Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Diplomová práce

Porovnání testovacích frameworků Protractor a Cypress na platformě Angular

Bc. David Pražák

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. David Pražák

Systémové inženýrství a informatika
Informatika

Název práce

Porovnání testovacích frameworků Protractor a Cypress na platformě Angular

Název anglicky

Comparison of test frameworks Protractor and Cypress on the Angular platform

---

Cíle práce

Diplomová práce je tematicky zaměřena na problematiku testování webových aplikací na platformě Angular.
Hlavním cílem práce je porovnání frameworků Protractor a Cypress pro použití ve vybrané oblasti.
Dílčí cíle práce jsou:
- charakteristika nástrojů pro testování webových aplikací,
- výběr vhodných metrik pro porovnání frameworků,
- vyhodnocení testů vytvořených jednotlivými frameworky.

Metodika

Řešení diplomové práce je založeno na studiu a analýze odborných informačních zdrojů. Pro porovnání frameworků bude využita webová aplikace na platformě Angular, pro kterou budou vytvořeny shodné testy za použití frameworků Protractor a Cypress. Poté bude provedeno porovnání uvedených frameworků pomocí vhodných metod a parametrů. Na základě získaných poznatků budou formulovány závěry diplomové práce.

Doporučený rozsah práce

60 – 80 stran

Klíčová slova

Protractor, Cypress, framework, testování, webová aplikace, Angular

---

Doporučené zdroje informací

DARWIN, Peter Bacon; KOZLOWSKI, Pawel. AngularJS web application development. Packt Publishing Ltd, 2013. ISBN 978-1-78216-182-0.

CHAPLIN, Tim. AngularJS Test-driven Development. Packt Publishing Ltd, 2015. ISBN 978-1-78439-883-5.

PALMER, Jesse, et al. Testing Angular Applications. Manning Publications Co., 2018. ISBN 978-1-61729-364-1.

RAGONHA, Paulo. Jasmine JavaScript Testing. Packt Publishing Ltd, 2013. ISBN 978-1-78528-204-1.

Předběžný termín obhajoby

2020/21 LS – PEF

Vedoucí práce

Ing. Jan Masner, Ph.D.

Garantující pracoviště

Katedra informačních technologií

Elektronicky schváleno dne 29. 7. 2020

Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 21. 10. 2020

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 15. 02. 2021

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Porovnání testovacích frameworků Protractor a Cypress na platformě Angular" jsem vypracoval(a) samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor(ka) uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 30.03.2021 _____

Poděkování

Rád(a) bych touto cestou poděkoval(a) Ing. Janu Masnerovi, Ph.D. za pomoc a cenné rady při tvorbě této diplomové práce.

Porovnání testovacích frameworků Protractor a Cypress na platformě Angular

Abstrakt

Diplomová práce se zabývá porovnáním testovacích frameworků Protractor a Cypress při testování vnitropodnikové webové aplikace, založené na platformě Angular, spouštěné v prohlížečích Google Chrome, Mozilla Firefox a Microsoft Edge. Pro porovnání byly v obou frameworcích vytvořeny shodné testy, kterými byla aplikace v jednotlivých prohlížečích otestována. V průběhu vykonávání testů proběhla měření, která poskytla informace o rychlosti provádění jednotlivých testů a vytížení procesoru a paměti testovacího počítače. Naměřené hodnoty byly použity k porovnání frameworků ve zvolených prohlížečích. Závěrem bylo provedeno vyhodnocení s cílem volby vhodnějšího frameworku pro testování podobného typu webových aplikací.

Klíčová slova: Protractor, Cypress, framework, testování, webová aplikace, Angular

Comparison of Protractor and Cypress testing frameworks on the Angular platform

Abstract

The diploma thesis deals with the comparison of Protractor and Cypress testing frameworks when testing an in-house web application, based on the Angular platform, running in Google Chrome, Mozilla Firefox and Microsoft Edge browsers. For comparison, identical tests were created in both frameworks, which were used to test the application in individual browsers. During the tests, measurements were taken, which provided information on the speed of individual tests and the CPU and memory usage of the test computer. The measured values were used to compare frameworks in selected browsers. Finally, an evaluation was performed in order to select a more suitable framework for testing a similar type of web applications.

Keywords: Protractor, Cypress, framework, testing, web application, Angular

Obsah

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek

HTML - Hypertext Markup Language - Značkovací jazyk

NPM - Node Package Manager - Správce JavaScriptových balíčků

URL - Uniform Resource Locator - Jednotný lokátor zdroje

JSON - JavaScript Object Notation - Způsob zápisu dat

MB - MegaByte – Jednotka množství dat

1 Úvod

Webové aplikace jsou trendem dnešní doby. Z velké části tomu napomáhá rozšiřující se dostupnost internetu, kdy jeho uživatelé aktuálně tvoří již více jak polovinu světové populace. Pro vývoj webových aplikací se nabízí několik způsobů řešení. Jedním z nejnovějších přístupů je Single Page Application, česky jednostránková aplikace. Tento přístup je využíván například u produktů Facebook, Twitter, Linkedin, Gmail apod. K tvorbě jednostránkových aplikací nejčastěji slouží frameworky Angular a React.

Součástí vývoje každé kvalitní aplikace musí být také její důkladné otestování. Pokud je tento proces dobře nastaven, může nemalou částí přispět k vyšší výsledné kvalitě celé aplikace. Cílem každé softwarové firmy musí být produkce aplikací s minimálním množstvím chyb. Jakákoliv chyba, která pronikne do produkčního prostředí, přináší mnohem vyšší ztráty než chyba odhalená ještě před nasazením.

Velmi důležitou oblastí testování je funkční testování. Jeho úkolem je napodobováním činností uživatele při práci s aplikací odhalit maximum chyb a nedostatků. V případě dlouhodobých či rozsáhlých projektů je vhodné zvážit, zda by se část této činnosti neměla automatizovat. K automatizaci testování lze využít mnoho dostupných nástrojů. Aplikace postavené na platformě Angular lze s úspěchem automatizovaně testovat prostřednictvím k tomu určených testovacích frameworků. Mezi nejznámější testovací frameworky patří Protractor a Cypress.

Tato práce je zaměřena na porovnání těchto frameworků při testování vnitropodnikových aplikací založených na platformě Angular s cílem zjistit pro dané použití nejvhodnější variantu.

2 Cíl práce a metodika

2.1 Cíl práce

Diplomová práce je zaměřena na problematiku testování webových aplikací na platformě Angular. Zaměřuje se na možnosti využití testovacích frameworků v praxi a jejím hlavním cílem je porovnání frameworků Protractor a Cypress v oblasti vnitropodnikových aplikací na platformě Angular.

Dílčí cíle práce jsou:

- Charakteristika nástrojů pro testování webových aplikací.
- Výběr vhodných metrik pro porovnání frameworků.
- Vyhodnocení testů vytvořených jednotlivými frameworky.

2.2 Metodika

Řešení diplomové práce je založeno na studiu a analýze odborných informačních zdrojů. Pro porovnání frameworků bude využita webová aplikace na platformě Angular, pro kterou budou vytvořeny shodné testy za použití frameworků Protractor a Cypress. Na základě vytvořených testů bude provedeno měření rychlosti provedení těchto testů nad webovými prohlížeči Google Chrome, Mozilla Firefox a Microsoft Edge. Poté bude provedeno měření vytížení hardwarových prostředků v průběhu vykonávání jednoho konkrétního testu nad uvedenými prohlížeči. Data z jednotlivých měření poslouží pro porovnání frameworků, které bude realizováno formou vícekriteriální analýzy variant. Na základě získaných poznatků budou formulovány závěry diplomové práce.

3 Teoretická východiska

3.1 Testování softwaru

Testování softwaru je proces, při kterém dochází k ověřování správnosti chování testované aplikace z různých pohledů, aby se ověřilo splnění všech požadovaných parametrů pro její bezproblémový rutinní provoz. Jde především o následující oblasti testování [1], [2], [3], [4], [5], [6]:

- Unit testování
- Integrační testování
- Bezpečnostní testování
- Funkční testování
- Testování použitelnosti
- Testování výkonu
- Akceptační testování

Tato práce se zaměřuje na oblast funkčního testování aplikací.

3.1.1 Unit testování

Unit (jednotkové) testování softwaru slouží pro otestování zdrojového kódu aplikace. Jedná se o jednu z nejdůležitějších částí vývoje softwaru. Toto testování je prováděno vývojáři, kteří mají za cíl ověřit korektní chování jednotlivých komponent aplikace jako jsou funkce, knihovny, objekty apod. [1]

3.1.2 Integrační testování

Hlavním cílem integračního testování je otestování integračního rozhraní mezi moduly nebo spolupracujícími systémy testované aplikace. Toto testování probíhá při samotném vývoji testované aplikace. Integrační testování má na starosti testovací tým, který aplikaci manuálně nebo automatizovaně testuje. Cílem tohoto testování je odhalit chyby v testované aplikaci co nejdříve, aby mohly být opraveny před finálním nasazením testované aplikace do ostrého provozu. [2]

3.1.3 Bezpečnostní testování

Bezpečnostní testování se zaměřuje na několik oblastí. Těmito oblastmi jsou především kvalita kódu z pohledu dodržení bezpečnostních pravidel a standardů, konfigurace aplikačního prostředí, zabezpečení přístupu do aplikace, aplikačních dat atd. Součástí bezpečnostního testování je rovněž i ověření odolnosti proti útoku na aplikaci s cílem jejího poškození či ztráty dat. [3]

3.1.4 Funkční testování

Funkční testování softwaru slouží převážně pro eliminaci chyb, které vznikly neúmyslně při samotném vývoji softwaru. Samotné testování softwaru probíhá na základě testovacích scénářů, které obsahují podrobné informace o tom, co a jak má být otestováno. Testovací scénáře jsou nejčastěji uspořádány do kroků, které na sebe navazují a tester nebo automatizovaný test podle těchto kroků postupuje. Na základě výsledků testování lze rozhodnout, zda je otestovaný software připravený k předání do rutinního provozu, nebo je třeba pokračovat ve vývoji a testování. [3]

3.1.5 Testování použitelnosti

Testováním použitelnosti se ověřuje přehlednost aplikace z pohledu orientace uživatelů v testované aplikaci. Obvykle se zde využívá termínu User Experience neboli uživatelského zážitku. Cílem každé aplikace by mělo být maximalizovat tento uživatelský zážitek. K testování použitelnosti je k dispozici celá řada metod. Mezi známé metody patří například Card sorting, Focus group a heuristické testování. Pro testování použitelnosti lze také využít i nástroje jako jsou například Google Analytics. [4]

3.1.6 Testování výkonu

Testování výkonu se někdy také nazývá zátěžové testování. Pomocí této činnosti lze zjišťovat stabilitu a výkon aplikace a aplikační infrastruktury při různém provozním zatížení. Testování se může zaměřit na zvládání konkrétní zátěže, nebo na zjištění maximálního výkonu. Na základě získaných výstupů lze následně rozhodnout o nutnosti optimalizace aplikace či infrastruktury, nebo konstatovat, že bylo dosaženo požadovaného výkonnostního cíle testovaného řešení. [5]

3.1.7 Akceptační testování

Akceptační testování má za cíl prokázat, že otestovaná aplikace splňuje všechny požadavky zadavatele na její vzhled, funkčnost, výkon, použitelnost apod. a je možné jí nasadit do ostrého provozu. Testování provádí obvykle přímo zadavatel a bývá součástí akceptačního řízení při přebírání aplikace zákazníkem. [6]

3.2 Metody testování

3.2.1 White box

White box je testovací technika, která se používá především v průběhu vývoje aplikace. Toto testování provádí často přímo vývojář aplikace, protože je k němu zapotřebí znalost a přístup k programovému kódu. Testování se zaměřuje na kvalitu kódu z pohledu funkčnosti, přehlednosti, bezpečnosti, dodržení standardů apod. Tato technika by měla být aplikována tak, aby jednotliví testeři nekontrolovali části kódu jejichž jsou autory. To přináší vyšší efektivitu použité techniky a zúčastnění testeři si takto navíc mohou rozšířit své znalosti. [7], [8]

3.2.2 Black box

Black box testování je metoda testování, která navrhuje testovací případy založené na informacích ze specifikace požadavků (množina funkcí, které má aplikace poskytovat). Jedná se o metodu, ve které tester aplikace nemá přístup ke zdrojovému kódu testované aplikace. Aplikace je pak pro něj jakási skříňka, kterou může zkoumat pouze na povrchu. Ten je reprezentován grafickým uživatelským rozhraním (GUI). K obsahu skříňky nemá přístup. Místo toho má k dispozici informace, že do černé skříňky lze vložit nějaké vstupy a černá skříňka vrátí nějaké výstupy zpět. Tyto informace lze získat analýzou specifikace požadavků. Z nich tester získá informace o tom, jaké funkce má testovaná aplikace poskytovat. Následně může do černé skříňky vkládat očekávané vstupy a testovat, že černá skříňka vrací požadované výstupy. [7]

