [online]. 2021 [cit. 2021-11-29]. Dostupné z: https://vitalitychicago.com/blog/agile-projects-are-more-successful-traditional-projects/

[20] BAUMGARTNER, Manfred, Martin KLONK a Christian MASTNAK. *Agile Testing: The Agile Way to Quality*. 1st Edition. Switzerland: Springer, Cham, 2021, 257 s. ISBN 978-3-030-73209-7. Dostupné z: doi:https://doi.org/10.1007/978-3-030-73209-7

[21] FISHMAN, Stephen H. Testing is Good. Pyramids are Bad. Ice Cream Cones are the Worst. In: *Medium* [online]. 2016 [cit. 2021-11-28]. Dostupné z:

https://medium.com/@fistsOfReason/testing-is-good-pyramids-are-bad-ice-cream-cones-are-the-worst-ad94b9b2f05f

[22] RAFI, Dudekula, Katam MOSES, Kai PETERSEN a Mika MÄNTYLÄ. *Benefits and limitations of automated software testing: Systematic literature review and practitioner survey*. Zurich, Switzerland: IEEE, 2012. ISBN 978-1-4673-1822-8. Dostupné z: doi:https://doi.org/10.1109/IWAST.2012.6228988

[23] LENZ, Moritz. *Python Continuous Integration and Delivery: A Concise Guide with Examples*. 1st Edition. Berkeley, CA: Apress, 2018. ISBN 978-1-4842-4281-0. Dostupné z: doi:https://doi.org/10.1007/978-1-4842-4281-0

[24] VALLEY, Matt. False positive & false negative in software testing. In: *Testfully* [online]. Testfully, 2021 [cit. 2021-11-29]. Dostupné z: https://testfully.io/blog/false-positive-false-negative/

[25] Apache JMeter. In: *The Apache Software Foundation* [online]. [cit. 2021-11-30]. Dostupné z: https://jmeter.apache.org/

[26] YANG, Chou. Test Automation in 2020: Findings from the DevTestOps Landscape Report. In: *Mabl* [online]. 2020 [cit. 2021-11-30]. Dostupné z: https://www.mabl.com/blog/test-automation-in-2020-findings-from-the-devtestops-landscape-report

[27] Behavior Driven Development (BDD). In: *Agile Alliance* [online]. [cit. 2021-11-30]. Dostupné z: https://www.agilealliance.org/glossary/bdd/

[28] *Open Source Initiative: Licenses & Standards* [online]. In: . [cit. 2022-02-26]. Dostupné z: https://opensource.org/licenses

[29] GitHub: SeleniumHQ/selenium. In: *GitHub* [online]. [cit. 2022-02-26]. Dostupné z: https://github.com/SeleniumHQ/selenium

[30] Selenium: Install browser drivers. In: *Selenium* [online]. 2022 [cit. 2022-02-26]. Dostupné z: www.selenium.dev/documentation/webdriver/getting_started/install_drivers

[31] Selenium overview. In: *Selenium* [online]. 2021 [cit. 2022-02-26]. Dostupné z: https://www.selenium.dev/documentation/overview/

[32] Selenium: Selenium History. In: *Selenium* [online]. [cit. 2022-02-26]. Dostupné z: https://www.selenium.dev/history/

[33] Same-origin policy. In: *MDN Web Docs* [online]. 2022 [cit. 2022-02-26]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy

[34] Selenium: WebDriver: Getting started. In: *Selenium* [online]. 2022 [cit. 2022-02-26]. Dostupné z: https://www.selenium.dev/documentation/webdriver/getting_started/

[35] STEWART, Simon. Selenium: Announcing Selenium 4. In: *Selenium* [online]. [cit. 2022-02-26]. Dostupné z: https://www.selenium.dev/blog/2021/announcing-selenium-4/

[36] Selenium: Selenium IDE: Getting Started. In: *Selenium* [online]. 2019 [cit. 2022-02-26]. Dostupné z: https://www.selenium.dev/selenium-ide/docs/en/introduction/getting-started

[37] WebDriver: Documentation. In: *Selenium* [online]. [cit. 2022-02-26]. Dostupné z: https://www.selenium.dev/documentation/webdriver/

[38] Selenium WebDriver Achitecture. In: *ToolsQA* [online]. [cit. 2022-02-27]. Dostupné z: https://www.toolsqa.com/selenium-webdriver/selenium-webdriver-architecture/

[39] When to Use a Grid. In: *Selenium* [online]. [cit. 2022-02-27]. Dostupné z: https://www.selenium.dev/documentation/grid/applicability/

[40] Selenium Grid Components. In: *Selenium* [online]. [cit. 2022-02-27]. Dostupné z: https://www.selenium.dev/documentation/grid/components/

[41] Finding web elements. In: *Selenium* [online]. [cit. 2022-02-27]. Dostupné z: https://www.selenium.dev/documentation/webdriver/elements/finders/

[42] Locator strategies. In: *Selenium* [online]. [cit. 2022-02-27]. Dostupné z: https://www.selenium.dev/documentation/webdriver/elements/locators/