Testování černé skříňky se provádí od začátku životního cyklu softwarového projektu. Všichni testeři musí být zapojeni do tohoto cyklu od samotného začátku projektu. Při testování černé skříňky je potřeba, aby byli testeři důkladně obeznámeni s analýzou a s

požadavky zákazníků (specifikace požadavků). Pro testování metodou Black box je vhodné předem vytvořit testovací scénáře a připravit testovací data. [7]

Testovací scénář je sada kroků popisujících akce, které mají být prostřednictvím testovacího skriptu provedeny v testované aplikaci. Dobrý testovací scénář musí být přehledný a srozumitelný. Pokud nebude tyto požadavky splňovat, může dojít k jeho nepochopení, což může zapříčinit chybné otestování aplikace. Chybným otestováním aplikace lze rozumět takový stav, po kterém nebude správně otestována požadovaná funkčnost aplikace. [7]

Testovací data jsou množina hodnot požadovaného typu a struktury, která budou vkládána do aplikace při jejím testování a jsou zapotřebí pro otestování požadované funkčnosti. [7]

3.2.3 Grey box

Grey box testy jsou kombinací obou předešlých způsobů, tedy způsobu White box a způsobu Black box. Při tomto způsobu testování je do jisté míry známa vnitřní struktura aplikace, což přináší přístup k interním datovým strukturám a algoritmům testované aplikace. Navrhované testovací případy tak mohou být přizpůsobeny těmto znalostem. [8]

3.3 Způsob provádění testů

3.3.1 Manuální testování

Manuální testování patří mezi jednu z nejčastěji využívaných metod testování softwaru z důvodu nízkých nákladů pro jeho zavedení. Dlouhodobé manuální testování může ale náklady na testování výrazně navyšovat, a proto je vždy dobré zvážit, zda po ukončení základního vývoje nevyužít rovněž automatizované testování. Pro rozhodnutí, zda a v jakém rozsahu testování automatizovat, je ale důležité zvážit mnoho okolností, především možnosti společnosti pro zavedení automatizovaného testování a vhodnost aplikace k automatizovanému testování. Manuální testování je prováděno přímo konkrétními testery, kteří na základě testovacích scénářů, analytických dokumentů a dokumentace defektů přímo v grafickém uživatelském prostředí aplikace ověřují její bezchybnost a požadovanou funkcionalitu. K tomuto způsobu testování je nutná funkční testovaná aplikace a testující osoby musí mít alespoň základní znalosti o jejím fungování. Manuální testování se uplatňuje především v období dynamického vývoje testované

aplikace pro testování nových funkcionalit a následně pro udržení kvality vývoje při kontrole dalších nových funkcionalit a ověřování funkčnosti opravených defektů. Výhodou je jeho značná flexibilita a rychlost zahájení. Nevýhodou je jeho pomalost a u větších aplikací i personální náročnost. [9]

3.3.2 Automatizované testování

Automatizované testování je proces, při kterém se testovaná aplikace testuje pomocí tzv. skriptů, které byly vytvořeny za pomoci nástroje pro tvorbu automatizovaných testů. Tvorba takovýchto skriptů je závislá na testovacích scénářích, které obsahují konkrétní postupy pro otestování požadované funkčnosti testované aplikace. Úkolem těchto skriptů je otestování konkrétní části testované aplikace bez nutnosti zásahu člověka. Jelikož je automatizované testování závislé na životním cyklu vývoje testované aplikace, je vhodné jej začít využívat až po dokončení základního vývoje testované aplikace. Důvodem je převážně to, aby se testovací skripty nemusely stále přizpůsobovat častým změnám testované aplikace. Při zavádění automatizovaného testování je potřeba si uvědomit, že počáteční náklady na zavedení jsou poměrně značné a jejich ekonomický přínos se projeví často až po několika letech. Z tohoto důvodu se v některých případech, například u krátkodobých projektů, nevyplatí automatizované testování zavádět. Hlavní výhodou automatizovaného testování je jeho rychlost a v dlouhodobém horizontu ekonomická úspora. Nevýhodou jsou vysoké počáteční náklady a čas pro vývoj testovacích skriptů. [9]

V automatizovaném testování existuje i technika nazvaná vývoj řízený testy. V tomto případě se automatizované testy vyvíjí na základě podrobné analýzy již při zahájení vývoje aplikace. Testy se poté pravidelně spouští a v okamžiku, kdy přestanou vykazovat chyby, je dosaženo požadovaného chování aplikace. [10]

3.3.3 Semiautomatické testování

Semiautomatické testování je proces, při kterém se testovaná aplikace testuje kombinací manuálního a automatizovaného testování. Tím, že je aplikace otestována oběma způsoby, je maximalizována pravděpodobnost odhalení případné chyby v testované aplikaci. Tento způsob je v praxi nejčastěji využívaným způsobem funkčního testování. Nevýhodou semiautomatického testování jsou vysoké náklady vynaložené na tým pro manuální a automatizované testování. [9]

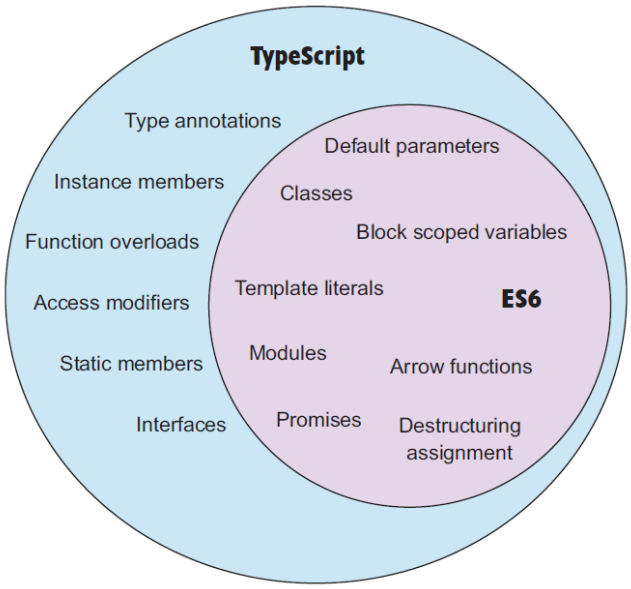
3.4 Angular

Angular je platforma a framework pro tvorbu jednostránkových webových aplikací. První verze frameworku byla vytvořena za pomocí jazyka JavaScript v roce 2009 společností Google. Pro vývoj webových aplikací se zde využívalo jazyků HTML a JavaScript. Důležitý zlom však nastal v roce 2016, kdy byla oficiálně vydána nová verze frameworku pod názvem Angular 2. Tato verze představuje kompletní přepis původní verze Angularu do jazyka TypeScript. Původní JavaScriptová verze frameworku však zůstala doposud využívána a je ze strany Google stále udržována. Z důvodu vznikajících zmatků ze strany vývojářů však došlo k přejmenování obou verzí Angularu. Původní JavaScriptová verze frameworku byla přejmenována na AngularJS a novější TypeScriptová verze byla přejmenována na Angular. [11], [12]

Obě zmíněné verze frameworku se zaměřují na tvorbu jednostránkových aplikací. Hlavní výhodou jednostránkových aplikací je oproti klasickým aplikacím to, že se po kliknutí na určitý odkaz nenačítá znovu celá webová stránka, ale načítá se pouze konkrétní obsah, který s použitým odkazem souvisí. Toto řešení tak poskytuje uživateli rychlý a plynulý zážitek s webovou aplikací. [13]

V rámci vývoje jednostránkových webových aplikací existuje mnoho dalších frameworků, které poskytují podobné funkce. Angular je však jedním z nejpoužívanějších frameworků v této oblasti. Jeho hlavní výhodou je velká komunitní podpora a především podpora ze strany Google. [13]

Pro vývoj aplikací v Angularu se používá jazyk TypeScript, který je určitou nadstavbou nad jazykem JavaScript. Jakýkoliv validní JavaScriptový kód je tak zároveň validním TypeScriptovým kódem. Výhodou v tomto případě je, že se při znalosti JavaScriptu není potřeba učit nový programovací jazyk. TypeScript zároveň přináší mnoho nových a užitečných funkcí, které v JavaScriptu chybí. Jedná se především o anotace, statické typování, klasické objektové orientované funkce jako jsou rozhraní a zapouzdření kódu. Tím, že je TypeScript nadstavbou nad JavaScriptem, tak poskytuje zároveň i klíčové funkce JavaScriptu viz obrázek č. 1. [13], [14]



Obrázek 1 - TypeSript a JavaScript [14]

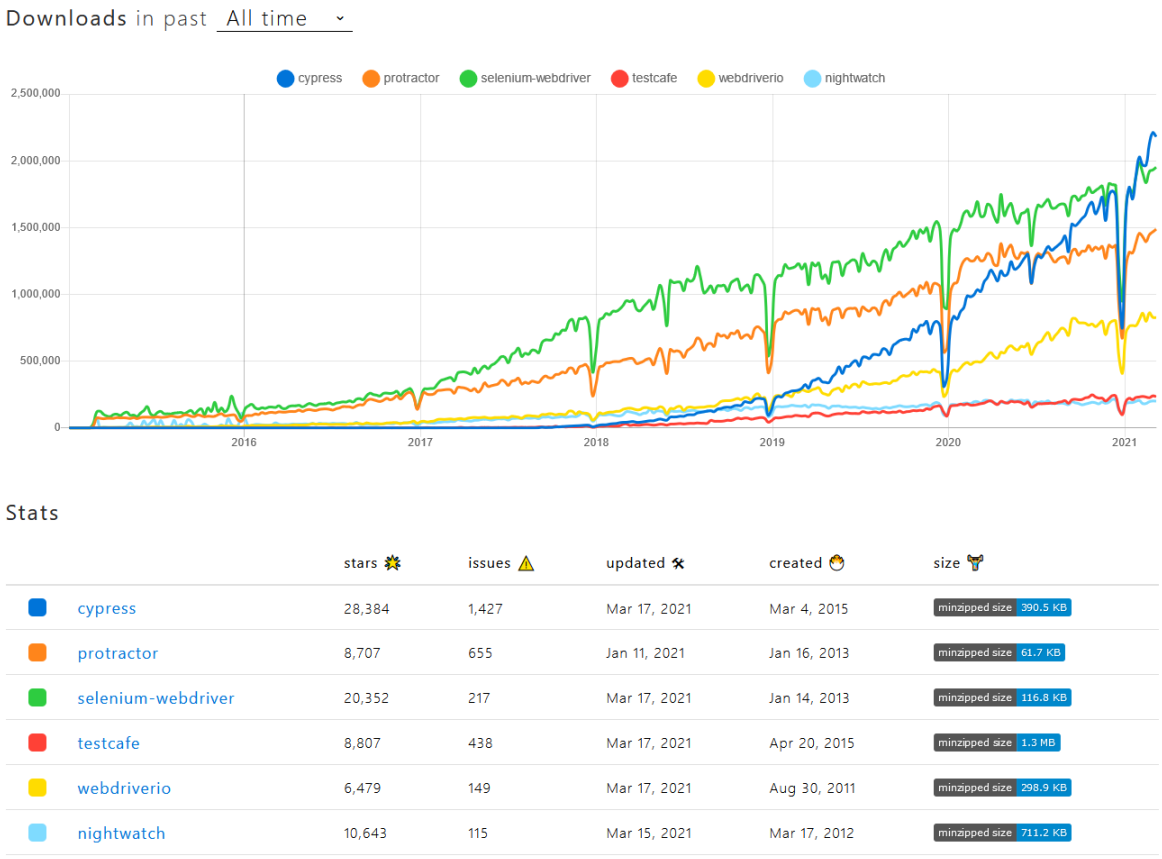
3.4.1 Testování na platformě Angular

V rámci testování aplikací v Angularu se využívají dva typy testů. Jedná se o unit testování a funkční testování. [14], [15]

Unit testy slouží k tomu, aby byla otestována funkčnost základních částí nebo jednotek zdrojového kódu. Každý unit test by měl testovat pouze jednu konkrétní oblast zdrojového kódu. Obecně lze těmito testy testovat funkce, metody, objekty, typy, hodnoty a další. Výhodou unit testů je, že jsou rychlé a opakovatelné, pokud jsou napsané správně a jsou spuštěny ve správném prostředí. Unit testy nejsou vhodné pro reprodukci skutečné uživatelské interakce. [14], [15]

Funkční testy mají za cíl otestovat funkčnost aplikace z pohledu simulace uživatelské interakce. Testy lze ověřit, zda aplikace poskytuje požadované funkce, zda tyto funkce pracují správně, zda má aplikace požadovaný vzhled, zda neobsahuje překlepy apod. Nevýhodou těchto testů je však jejich časová náročnost. [14], [15]

Pro automatizované testování aplikací na platformě Angular se využívají různé testovací nástroje a frameworky. Mezi nejpoužívanější patří Selenium, Cypress, WebdriverIO, Nightwatch Protractor a TestCafe. Popularitu jednotlivých frameworků lze vidět na obrázku č. 2. [16]



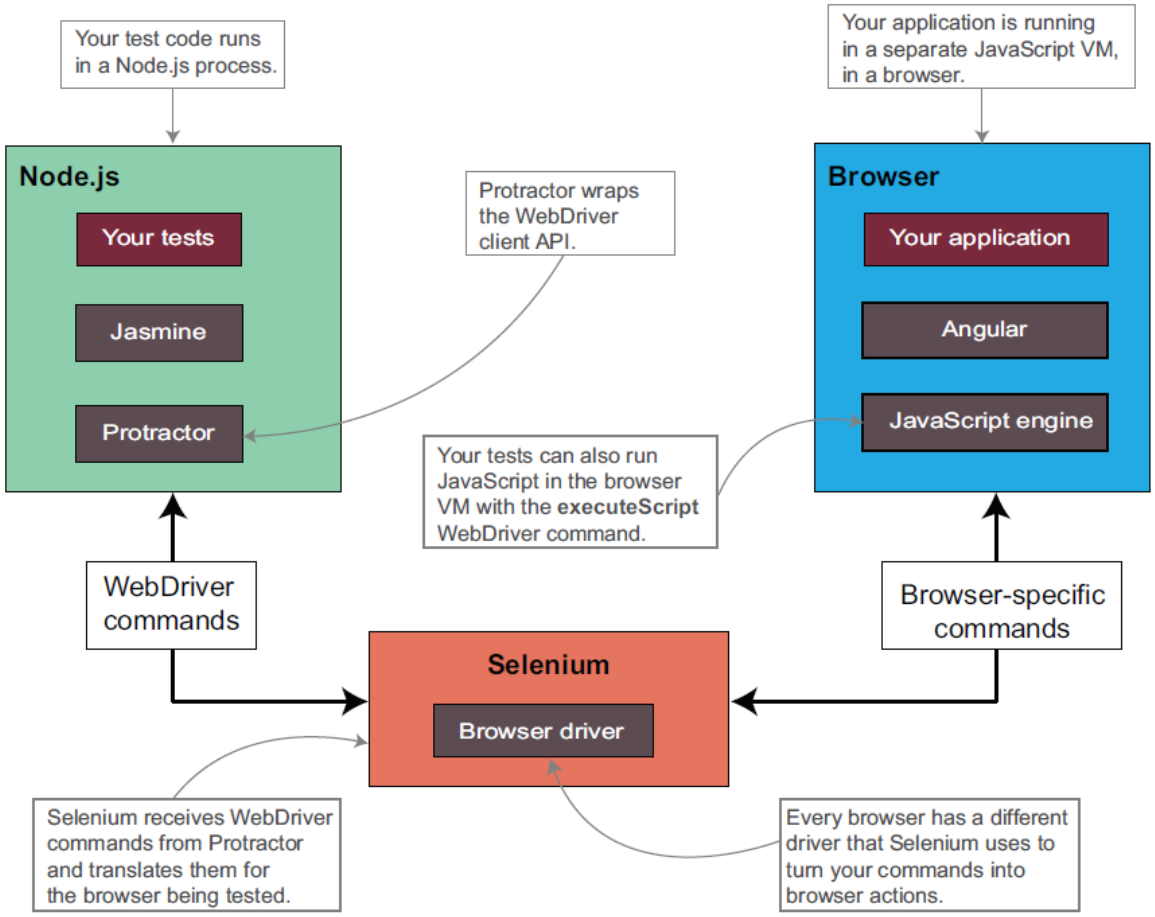
Obrázek 2 - Porovnání testovacích frameworků [17]

3.5 Protractor

Protractor je testovací framework, který je volně šířen pod licencí MIT. Slouží k testování webových aplikací z pohledu unit testování nebo funkčního testování. Funkční testování probíhá na základě komunikace s webovými prohlížeči, ve kterých vytvořené testy vykonávají v testované aplikaci požadovanou činnost. Protractor byl primárně vyvinut pro testování aplikací vytvořených na platformě Angular. [14]

3.5.1 Architektura frameworku

Pro spuštění testů v Protractoru se využívá procesu Node.js, který odesílá příkazy na Selenium Server. Tento server následně komunikuje s webovými prohlížeči pomocí WebDriveru a JavaScriptovými funkcemi viz obrázek č. 3. [14]

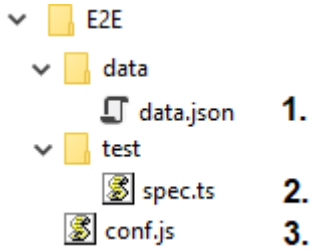


Obrázek 3 - Architektura Protractoru [14]

Protractor je Node.js program, který pro provádění testů využívá frameworku Jasmine nebo Mocha. Samotné provádění testů je realizováno tak, že Protractor provede zabalení WebDriver klienta, ke kterému následně přidá funkce pro testování aplikace. Tato data jsou předána na Selenium Server, který je přeloží a předá prohlížeči. Pro komunikaci s prohlížečem se využívá ovladačů webových prohlížečů. [14], [18]

3.5.2 Struktura souborů

Každý projekt v Protractoru se skládá z několika základních souborů, které reprezentují konfiguraci frameworku, testy a data pro testy. Konfigurační soubor bývá umístěn v kořenovém adresáři projektu a obsahuje informace o tom, jakým způsobem mají být spouštěny vytvořené testy. Zbylé soubory mohou být umístěny ve vlastních adresářích viz obrázek č. 4. [14]



Obrázek 4 - Souborová struktura Protractoru (vlastní zpracování)

1. Data je v rámci frameworku možné ukládat do souboru ve formátu JSON.
2. Jednotlivé testy se ukládají do souboru spec.ts.
3. Konfigurační soubor je reprezentován souborem conf.js.

3.5.3 Konfigurace prostředí

U Protractoru je konfigurace řešena v rámci konfiguračního souboru `conf.js`, který je k dispozici v kořenovém adresáři vytvořeného projektu. V rámci této konfigurace je na samotný úvod potřeba nastavit, jakým způsobem bude testovací framework přistupovat k ovladačům jednotlivých prohlížečů. Tento přístup lze realizovat dvěma způsoby. [14], [19], [20]

Prvním a zároveň defaultním způsobem je připojení pomocí Selenium Serveru. Při spuštění jakéhokoli testu si Protractor v defaultním nastavení automaticky vytváří instanci Selenium Serveru, prostřednictvím které je test vykonáván. Pro vytvoření instance Selenium Serveru slouží `jar` soubor (např. `selenium-server-standalone-3.0.1.jar`), který je defaultně umístěn v adresáři `node_modules/protractor/node_modules/webdriver-manager/selenium`. V konfiguraci Protractoru je možné nastavit vlastní cestu k Selenium Serveru. K tomu slouží parametr `seleniumServerJar`. [14], [19], [20]

```
exports.config = {
    seleniumServerJar: '../selenium-server-standalone-
3.0.1.jar',
};
```

V případě, že pro Selenium Server není potřeba vytvářet novou instanci, protože již běží na konkrétní adrese, využívá se parametru `seleniumAdress`. [14], [19], [20]

```
exports.config = {
    seleniumAddress: 'http://localhost:4444/wd/hub',
};
```

Druhým způsobem je připojení k ovladačům prohlížečů tzv. napřímo, pro které slouží parametr `directConnect`. Při tomto nastavení Protractor umožňuje spouštět jednotlivé testy bez použití Selenium Serveru. Veškerá komunikace tak probíhá přímo přes samotné ovladače prohlížečů. Tento způsob připojení je však dostupný pouze pro prohlížeče Chrome a Firefox s ovladači `chromedriver` a `geckodriver`. [14], [20]

```
exports.config = {
    directConnect: true,
};
```

V konfiguračním souboru Protractoru je dále nutné nadefinovat cestu k souborům reprezentující testy. Toto nastavení se provádí pomocí parametru `specs`. [14], [19]

```
exports.config = {
    specs: ['../at/specs/**/*.spec.ts'],
};
```

V rámci jednotlivých testů lze vytvářet rovněž testovací sady, které mohou obsahovat jeden či více testů ke spuštění. K vytvoření testovacích sad slouží parametr `suites`. [22]

```
exports.config = {
    suites: {
      prihlaseni: ['../at/specs/prihlaseni.spec.ts'],
    },
};
```

Dále je potřeba v konfiguračním souboru nastavit prohlížeč, v kterém se budou vytvořené testy spouštět a počet vytvářených instancí prohlížeče. Pro konfiguraci pouze jednoho konkrétního prohlížeče se využívá parametr `capabilities`. Nastavení počtu instancí konkrétní verze prohlížeče se provádí pomocí parametru `MaxInstances`. [14], [23]

```
exports.config = {
    capabilities: {
      browserName: 'chrome',
    },
    MaxInstances: 1;
};
```

V případě, že se mají testy provádět na více než jednom prohlížeči, je nutné v konfiguraci využít parametru `multiCapabilities`. Při využití více prohlížečů lze konfigurovat také způsob jejich spouštění. Pro nastavení paralelního spouštění prohlížečů slouží parametr `maxSessions`. [24]

```
exports.config = {
    multiCapabilities: [{
      'browserName': 'chrome'
    }, {
      'browserName': 'firefox'
    }]
    maxSessions: 2,
};
```

Posledním vyžadovaným nastavením je nastavení využití testovacího frameworku. Protractor defaultně využívá frameworku Jasmine, ale umožňuje také využití frameworku Mocha či dalších frameworku. Pro nastavení konkrétního frameworku slouží parametr `framework`. Jeho konfiguraci je možné specifikovat parametrem `jasmineNodeOpts`. [25], [27]

```
exports.config = {
    framework: 'jasmine',
    jasmineNodeOpts: {
      showColors: true,
      defaultTimeoutInterval: 300000,
    },
};
```

Konfigurační soubor může být dále doplněn o další parametry. Tyto parametry nejsou povinné, ale umožňují přizpůsobit testovací prostředí požadovaným potřebám. Seznam těchto parametrů je uveden v následující tabulce č. 1.

Parametr	Popis
`baseUrl?: string;`	Nadefinování URL adresy testované aplikace.
`rootElement?: string;`	Nadefinování elementu, na kterém má být vyhledávána Angular aplikace.
`allScriptsTimeout?: number;`	Nastavení timeoutu pro vykonání příkazu.
`getPageTimeout?: number;`	Nastavení timeoutu pro načtení webové stránky.
`beforeLaunch?: () => void;`	Funkce obsahující příkazy, které se provádějí před nastartováním testovacího prostředí.
`onPrepare?: () => void;`	Funkce obsahující příkazy, které se provádějí před spuštěním testů.
`onComplete?: () => void;`	Funkce obsahující příkazy, které se mají provést po dokončení všech spuštěných testů.
`params?: any;`	Definice parametrů, které se využívají v rámci testovacího prostředí.
`restartBrowserBetweenTests?: boolean;`	Nastavení restartování prohlížeče po provedení testu.
`logLevel?: 'ERROR'|'WARN'|'INFO'|'DEBUG';`	Nastavení úrovně logování Protractoru.

Tabulka 1 - Parametry konfiguračního souboru Protractoru [14], [26]

3.5.4 Syntaxe

Protractor využívá kromě vlastní syntaxe i syntaxe z jiných testovacích frameworků. Defaultně se používá syntaxe Jasmine. Dále lze využít např. Mocha, Cucumber, Serenity/JS atd.. Tyto testovací frameworky jsou využity převážně pro funkce `describe`, `it` a `expect`. [14], [27]

```
describe("Testovací sada", function() {
  it("Test", function() {
    expect(true).toEqual(true);
  });
});
```

Funkce `describe` představuje tzv. testovací sadu. Tato testovací sada se může skládat z jednoho či více samostatných testů. Funkci `describe` se předávají dva parametry. Prvním parametrem je textový řetězec, který pojmenovává testovací sadu. Druhým parametrem je `function()`, v rámci které se řeší jednotlivé příkazy. [14], [21]

Funkce `it` slouží pro vytvoření specifického testu a bývá většinou součástí uvedeného `describe`. Tato funkce má shodné parametry s funkcí `describe`, kdy se v textovém řetězci uvádí název konkrétního testu. [14], [21]

Poslední z funkcí `expect` slouží pro ověření očekávaného stavu. V rámci této funkce se ověřuje, zda je skutečná hodnota totožná s očekávanou hodnotou. Skutečná hodnota se uvádí jako parametr funkce `expect`. Pro očekávané hodnoty se využívají tzv. matcher funkce, které mohou být tvořeny `toBe()`, `toEqual()`, `toContain()` a další. [14], [21]

Vlastní syntaxe frameworku Protractor obsahuje převážně příkazy pro získání objektů na webové stránce a jejich následné využití ve formě kliknutí, vkládání hodnot či jejich čtení. V Protractoru se pro získání objektu využívá globální funkce `element`, která je doplněna o tzv. lokátor. Lokátor slouží k identifikaci objektu na webové stránce. [14], [28]

```
element(lokátor);
```

Pro identifikaci objektů nabízí Protractor velké množství lokátorů, které mohou být využity. Seznam používaných lokátorů je uveden v následující tabulce č. 2.