[43] Selenium Locators — A Detailed Guide. In: *Medium* [online]. [cit. 2022-02-27]. Dostupné z: https://medium.com/analytics-vidhya/selenium-locators-a-detailed-guide-784572b5d718

[44] Interacting with web elements. In: *Selenium* [online]. [cit. 2022-02-27]. Dostupné z: https://www.selenium.dev/documentation/webdriver/elements/interactions/

[45] Waits. In: *Selenium* [online]. [cit. 2022-02-27]. Dostupné z: https://www.selenium.dev/documentation/webdriver/waits/

[46] ExpectedConditions Class. In: *Selenium* [online]. [cit. 2022-02-27]. Dostupné z: https://www.selenium.dev/selenium/docs/api/dotnet/html/T_OpenQA_Selenium_Support_UI_ExpectedConditions.htm

[47] Visual Studio 25th Anniversary. In: *Microsoft* [online]. [cit. 2022-04-01]. Dostupné z: https://visualstudio.microsoft.com/cs/vs-25th-anniversary/

[48] Visual Studio Marketplace: Extensions for the Visual Studio family of products. In: *VS Marketplace* [online]. [cit. 2022-04-01]. Dostupné z: https://marketplace.visualstudio.com/

[49] Visual Studio: Compare Visual Studio 2022 Editions. In: *Microsoft* [online]. [cit. 2022-04-01]. Dostupné z: https://visualstudio.microsoft.com/vs/compare/

[50] Microsoft technical documentation: An introduction to NuGet. In: *Microsoft* [online]. [cit. 2022-04-02]. Dostupné z: https://docs.microsoft.com/en-us/nuget/what-is-nuget

[51] OSHEROVE, Roy. *The art of unit testing with examples in C#*. 2nd ed. New York: Manning, 2014. ISBN 9781617290893.

[52] Xunit. In: *NuGet* [online]. [cit. 2022-04-04]. Dostupné z: https://www.nuget.org/packages/xunit

[53] Desktop Browser Market Share Worldwide: Jan 2009 - Apr 2022. In: *Statcounter Global Stats* [online]. [cit. 2022-04-04]. Dostupné z: https://gs.statcounter.com/browser-market-share/desktop/worldwide#monthly-200901-202204

[54] Chrome DevTools: Overview. In: *Chrome Developers* [online]. [cit. 2022-04-05]. Dostupné z: https://developer.chrome.com/docs/devtools/overview/

[55] List of Chromium Command Line Switches. In: *Peter Beverloo* [online]. [cit. 2022-04-06]. Dostupné z: https://peter.sh/experiments/chromium-command-line-switches/

10 Přílohy

Příloha 1 – Struktura přiloženého souboru: UHK-FIM-Selenium-Adam-Kucera.zip

Struktura přiloženého souboru: UHK-FIM-Selenium-Adam-Kucera.zip

Konfigurační soubor pro nastavení test jobu v GitLab pipeline

\gitlab\job-config-file\.gitlab-ci.yml

Soubor C# projektu k otevření všech tříd a referencí

\visual-studio\UHK-Selenium-Adam-Kucera\UHK-Selenium-Adam-Kucera.csproj

Soubor k otevření třídy Steps, která obsahuje jednotlivé kroky, ze kterých jsou testy složeny

\visual-studio\UHK-Selenium-Adam-Kucera\Steps.cs

Soubor k otevření třídy Tests, která obsahuje testy

\visual-studio\UHK-Selenium-Adam-Kucera\Tests.cs

Soubor k otevření třídy SeleniumConfig obsahující základní nastavení Chrome prohlížeče použitého při testech

\visual-studio\UHK-Selenium-Adam-Kucera\SeleniumConfig.cs

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání bakalářské práce

Autor:	**Adam Kučera**
Studium:	I1900330
Studijní program:	B0688A140001 Informační management
Studijní obor:	Informační management
Název bakalářské práce:	**Automatické testy v Selenium framework**
Název bakalářské práce AJ:	Automation tests in Selenium framework

Cíl, metody, literatura, předpoklady:

Cílem práce je popsat, navrhnout a otestovat scénáře pro automatické testy, zaměřené na framework Selenium a jeho zabezpečení, vytvořené v jazyce C#, běžící v GitLabu a formu následné prezentace výsledků těchto testů.
V teoretické části autor zpracuje problematiku automatických testů s důrazem na framework Selenium, platformu GitLab a přístupy k zobrazování výsledků automatických testů.
V praktické části autor navrhne a otestuje scénáře testů pro reálné webové aplikace a otestuje navržený systém prezentace výsledků těchto testů v přehledném grafickém provedení.

DAS, Ravi a Greg JOHNSON. *Testing and Securing Web Applications*. London, United Kingdom: Taylor & Francis, 2020. ISBN 9780367333751.

Garantující pracoviště:	Katedra informačních technologií, Fakulta informatiky a managementu
Vedoucí práce:	Mgr. Josef Horálek, Ph.D.
Datum zadání závěrečné práce:	21.1.2020