Lokátor	Syntaxe v Protractoru
by.id	element(by.id('contact-email');
by.xpath	element(by.xpath('//ul/li/a'));
by.tagName	element(by.tagName('app-contact'));
By.linkText	element(by.linkText('Add contact'));
by.partialLinkText	element(by.partialLinkText('contact'));
by.css	element(by.css('.contact-email'));
by.cssContainingText	element(by.cssContainingText('button','Přihlásit se'));
by.buttonText	element(by.buttonText('Přihlásit se'));
by.partialButtonText	element(by.partialButtonText('Přihlásit'));
by.model	element(by.model('contact.name'));
by.binding	element(by.binding('contact.name'));

Tabulka 2 - Lokátory pro identifikaci objektů v Protractoru [14], [28]

Nad uvedenými elementy lze využívat příkazů, které umožňují práci s těmito elementy. Tyto příkazy mají simulovat interakci uživatele s aplikací a uvádějí se na konci příkazu pro získání objektu. Jedná se tedy o spojení příkazu pro získání objektu s příkazem pro jeho interakci. V následující tabulce č. 3 je uveden seznam využívaných příkazů. [14], [28]

Interakce s objektem	Popis
`element().click();`	Kliknutí na objekt.
`element().sendKeys("Test");`	Vložení textu např. do inputu.
`element().getText();`	Získání textu z objektu.
`element().clear();`	Smazání textu z objektu.
`element().isPresent();` `element().isElementPresent();`	Testování přítomnosti objektu
`element().getAttribute();`	Získání atributu z objektu.
`element().getSize();`	Získání velikosti objektu.
`element().getLocation();`	Získání souřadnic x a y výskytu objektu.

Tabulka 3 - Příkazy pro práci s objekty v Protractoru [14], [28]

Před samotnou interakcí objektu je v některých případech potřeba využít čekání na dostupnost objektu. Obecné čekání se realizuje příkazem `browser.sleep()`. [14]

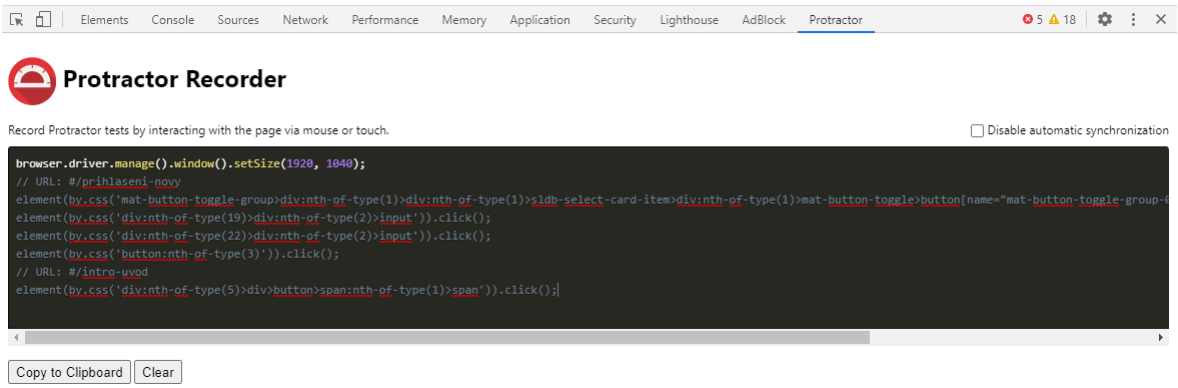
```
browser.sleep(ms: number);
```

Pro testování dostupnosti či nedostupnosti konkrétního objektu se využívá příkaz `browser.wait()`. V rámci tohoto čekání lze testovat několik vlastností objektu, které jsou charakterizovány pomocí `ExpectedConditions`. Při čekání na přítomnost objektu se využívá `presenceOf`. V opačném případě, kdy se čeká na nedostupnost objektu se využívá `stalenessOf`. Dále je možné využít čekání na stav ke kliknutí, který je reprezentován `elementToBeClickable`. Pro čekání na viditelnost či neviditelnost objektu slouží `visibilityOf` nebo `invisibilityOf`. V posledním případě je možné čekat také na to, až bude testovaný element obsahovat určitý text `textToBePresentInElement`. [14], [29]

```
browser.wait(ExpectedConditions.presenceOf(element(by.id('')
)), 5000);
browser.wait(ExpectedConditions.stalenessOf(element(by.id('
'))), 5000);
browser.wait(ExpectedConditions.elementToBeClickable(elemen
t(by.id(''))), 5000);
browser.wait(ExpectedConditions.visibilityOf(element(by.id(
''))), 5000);
browser.wait(ExpectedConditions.invisibilityOf(element(by.i
d(''))), 5000);
browser.wait(ExpectedConditions.textToBePresentInElement(el
ement(by.id('')), 'test'), 5000);
```

3.5.5 Nahrávání testů

V případě Protractoru je potřeba počítat s tím, že tento framework neobsahuje žádnou funkcionalitu, která by umožňovala vytvářet testy pomocí nahrávání interakce uživatele s aplikací. Pro nahrávání testů je však pro tento framework k dispozici několik nástrojů, které jsou dostupné převážně formou rozšíření do webového prohlížeče. Jedním z těchto nástrojů je Protractor Recorder, který je použitelný v prohlížeči Google Chrome. Tento plugin umožňuje nahrávat testy interakcí uživatele na webové stránce pomocí myši nebo klávesnice. Po jeho instalaci se automaticky přidá jako další záložka mezi nástroje pro vývojáře viz obrázek č. 5. [30]



Obrázek 5 – Rozšíření pro nahrávání testů Protractor Recorder (vlastní zpracování)

3.5.6 Reportování

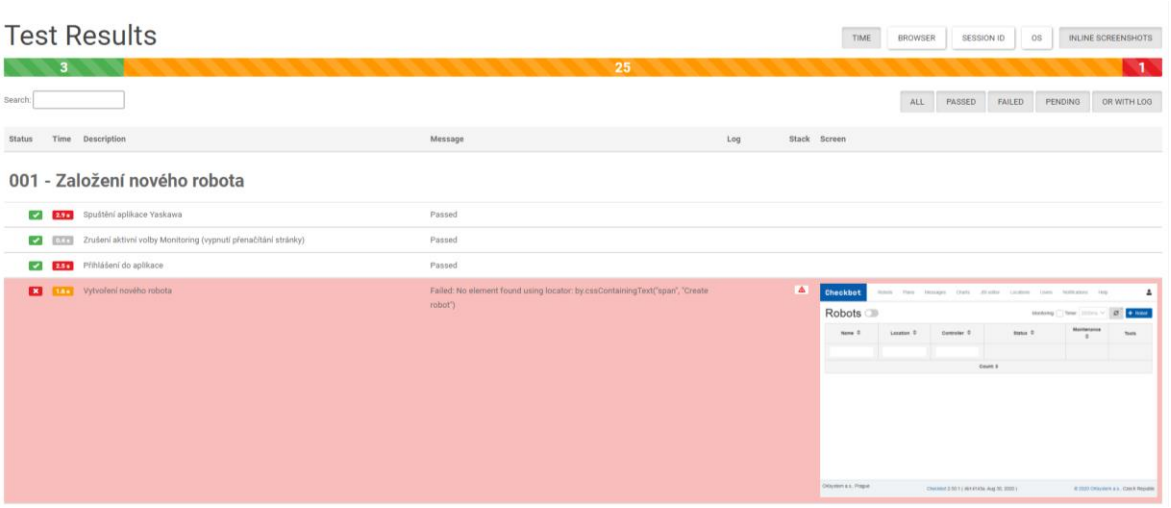
Reportování je v Protractoru řešeno převážně formou pluginů, které jsou nainstalovány pomocí příkazu `npm install`. Pro získání přehledných reportů je tedy nutné si vyhledat vhodný plugin, který zajistí vytvoření reportů z provedených testů. Mezi takovéto pluginy patří například plugin s názvem `protractor-beautiful-reporter`. Instalace tohoto pluginu se provádí prostřednictvím NPM. [31], [32]

```
npm install protractor-beautiful-reporter --save-dev
```

Aktivace reportovacích pluginů se provádí v konfiguračním souboru Protractor ve funkci `onPrepare()`. Pro použití vlastního reportovacího pluginu je nutné použít funkci `jasmine.getEnv().addReporter()` s parametrem předávaného pluginu. Vybraný reportovací plugin může být nakonfigurován pomocí parametrů. Mezi hlavní parametry ke konfiguraci patří adresář, do kterého se mají vytvořené reporty ukládat. Toto nastavení je reprezentováno parametrem `baseDirectory`. Dalším parametrem je nastavení složky k ukládání vytvořených obrazovek pomocí `screenshotsSubfolder`. Posledním parametrem je nastavení vytváření obrazovek pouze v případě testů, které skončí s chybami. Jedná se o parametr `takeScreenShotsOnlyForFailedSpecs`. Defaultně je tato hodnota nastavena na hodnotu `false`, takže se vytvářejí obrazovky i v případě úspěšného dokončení testu. [32]

```
const htmlReporter = require('protractor-beautiful-
reporter');
exports.config = {
     onPrepare() {
          jasmine.getEnv().addReporter(new htmlReporter({
             baseDirectory: 'report',
             screenshotsSubfolder: 'screenshots',
             takeScreenShotsOnlyForFailedSpecs: true,
          }).getJasmine2Reporter());
     }
}
```

Na následujícím obrázku č. 6 je ukázka reportu s chybou, vytvořeného pluginem `protractor-beautiful-reporter`. Každá jednotlivá funkce `it` je v reportu reprezentována jedním řádkem. Na poslední řádce je vypsána chyba a v pravé části je zobrazena obrazovka aplikace, na které byla chyba generována.



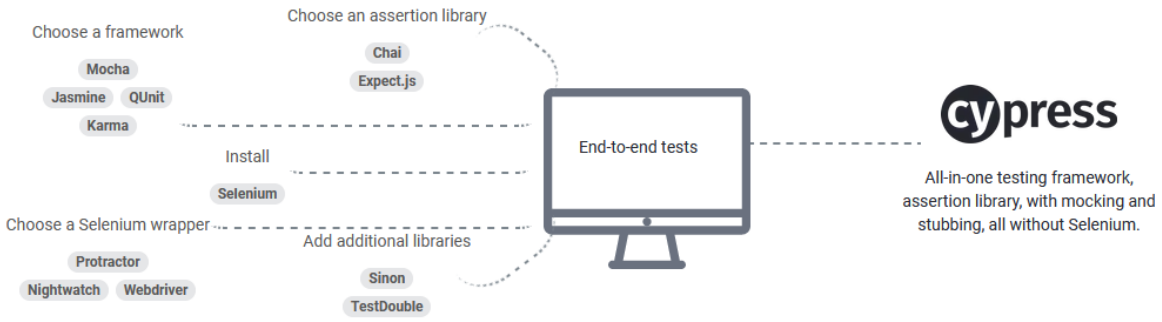
Obrázek 6 - HTML Report z pluginu protractor-beautiful-reporter (vlastní zpracování)

3.6 Cypress

Cypress je testovací framework, který je volně šířen pod licencí MIT. Zaměřuje se na testování aplikací formou funkčního testování. Jako jeden z mála frameworků využívá vlastní architekturu, která není závislá na Seleniu. Jeho hlavní výhodou je jeho jednoduchost, rychlost a univerzálnost. Cílem frameworku je ulehčení testování aplikací z pohledu konfigurace, psaní a spouštění testů. Lze tak použít k testování webových aplikací na jakékoliv platformě. [33]

3.6.1 Architektura frameworku

Cypress využívá vlastní architekturu pro testování webových aplikací. Pro psaní testů se využívá JavaScript, který je po spuštění testů rovnou vykonáván na straně prohlížeče. V rámci samotného testování tedy neprobíhá žádná komunikace mimo webový prohlížeč. Cypress také nevyžaduje pro svůj provoz dodatečnou instalaci dalších nástrojů. Je vytvořen tak, že veškeré potřebné nástroje jsou už automaticky jeho součástí viz obrázek č. 7. [33]



Obrázek 7 - Architektura Cypressu [33]

Cypress spoléhá na nejpoužívanější testovací frameworky a knihovny, které jsou dostupné zdarma. Lze jej považovat za určitou nadstavbu nad frameworkem Mocha. Jako asertovací knihovna se zde využívá knihovna Chai. [33]

3.6.2 Struktura souborů

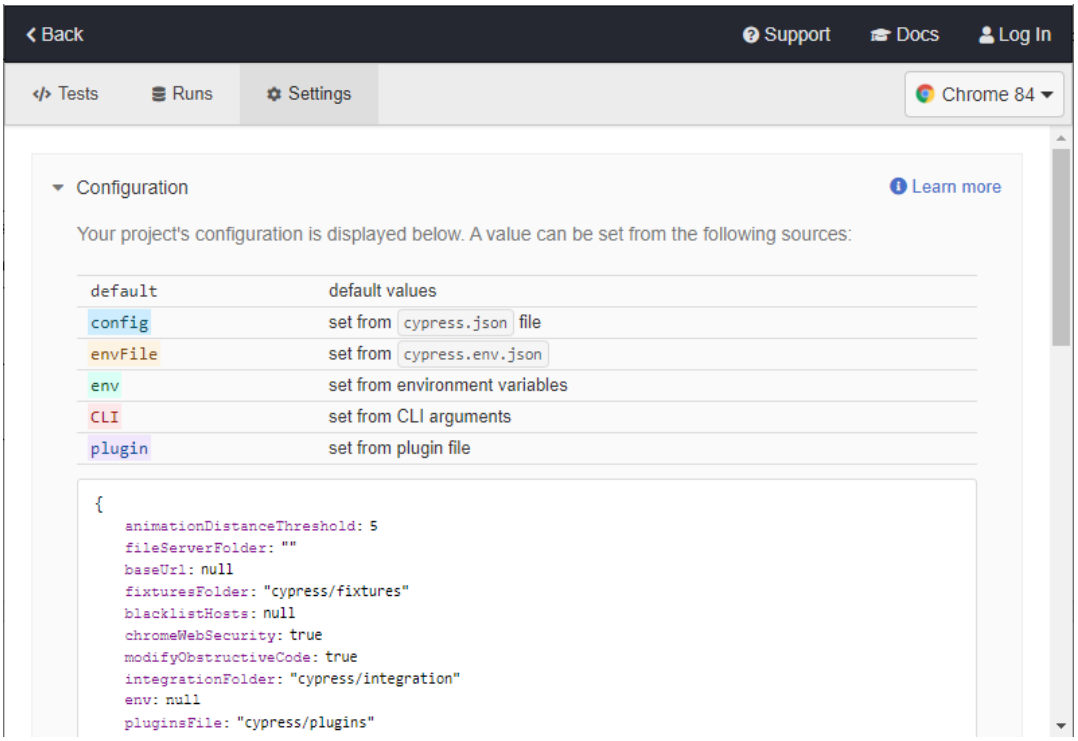
Struktura souborů v Cypressu se skládá z konfiguračního souboru projektu a několika dalších adresářů. Tato struktura je vytvořena automaticky při vytváření nového projektu. [34], [35]

```
/cypress
    /fixtures
    /integration
    /plugins
    /support
- cypress.json
```

Veškerá konfigurace projektu v Cypressu se ukládá do souboru `cypress.json`. Složka `fixtures` slouží pro ukládání statických dat prostřednictvím souboru typu JSON. Testy jsou umístěny ve složce `integration`. Ve složce `plugins` je umístěn soubor `index.js`, v kterém je uveden seznam instalovaných a používaných pluginů. Poslední složka `support` slouží k vytváření vlastních příkazů, které se provádějí před spuštěním testu. Každá z uvedených složek obsahuje po instalaci soubory s příklady užití. [34], [35]

3.6.3 Konfigurace prostředí

Konfigurace prostředí pro Cypress je velmi jednoduchá. Většina parametrů nastavení má po instalaci nastaveny vlastní defaultní hodnoty, takže není zapotřebí žádných rozsáhlých úprav nebo doplnění. Na základě tohoto přístupu ke konfiguraci lze Cypress využívat ihned po vytvoření nového projektu. Pro zobrazení defaultního či vlastního nastavení parametrů lze využít Cypress aplikaci, která je součástí nainstalovaného testovacího frameworku viz obrázek č. 8. Konfigurační soubor Cypressu obsahuje pouze uživatelské nastavení hodnot a defaultní nastavení zde není uvedeno. [37]



Obrázek 8 - Cypress aplikace (vlastní zpracování)

Konfigurace Cypressu obsahuje velké množství parametrů, kterými lze jeho chování přizpůsobit požadavkům. K nim patří například nastavení adresářové struktury projektu. [38]

```
fixturesFolder:"cypress/fixtures"
integrationFolder:"cypress/integration"
pluginsFile:"cypress/plugins"
screenshotsFolder:"cypress/screenshots"
supportFile:"cypress/support"
videosFolder:"cypress/videos"
componentFolder:"cypress/component"
```

Dále lze například nastavit jednotlivé timeouty, na základě, kterých framework čeká na vykonání konkrétních akcí. [38]

```
defaultCommandTimeout:4000
execTimeout:60000
pageLoadTimeout:60000
requestTimeout:5000
responseTimeout:30000
taskTimeout:60000
```

Cypress také sám identifikuje nainstalované prohlížeče a nabídne je automaticky k testování. Kromě toho nabízí prohlížeč Electron, který je defaultním prohlížečem Cypressu a je součástí instalace tohoto frameworku. [39]

```
browsers: Array (4)
0:Chrome
1:Firefox
2:Edge
3:Electron
```

V níže uvedené tabulce č. 4 je uveden seznam dalších nejpoužívanějších parametrů Cypressu.

Parametr	Popis
`animationDistanceThreshold:5`	Vzdálenost pixelů pro považování za animaci.
`baseUrl:null`	URL adresa použitá pro příkazy `cy.visit()` a `cy.request()`
`chromeWebSecurity:true`	Využití nastavení zabezpečení v prohlížeči Chrome.
`modifyObstructiveCode:true`	Nastavení úpravy souborů `.js` nebo `.html` při nevalidním kódu.
`numTestsKeptInMemory:50`	Nastavení počtu testů uchovávaných v paměti počítače.
`testFiles:"**/*.*"`	Definice souborů s testy.
`trashAssetsBeforeRuns:true`	Nastavení promazávání složek se screenshoty a videem před spuštěním testu.
`viewportWidth:1920`	Nastavení šířky okna prohlížeče.
`viewportHeight:1080`	Nastavení výšky okna prohlížeče.
`video:true`	Nastavení nahrávání videa při spuštěném testu.
`videoCompression:32`	Nastavení komprimace videa.
`videoUploadOnPasses:true`	Nastavení ukládání videa i v případě korektně provedených testů.
`screenshotOnRunFailure:true`	Nastavení vytvoření screenshotu při chybovém testu.
`watchForFileChanges:true`	Nastavení sledování změny v testu.
`waitForAnimations:true`	Nastavení čekání na animace objektů.

Tabulka 4 - Parametry konfiguračního souboru Cypressu [38]

3.6.4 Syntaxe

Cypress je určitou nadstavbou nad testovacím frameworkem Mocha. Ten v Cypressu zprostředkovává především funkce `describe` a `it`. Funkci `expect` zajišťuje v Cypressu assertovací knihovna Chai. [36]

Pro identifikaci objektů Cypress využívá příkazu `get` s parametry `selector`, nebo `alias`. Tyto identifikace mohou být dále doplněny o parametr `options`. Tím lze objektu nastavit logování, nebo timeout pro čekání na objekt. [40]

```
cy.get('button');
cy.get('@users');
cy.get('button', options);
cy.get('@users', options);
cy.contains('type');
```

Cypress dále používá několik příkazů pro interakci s objekty. Tyto příkazy se uvádějí vždy na konci příkazu pro získání objektu. Seznam používaných příkazů je uveden v následující tabulce č. 5. [41]

Interakce s objektem	Popis
`cy.get().click();`	Příkaz pro kliknutí na objekt.
`cy.get().dblclick();`	Příkaz, který provede dvojité kliknutí na objekt.
`cy.get().rightclick();`	Příkaz, který simuluje pravé kliknutí myši na objektu.
`cy.get().type();`	Příkaz pro vložení textu objektu.
`cy.get().clear();`	Příkaz pro odstranění textu objektu.
`cy.get().check();`	Příkaz pro zaškrtnutí checkboxu.
`cy.get().uncheck();`	Příkaz pro zrušení zaškrtnutí checkboxu.
`cy.get().select();`	Příkaz pro výběr objektu.
`cy.get().trigger();`	Příkaz pro provedení triggeru nad objektem.
`cy.get().should();`	Příkaz pro ověření stavu objektu.

Tabulka 5 - Příkazy pro práci s objekty v Cypressu [41]

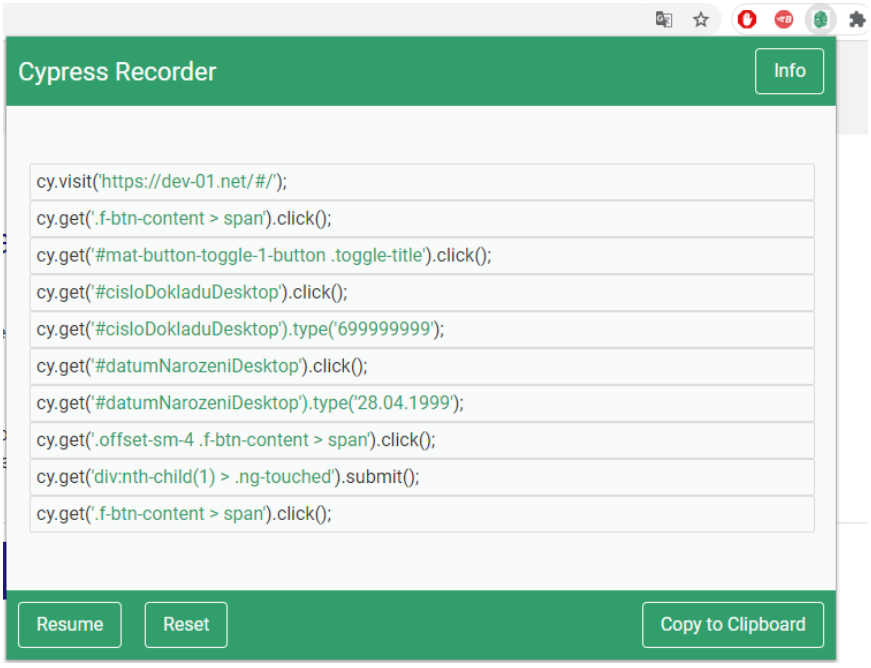
V Cypressu jsou dále k dispozici příkazy pro čekání na použitelnost objektu. V rámci frameworku lze využít buď obecného čekání nebo příkazu pro čekání na použitelnost konkrétního objektu viz tabulka č. 6. [40], [42]

Příkaz	Popis
`cy.wait(time: number);`	Obecné čekání, které je definováno časem v milisekundách.
`cy.get('button', {timeout: 5000});`	Čekání na objekt se selectorem button po dobu 5 vteřin.

Tabulka 6 - Příkazy pro čekání v Cypressu [40], [42]

3.6.5 Nahrávání testů

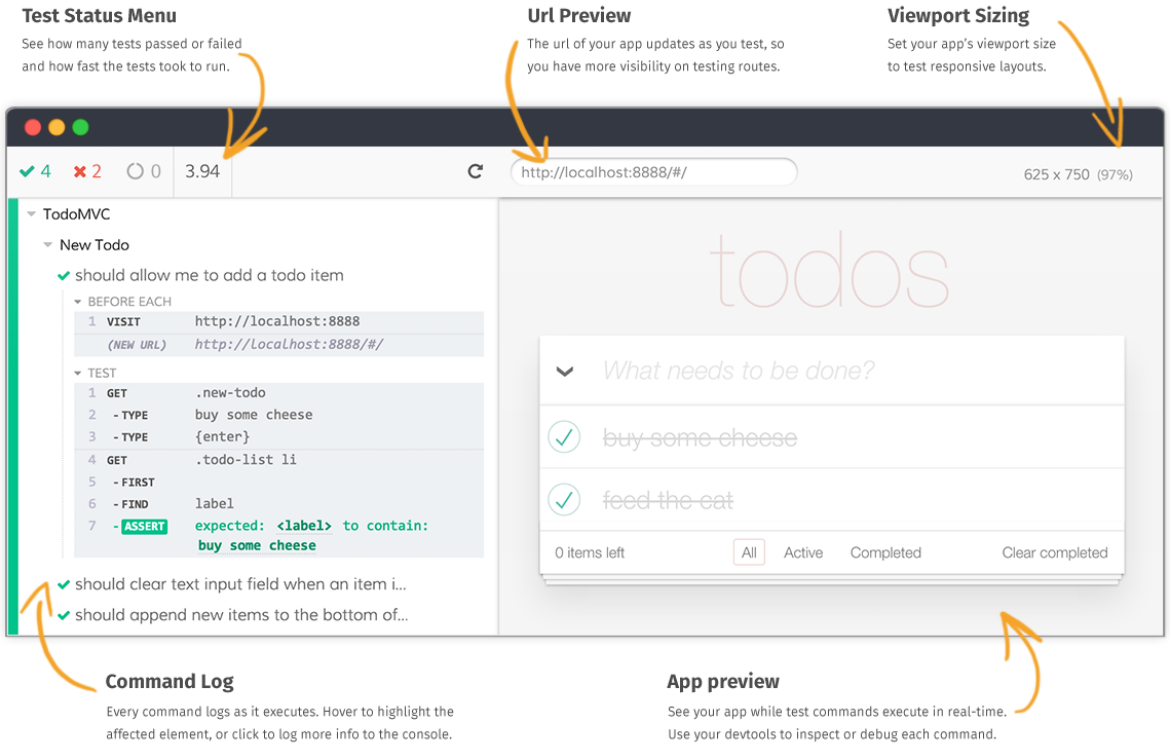
Cypress jako framework neobsahuje žádnou podporu nahrávání testů při interakci uživatele s aplikací. Existují ale pluginy do webových prohlížečů, které tuto funkci poskytují. Jedním z dobře použitelných rozšíření je Cypress Recorder. Ten je po instalaci do prohlížeče dostupný prostřednictvím své ikony v pravém horním rohu okna prohlížeče viz obrázek č. 9. [43]



Obrázek 9 - Rozšíření pro nahrávání testů Cypress Recorder (vlastní zpracování)

3.6.6 Reportování

Pro účely reportování je Cypress vybaven nástrojem Test Runner, který umožňuje sledovat průběh testu při jeho vykonávání. Během testu tak lze sledovat provádění jednotlivých příkazů a zároveň chování testované aplikace. Test Runner je spouštěn automaticky společně s webovým prohlížečem, ve kterém je test vykonáván viz obrázek č. 10. [44]



Obrázek 10 - Cypress Test Runner [44]

Tento nástroj však umožňuje sledovat pouze průběh aktuálně prováděného testu. Pokud je zapotřebí uchovávat výsledky již provedených testů, je v Cypressu možné použít reportovací nástroj nad frameworkem Mocha. Tím může být přímo Mocha reporter, nebo reportér Teamcity či Junit. Tyto reportéry však neposkytují tak komfortní výstupy jako dostupné pluginy. V případě Cypressu bývá často zmiňován plugin `mochawesome`, který je distribuovaný ve formě NPM balíčku. [35]

```
npm install --save-dev mochawesome
```

Nastavení nainstalovaného pluginu jako reportovacího nástroje se provádí v konfiguračního souboru Cypressu prostřednictvím parametru `reporter`. Ten specifikuje plugin požadovaný pro reportování. Konfigurace reportovacího pluginu se nastavuje pomocí parametru `reporterOptions`. V něm lze specifikovat požadované chování reportéru pomocí konfiguračních parametrů uvedených v tabulce č. 7. [45]

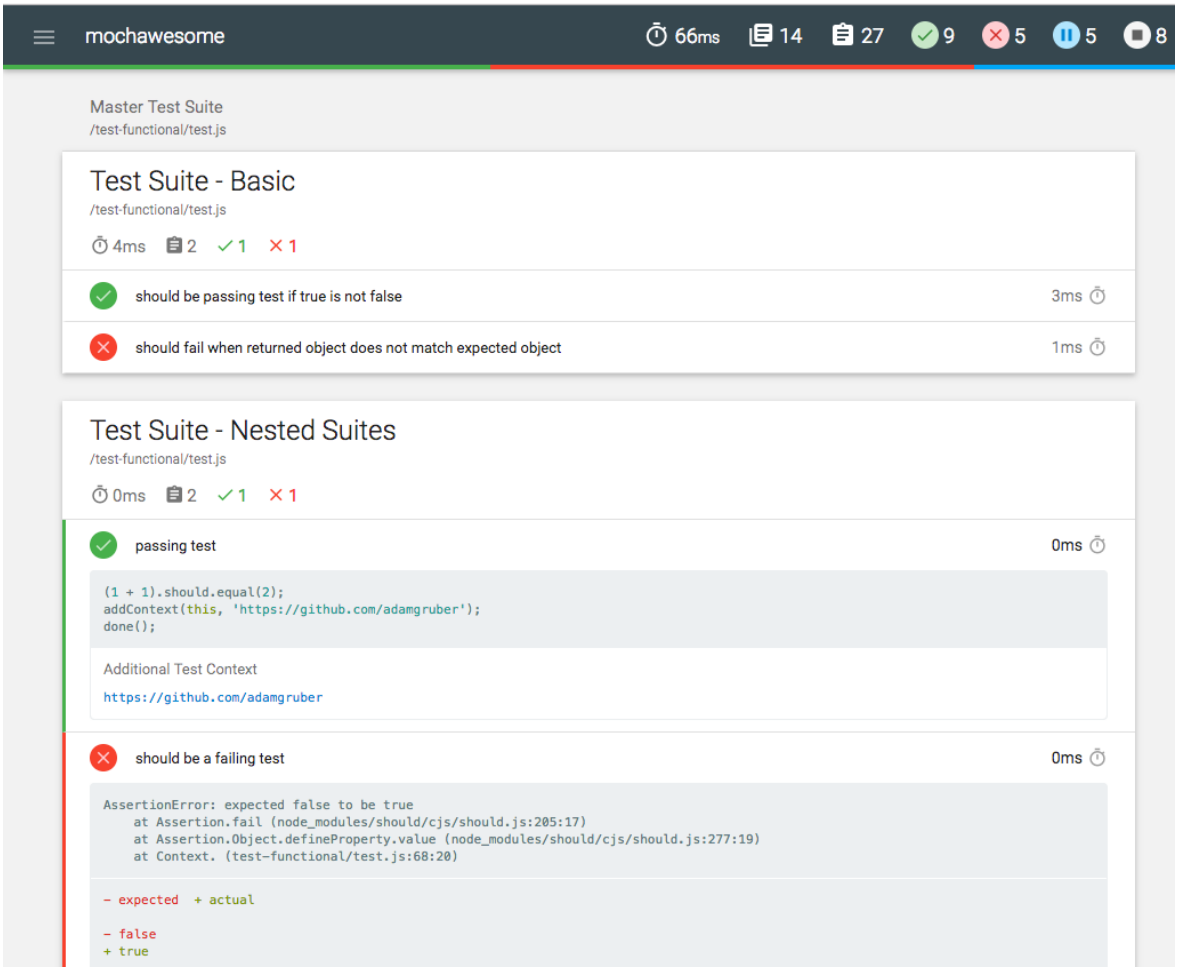
Parametr	Popis
`reportDir`	Nastavení cesty k ukládání reportů.
`overwrite`	Nastavení přepisování vytvořených reportů.
`html`	Nastavení vytváření reportů v podobě html.
`json`	Nastavení vytváření reportů v podobě json.

Tabulka 7 - Konfigurace reportování v Cypressu [45]

Konfigurační soubor obsahující nastavení pluginu pro reportování může vypadat následovně. [45]

```
{
  "reporter": "mochawesome",
  "reporterOptions": {
      "reportDir": "cypress/report/mochawesome-report",
      "overwrite": false,
      "html": true,
      "json": true,
   }
}
```

Na obrázku č. 11 je příklad reportu ve formátu HTML vytvořený pluginem `mochawesome`.



Obrázek 11 - HTML Report z pluginu Mochawesome [45]

3.7 Porovnání frameworků Protractor a Cypress z pohledu jiných autorů

3.7.1 Tymoteusz Kupcewicz

Tymoteusz Kupcewicz hodnotí Protractor jako jeden ze starších testovacích frameworků, který je vhodný převážně pro testování Angular aplikací, a to především proto, že jeho autorem je tým, který zároveň vytvořil i samotný Angular. Upozorňuje však na to, že tento dříve oblíbený a často používaný nástroj začíná ztrácet svou popularitu na úkor ostatních nástrojů. Jako klady tohoto frameworku uvádí práci s Angularem a paralelní práci s prohlížeči. Jako negativa uvádí složitější debuggování a snižující se popularitu. [16]

U Cypressu naopak uvádí, že se jedná o framework, u kterého jeho popularita neustále roste. Framework nabízí uživateli řešení typu vše v jednom, které vyžaduje minimální konfiguraci a není založeno na Seleniu. Jako klady tohoto frameworku uvádí jednoduchou konfiguraci, jednoduché psaní testů, způsob debuggování a velmi přívětivou uživatelskou dokumentaci. Jako negativa naopak uvádí absenci určitých funkcionalit a zpoplatněnou plnou verzi v podobě Cypress Dashboard. [16]

3.7.2 Sahil Goyal

Sahil Goyal ve svém porovnání testovacích frameworků hodnotí Protractor velmi negativně. Jako jeden z hlavních problémů Protractoru uvádí absenci aktualizací. Framework tak nepřináší žádná významná vylepšení, opravy chyb ani žádné aktualizace v oblasti dokumentace. Dalším problémem frameworku je jeho složitost. Protractor přidává tisíce řádků zdrojového kódu nad již obsáhlou knihovnu `selenium-webdriver`, což přináší následně problémy v případě vzniku chyby. Při jejím řešení je velmi složité odhalit kde a proč chyba nastala. Problém Protractoru spočívá také v jeho designu, kdy se ve zdrojovém kódu testů používá nepřímé ovládání objektů na webové stránce. Vytvořené testy jsou plné zřetězených metod a smyček, což může vést ve výsledku k nestabilním a nečitelným testům. Sahil Goyal na závěr hodnocení Protractoru upozorňuje, že framework rozhodně není vyžadován pro testování webových aplikací v Angularu. Vznikl sice jako primární nástroj pro Angular, ale po určitém čase se začaly objevovat i jiné frameworky, které lze v rámci testování na této platformě využít. Největší výhodou Protractoru oproti ostatním nástrojům byla synchronizace s Angularem. Framework si tak dokázal sám ohlídat načtení všech objektů na stránce před provedením konkrétní interakce s objektem. S postupným vývojem Angularu však začala tato funkcionalita ztrácet svoji spolehlivost a pro některé případy je tedy zapotřebí využít čekání s použitím Selenium příkazu. [46]

Cypress je velmi dobrým nástrojem pro testování webových aplikací v průběhu vývoje. Jeho největší výhodou je, že veškeré příkazy jsou spouštěný uvnitř prohlížeče, což přináší více možností pro testování webových aplikací. Napsané testy jsou tak spouštěny a vyhodnocovány přímo v prohlížeči. Výhodou tohoto řešení je bezproblémová interakce s objekty na webové stránce. Problém však může nastat v případě komunikace s backendem, který je reprezentován serverem nebo databází. V tomto případě nelze využít připojení nebo importování knihoven a modulů napřímo. Komunikaci s backendem lze

však vyřešit pomocí Node balíčků. Cypress také neumožňuje paralelní spouštění testů nad více prohlížeči zároveň a neumožňuje ani práci se záložkami prohlížeče. To je zapříčiněno tím, že jsou spouštěné testy vykonávány přímo v samotném prohlížeči. Cypress tedy nemůže v průběhu vykonávání testů ovládat více než jeden konkrétní prohlížeč. [46]

3.7.3 Tulio Castro

Tulio Castro porovnával testovací frameworky podle několika aspektů. Pokud začneme od samotné instalace frameworků, tak zde se samotná instalace obou frameworků od sebe nijak významně neliší, protože se oba frameworky instalují prostřednictvím NPM. Rozdíl však nastává po instalaci, kdy je u Protractoru nutné zvolit testovací framework a asertovací knihovnu. Ve výchozím nastavení se využívá frameworku Jasmine a knihovny Chai. U Cypressu se testovací framework a asertovací knihovna nemusejí řešit, protože jsou již součástí samotného frameworku. [31]

Druhým porovnávaným aspektem byla čitelnost a údržba kódu. Oba frameworky využívají k interakci s objekty různých, ale velmi podobných příkazů. Při porovnání frameworků z pohledu zdrojového kódu je Cypress nepatrně úspornějším frameworkem. Struktura kódu je však v obou případech téměř stejná. [31]

Třetí aspekt řeší, jak moc složité je naučit se pracovat s konkrétním frameworkem. Cypress má k dispozici dokumentaci na jednom konkrétním místě. U Protractoru je nutné procházet různé dokumentace podle řešeného problému. V případě otázek na framework je zapotřebí procházet dokumentaci pro Jasmine. Pokud se jedná o otázky k asertování, tak zde je zapotřebí využít dokumentaci Chai. [31]

Dalšími porovnávanými aspekty byly výkon, reportování a debugování. Podle dostupných benchmarků se zdá být Cypress rychlejším frameworkem než Protractor. Reportování je v Protractoru zapotřebí řešit pomocí pluginů, které jsou dostupné jako Node balíčky. Tyto pluginy je poté nutné nakonfigurovat v konfiguračním souboru testovacího frameworku. Cypress má naopak k dispozici nativní nástroj pro reportování, který je k dispozici při každém spuštění testu. Debugování se v obou frameworcích řeší jiným způsobem. V případě Protracotru je zapotřebí vytvořit konfigurační debugovací soubor a nakonfigurovat vývojové prostředí pro debugování. U Cypressu je debugování daleko jednodušší, protože stačí do zdrojového kódu napsat slovo `debugger` nebo využít funkci `debug()`. [31]

4 Vlastní práce

Vlastní práce je zaměřena na porovnání frameworků Protractor a Cypress nad vnitropodnikovou webovou aplikací na platformě Angular. Pro porovnávání byla vybrána kritéria rychlost a vytížení hardwarových prostředků.

Úvodem praktické části bylo provedeno nakonfigurování frameworků. Po konfiguraci frameworků byly vytvořeny testovací scénáře, které posloužily jako základ k vytvoření automatizovaných testů. Takto vytvořené testy byly poté spouštěny nad webovými prohlížeči Google Chrome, Mozilla Firefox a Microsoft Edge. Získané výsledky z provedených testů posloužily pro porovnání obou frameworků z pohledu kritéria rychlosti. Dále bylo provedeno monitorování vytížení hardwarových prostředků při běžícím testu nad uvedenými prohlížeči. Na základě získaných výsledků bylo na závěr provedeno porovnání a vyhodnocení frameworků.

4.1 Charakteristika aplikace

Pro porovnání testovacích frameworků byla využita vnitropodniková webová aplikace, která se zabývá monitorováním provozu robotů. Podrobnější funkcionalita aplikace je popsána v kapitolách zaměřujících se na tvorbu testovacích scénářů a následných automatizovaných testů.

Architektura této aplikace je rozdělena na frontendovou a backendovou část. Frontendová část byla vytvořena prostřednictvím TypeScriptového frameworku Angular 11. Pro backendovou část byl využit Spring Boot s relačním databázovým systémem H2 v paměti.

4.2 Konfigurace frameworku

4.2.1 Protractor

Veškerá konfigurace frameworku Protractor se provádí prostřednictvím souboru `conf.js`. Pro porovnání frameworků byla využita následující konfigurace.

Úvodem konfigurace bylo specifikováno využití JavaScriptových balíčků. Tyto balíčky se instalují prostřednictvím NPM a jejich cílem je doplnění potřebné funkcionality v testovanému frameworku. V rámci porovnávání frameworků bylo pro Protractor využito následujících balíčků.

```
const { SpecReporter } = require('jasmine-spec-reporter');
const failFast = require('jasmine-fail-fast');
```

Prvním použitým balíčkem je balíček `jasmine-spec-reporter`, který zajišťuje vypisování průběhu testu do console. Druhým použitým balíčkem je balíček `jasmine-fail-fast`, který slouží pro ukončení celého testu v případě výskytu chyby. Chyba může být způsobena chybou testované aplikace nebo chybou vlastního testu.

```
allScriptsTimeout: 30000,
specs: ['./specs/*.spec.ts'],
  suites: {
    vytvoreniRobota: ['./specs/vytvoreni.robota.spec.ts'],
    nacteniDatRobota:
['./specs/nacteni.dat.robota.spec.ts'],
    planMonitoringu:
['./specs/plan.pro.monitoring.spec.ts'],
    planBackup: ['./specs/plan.pro.backup.spec.ts'],
    grafUtilizace: ['./specs/graf.utilizace.spec.ts'],
    grafMonitoring: ['./specs/graf.monitoring.spec.ts'],
    grafMaintenance: ['./specs/graf.maintenance.spec.ts']
  },
```

V rámci konfigurace byl pro všechny skripty nastaven třicetivteřinový timeout čekání na objekt a bylo specifikováno umístění jednotlivých testů. Rovněž byly v rámci konfigurace vytvořeny testovací sady. Dále byly nastaveny prohlížeče, na kterých budou testy spouštěny a způsob komunikace s prohlížeči, který bude použit. Nastavení prohlížečů bylo provedeno v části `capabilities`, kam byly doplněny všechny testované prohlížeče. Test byl spouštěn vždy pouze na jednom z nakonfigurovaných prohlížečů. Aktuálně nepoužité prohlížeče byly v konfiguračním souboru vždy zakomentovány.

```
capabilities: {
   'browserName': 'chrome'
   //'browserName': 'firefox'
   //'browserName': 'MicrosoftEdge'
},
```

Pro prohlížeče Google Chrome a Mozilla Firefox byl způsob komunikace nastaven jako přímé připojení pomocí parametru `directConnect`.

```
directConnect: true,
```

V případě přímého připojení bylo zapotřebí provést stažení ovladačů uvedených prohlížečů. Ke stažení těchto ovladačů byl využit následující terminálový příkaz.

```
webdriver-manager update
```

Pro prohlížeč Microsoft Edge je nastavení způsobu komunikace složitější. Tento prohlížeč nepodporuje přímý přístup pomocí parametru `directConnect`, ale je nutné použít komunikaci prostřednictvím Selenium Serveru. Pro spuštění testů v tomto prohlížeči byl stažen ovladač WebDriver určený pro tento prohlížeč z oficiálních webových stránek Microsoftu. Stažený ovladač byl následně umístěn v rámci projektu do adresáře `webDrivers`. Umístění ovladače bylo specifikováno v konfiguračním souboru pomocí parametru `Dwebdriver.edge.driver`.

```
directConnect: false,
localSeleniumStandaloneOpts: {
  jvmArgs: ['-
Dwebdriver.edge.driver=./webDrivers/msedgedriver.exe'],
  },
autoStartStopServer: true,
maxSessions: 1,
```

Dále byla v konfiguračním souboru definována URL adresa testované aplikace a framework použitý pro testování aplikace včetně jeho konfiguračních parametrů.

```
baseUrl: 'http://test:9090/#/',
framework: 'jasmine',
jasmineNodeOpts: {
  showColors: true,
  defaultTimeoutInterval: 30000,
},
```

Poslední část konfiguračního souboru obsahuje funkci `OnPrepare()`, která zajišťuje veškerou inicializaci prostředí před spuštěním samotných testů. V této části je realizováno využití JavaScriptových balíčků, které byly deklarovány v samotném úvodu konfigurace.

```
onPrepare() {
  require('ts-node').register({
    project: require('path').join(__dirname,
'./tsconfig.e2e.json')
  });
  jasmine.getEnv().addReporter(new SpecReporter({ spec: {
displayStacktrace: true } }));
  jasmine.getEnv().addReporter(failFast.init());
}
```

4.2.2 Cypress

V Cypressu je veškerá konfigurace umístěna v souboru `cypress.json`. Konfiguraci velmi usnadňuje aplikace Cypress, která je součástí instalace frameworku. V této aplikaci je možné dohledat veškeré výchozí hodnoty konfigurace a v případě potřeby je změnit. Pro porovnání frameworků byla použita následující konfigurace.

```
{
  "baseUrl":"http://test:9090/#/",
  "defaultCommandTimeout":30000,
  "pageLoadTimeout":30000,
  "viewportWidth":1600,
  "viewportHeight":1200,
  "video":false
}
```

Pro testy byla nastavena defaultní URL adresa testované aplikace. Dále byl konfigurován timeout jednotlivých příkazů a načítání webových stránek na hodnotu 30 vteřin. Pro prohlížeč bylo definováno stejné rozlišení jako u Protractoru. Cypress má defaultně zapnuto nahrávání videa ze spuštěných testů. Jelikož Protractor tuto funkci neobsahuje, bylo toto nahrávání pro účely porovnání vypnuto.

4.3 Testovací scénáře

Součástí této kapitoly je vytvoření testovacích scénářů, které poslouží jako základ pro tvorbu automatizovaných testů. Tyto testovací scénáře byly vytvořeny na základě funkcí poskytovaných testovanou aplikací a jejich cílem je otestovat jednotlivé základní funkčnosti, které aplikace nabízí.

4.3.1 Vytvoření robota

Krok	Akce	Očekávaný výsledek
1	Na hlavní stránce klikněte na: *Login*.	Je zobrazen formulář pro zadání přihlašovacích údajů.
2	Vyplňte uživatelské jméno s heslem a přihlaste se do aplikace pomocí tlačítka: *Sign in*.	Došlo k přesměrování na hlavní stránku aplikace.
3	Klikněte na tlačítko: *Create robot*.	Je zobrazen formulář pro zaevidování nového robota.
4	Vyplňte libovolný název robota.	Je vyplněn název robota.
5	Vyberte příslušný model robota.	Je vybrán model robota.
6	Vyberte transfer robota.	Je vybrán transfer robota.
7	Vyplňte IP adresu robota.	Je vyplněna IP adresa robota.
8	Vyplňte *Serial number* robota.	Je vyplněno *Serial number* robota.
9	Vyberte lokaci robota: *Default location*.	Je vybrána lokace *Default location*.
10	Vyplňtě *Query once per ms*.	Je vyplněno *Query once per ms*.
11	Vyplňte maximální počet připojení.	Je vyplněn maximální počet připojení.
12	Klikněte na tlačítko *Save*.	Došlo k uložení robota a přesměrování na hlavní stránku aplikace.

Tabulka 8 - Testovací scénář Vytvoření robota (vlastní zpracování)

4.3.2 Načtení dat robota

Krok	Akce	Očekávaný výsledek
1	Na hlavní stránce klikněte na: *Login*.	Je zobrazen formulář pro zadání přihlašovacích údajů.
2	Vyplňte uživatelské jméno s heslem a přihlaste se do aplikace pomocí tlačítka: *Sign in*.	Došlo k přesměrování na hlavní stránku aplikace.
3	Klikněte na název založeného robota.	Je zobrazen formulář zobrazující detailní údaje o robotovi.
4	Vyberte záložku *Read data from robot*.	Je vybrána záložka *Read data from robot*.
5	Na záložce *Named signals* klikněte na: *Update signals from file*.	Došlo k zobrazení tabulky se signály robota.
6	Ověřte, že se načetly údaje do tabulky se signály robota.	Je ověřeno, že byly načteny údaje do tabulky se signály robota.
7	Vyberte podzáložku *Named registers*.	Je vybrána podzáložka *Named registers*.
8	Klikněte na tlačítko: *Update registers from file*.	Došlo k zobrazení tabulky s registry robota.
9	Ověřte, že se načetly údaje do tabulky s registry robota.	Je ověřeno, že byly načteny údaje do tabulky s registry robota.
10	Vyberte podzáložku *Named variables*.	Je vybrána podzáložka *Named variables*.
11	Klikněte na tlačítko: *Update variable names from file*.	Došlo k zobrazení tabulky s proměnnými robota.
12	Ověřte, že se načetly údaje do tabulky s proměnnými robota.	Je ověřeno, že byly načteny údaje do tabulky s proměnnými robota.

Tabulka 9 - Testovací scénář Načtení dat robota (vlastní zpracování)

4.3.3 Plán pro monitoring

Krok	Akce	Očekávaný výsledek
1	Na hlavní stránce klikněte na: *Login*.	Je zobrazen formulář pro zadání přihlašovacích údajů.
2	Vyplňte uživatelské jméno s heslem a přihlaste se do aplikace pomocí tlačítka: *Sign in*.	Došlo k přesměrování na hlavní stránku aplikace.
3	Na hlavní stránce klikněte na: *Plans*.	Je zobrazen formulář plánů.
4	Klikněte na tlačítko: + *Monitoring plan*.	Je zobrazen formulář pro zaevidování nového monitoringu.
5	Vyplňte libovolný popis monitoringu.	Je vyplněn popis monitoringu.
6	Vyberte robota k monitoringu.	Je vybrán robot k monitoringu.
7	Vyplňte čas monitorování robota.	Je vyplněn čas monitorování robota.
8	Přidejte nové proměnné *PocetKusu* a *PocetUkonu*.	Jsou přidány nové proměnné *PocetKusu* a *PocetUkonu*.
9	Klikněte na tlačítko *Save*.	Došlo k uložení plánu monitoringu a přesměrování na hlavní stránku aplikace.

Tabulka 10 - Testovací scénář Plán pro monitoring (vlastní zpracování)

4.3.4 Plán pro backup

Krok	Akce	Očekávaný výsledek
1	Na hlavní stránce klikněte na: *Login*.	Je zobrazen formulář pro zadání přihlašovacích údajů.
2	Vyplňte uživatelské jméno s heslem a přihlaste se do aplikace pomocí tlačítka: *Sign in*.	Došlo k přesměrování na hlavní stránku aplikace.
3	Na hlavní stránce klikněte na: *Plans*.	Je zobrazen formulář plánů.
4	Klikněte na tlačítko: + *Backup plan*.	Je zobrazen formulář pro zaevidování nového monitoringu.
5	Vyplňte libovolný popis zálohování.	Je vyplněn popis zálohování.
6	Vyberte robota k zálohování.	Je vybrán robot k zálohování.
7	Vyberte interval zálohování: *Every day*.	Je vybrán interval zálohování *Every day*.
8	Vyplňte libovolný čas zálohování.	Je vyplněn čas zálohování.
9	Klikněte na přidání typu souboru a vyberte všechny dostupné typy.	Je přidán seznam souborů k zálohování.
	Klikněte na tlačítko *Save*.	Došlo k uložení plánu zálohování a přesměrování na hlavní stránku aplikace.

Tabulka 11 - Testovací scénář Plán pro backup (vlastní zpracování)

4.3.5 Graf - Utilizace

Krok	Akce	Očekávaný výsledek
1	Na hlavní stránce klikněte na: *Login*.	Je zobrazen formulář pro zadání přihlašovacích údajů.
2	Vyplňte uživatelské jméno s heslem a přihlaste se do aplikace pomocí tlačítka: *Sign in*.	Došlo k přesměrování na hlavní stránku aplikace.
3	Na hlavní stránce klikněte na: *Charts*.	Je zobrazen formulář grafů.
4	Klikněte na tlačítko: + *Chart*.	Je zobrazen formulář pro vytvoření nového grafu.
5	Vyplňte libovolný název grafu.	Je vyplněn název grafu.
6	Klikněte na volbu *Utilization*.	Je vybrána volba *Utilization*.
7	Vyberte typ plánu: *Engine hours plan*.	Je vybrán typ plánu *Engine hours plan*.
8	V části rozsah vyberte hodnotu: *This week.*	Je vybrán rozsah *Last 7 days*.
9	V části jednotek klikněte na volbu: *Hour*.	Je vybrána hodnota *Hour*.
	Klikněte na tlačítko *Save*.	Došlo k uložení grafu a přesměrování na hlavní stránku aplikace.

Tabulka 12 - Testovací scénář Graf – Utilizace (vlastní zpracování)

4.3.6 Graf - Monitoring

Krok	Akce	Očekávaný výsledek
1	Na hlavní stránce klikněte na: *Login*.	Je zobrazen formulář pro zadání přihlašovacích údajů.
2	Vyplňte uživatelské jméno s heslem a přihlaste se do aplikace pomocí tlačítka: *Sign in*.	Došlo k přesměrování na hlavní stránku aplikace.
3	Na hlavní stránce klikněte na: *Charts*.	Je zobrazen formulář grafů.
4	Klikněte na tlačítko: + *Chart*.	Je zobrazen formulář pro vytvoření nového grafu.
5	Vyplňte libovolný název grafu.	Je vyplněn název grafu.
6	Vyberte libovolný plán.	Je vybrán plán.
7	Přidejte proměnné *D000 (PocetKusu)* a *D001 (PocetUkonu)*.	Jsou přidány proměnné *D000 (PocetKusu)* a *D001 (PocetUkonu)*.
8	V části rozsah vyberte hodnotu: *Last 7 days*.	Je vybrán rozsah *Last 7 days*.
9	V části jednotek klikněte na volbu: *Hour*.	Je vybrána hodnota *Hour*.
	Klikněte na tlačítko *Save*.	Došlo k uložení grafu a přesměrování na hlavní stránku aplikace.

Tabulka 13 - Testovací scénář Graf – Monitoring (vlastní zpracování)

4.3.7 Graf - Maintenance

Krok	Akce	Očekávaný výsledek
1	Na hlavní stránce klikněte na: *Login*.	Je zobrazen formulář pro zadání přihlašovacích údajů.
2	Vyplňte uživatelské jméno s heslem a přihlaste se do aplikace pomocí tlačítka: *Sign in*.	Došlo k přesměrování na hlavní stránku aplikace.
3	Na hlavní stránce klikněte na: *Charts*.	Je zobrazen formulář grafů.
4	Klikněte na tlačítko: + *Chart*.	Je zobrazen formulář pro vytvoření nového grafu.
5	Vyplňte libovolný název grafu.	Je vyplněn název grafu.
6	Klikněte na volbu *Maintenance*.	Je vybrána volba *Maintenance*.
7	Vyberte typ grafu *Doughnut*.	Je vybrán typ grafu *Doughnut.*
8	Klikněte na tlačítko *Save*.	Došlo k uložení grafu a přesměrování na hlavní stránku aplikace.

Tabulka 14 - Testovací scénář Graf – Maintenance (vlastní zpracování)

4.4 Automatizované testy

Tato kapitola popisuje vytvoření automatizovaných testů za využití frameworků Protractor a Cypress. Testy byly vytvořeny na základě testovacích scénářů uvedených v předchozí kapitole. Jelikož se v rámci jednotlivých testů využívají stejné objekty, bylo provedeno rozdělení testů na soubory obsahující identifikaci objektů na stránce a soubory, které obsahují akce nad těmito objekty.

4.4.1 Úvodní stránka aplikace

Prvním vytvořeným souborem je soubor obsahující identifikaci objektů na hlavní stránce testované aplikace. Z této stránky dochází k přesměrování na přihlašovací dialog tlačítkem Login. Dále se lze přesunout na stránky se založenými plány nebo grafy. Součástí této stránky je také tlačítko pro vytvoření nového robota. Z důvodu

automatického obnovování webové stránky je zde k dispozici také zaškrtávací políčko Monitoring. Jelikož Protractor nedokáže komunikovat s testovací aplikací v případě zapnutého automatického obnovování, bylo ve všech testech před spuštěním samotného testu využito ukončení automatického obnovování pomocí zrušení zaškrtávacího políčka Monitoring.

Identifikace objektů na úvodní stránce je v Protractoru řešena v souboru `main.page.ts` (Příloha č. 1). V Cypressu je tato identifikace řešena v souboru `main.page.js` (Příloha č. 2).

4.4.2 Přihlašovací dialog

Z úvodní stránky aplikace se uživatel přesouvá na přihlašovací dialog. Tento dialog obsahuje objekty k vyplnění přihlašovacích údajů. Jedná se o pole pro vyplnění přihlašovacího jména, hesla a tlačítko pro přihlášení. Součástí vytvořeného souboru je mimo identifikaci objektů také metoda pro přihlášení uživatele do aplikace, která se využívá v každém testu. Tato metoda využívá importu objektu tlačítka *Login* z úvodní stránky aplikace, takže samotné volání metody lze realizovat už na formuláři úvodní stránky aplikace.

Pro přihlašovací dialog byl v Protractoru vytvořen soubor `login.page.ts` (Příloha č. 3). V Cypressu je tento dialog řešen prostřednictvím souboru `login.page.js` (Příloha č. 4).

4.4.3 Dialog pro vytvoření nového robota

V rámci prvního testovacího scénáře dochází k vytvoření nového robota. Pro vytvoření nového robota slouží v aplikaci samostatný dialog, který obsahuje několik povinných údajů k vyplnění. Jedná se o pole pro vyplnění názvu robota, objekty pro výběr modelu robota, způsob transferu a lokaci robota z nabízených možností. Dále jsou zde pole pro vyplnění IP adresy robota, sériového čísla robota, QueryOncePerMs a MaxConnection.

Dialog pro vytvoření nového robota je v Protractoru řešen prostřednictvím souboru `robot.dialog.page.ts` (Příloha č. 5). Cypress řeší tuto stránku v souboru `robot.dialog.page.js` (Příloha č. 6).

4.4.4 Údaje robota

Údaje vytvořeného robota jsou evidovány na samostatné stránce, na které lze provádět akce nad tímto robotem. Pro účely testování se zde využívá objektů tabulek s daty signálů, registrů a proměnných. Tyto tabulky jsou k dispozici na samostatných záložkách a pro aktualizaci jejich údajů se využívá tlačítka *Update*.

Identifikace objektů na této stránce je v Protractoru řešena prostřednictvím souboru `robot.page.ts` (Příloha č. 7). V Cypressu se využívá soubor `robot.page.js` (Příloha č. 8).

4.4.5 Evidence plánů

Dalším vytvořeným souborem je soubor obsahující objekty na stránce s evidencí plánů. Na této stránce je v rámci testů využito pouze dvou objektů, které představují tlačítka pro vytvoření monitorovacího a zálohovacího plánu.

Pro Protractor byl vytvořen soubor `plans.page.ts` (Příloha č. 9). Cypress obsahuje soubor `plans.page.js` (Příloha č. 10).

4.4.6 Zálohovací plán

Formulář pro vytvoření zálohovacího plánu obsahuje několik údajů k vyplnění. Jedná se o pole pro vyplnění názvu, které je následováno výběrem robota, pro kterého bude prováděna záloha. Po výběru konkrétního robota se uživateli nabídne možnost dodatečné konfigurace zálohování. Ta je reprezentována objektem pro výběr frekvence zálohování, polem pro vyplnění konkrétního času zálohování a výběrem typů vzniklých souborů.

Identifikace všech využívaných objektů je v Protractoru řešena v souboru `plan.backup.page.ts` (Příloha č. 11). Pro Cypress byl vytvořen soubor `plan.backup.page.js` (Příloha č. 12).

4.4.7 Monitorovací plán

Monitorovací plán je řešen podobným způsobem jako předchozí plán pro zálohování. U monitorovacího plánu se však zadává pouze monitorovací čas ve vteřinách a vyplňuje se zde seznam monitorovaných proměnných.

Pro tuto stránku byl v Protractoru vytvořen soubor `plan.monitoring.page.ts` (Příloha č. 13). V Cypressu byl vytvořen soubor `plan.monitoring.page.js` (Příloha č. 14).

4.4.8 Evidence grafů

Stejně jako stránka s evidencí plánu je v aplikaci k dispozici také stránka s evidencí grafů. Pro účely testů bude na této stránce využito tlačítko pro vytvoření nového grafu.

Pro tuto stránku byly vytvořeny samostatné soubory pro oba frameworky. Jedná se o soubor `charts.page.ts` v Protractoru (Příloha č. 15) a `charts.page.js` v Cypressu (Příloha č. 16).

4.4.9 Dialog pro vytvoření nového grafu

Pro vytvoření grafu se v aplikaci využívá stejného dialogu pro různé typy grafů. V rámci testů, které se zabývají vytvořením grafů, se využívá pouze jeden soubor obsahující objekty pro všechny typy grafů. V tomto souboru byla řešena identifikace objektů pro vyplnění názvu grafu, výběr typu grafu, přidání proměnných a určení rozsahu grafu.

Identifikace objektů tohoto dialogu je v Protractoru realizována prostřednictvím souboru `create.chart.page.ts` (Příloha č. 17). V Cypressu byl vytvořen soubor `create.chart.page.js` (Příloha č. 18).

4.4.10 Obecné funkce

V rámci samotného testování byly pro některé akce vytvořeny univerzální funkce, které lze využívat napříč celým projektem. Cílem těchto funkcí je usnadnění práce při opakovaném provádění stejné akce nad různými objekty. Jedná se především o funkci `vyberZLabelDropdownuText`, která slouží pro výběr konkrétní hodnoty z dropdownu, který se aktivuje kliknutím na objekt. Dále zde byla vytvořena funkce `vyberZInputDropdownuText`, která slouží stejně jako předchozí funkce k výběru z dropdownu, ale v tomto případě se dropdown aktivuje vložením textu do objektu typu input. Poslední vytvořenou funkcí je `tabulkaPocetRadku`, která vrací počet řádků konkrétní tabulky.

Tyto obecné funkce byly vytvořeny v souborech `util.ts` pro Protractor (Příloha č. 19) a `util.js` pro Cypress (Příloha č. 20).

4.4.11 Test Vytvoření robota

Na základě testovacího scénáře z tabulky č. 8 byl vytvořen automatizovaný test v obou porovnávaných frameworcích. Cílem tohoto testu je vytvořit v aplikaci robota, s kterým se následně pracuje v ostatních testech.

V Protractoru byl test vytvořen v souboru `vytvoreni.robota.spec.ts` (Příloha č. 21). V Cypressu je test součástí souboru `vytvoreni.robota.spec.js` (Příloha č. 22).

4.4.12 Test Načtení dat robota

Na základě testovacího scénáře z tabulky č. 9 byl vytvořen automatizovaný test, který navazuje na předchozí test tím, že prochází jednotlivá data vytvořeného robota. V rámci testu se kontrolují signály, registry a proměnné robota.

V Protractoru byl test vytvořen v souboru `nacteni.dat.robota.ts` (Příloha č. 23). V Cypressu je test součástí souboru `nacteni.dat.robota.js` (Příloha č. 24).

4.4.13 Test Plán pro monitoring

Tento test byl vytvořen na základě testovacího scénáře z tabulky č. 10. Cílem tohoto testu je vytvořit monitorovací plán, který monitoruje data konkrétního robota. V tomto testu se pro vytvoření monitorovacího plánu využívá nově vytvořený robot z prvního testu.

V Protractoru byl test vytvořen v souboru `plan.pro.monitoring.spec.ts` (Příloha č. 25). V Cypressu je test součástí souboru `plan.pro.monitoring.spec.js` (Příloha č. 26).

4.4.14 Test Plán pro backup

Tento test vychází z testovacího scénáře z tabulky č. 11. Jedná se o velmi podobný test předchozímu testu s tím rozdílem, že se v tomto případě nevytváří monitorovací plán, ale vytváří se zálohovací plán robota.

V Protractoru byl test vytvořen v souboru `plan.pro.backup.ts` (Příloha č. 27). V Cypressu je test součástí souboru `plan.pro.backup.js` (Příloha č. 28).

4.4.15 Test Graf – Utilizace

Tento test byl vytvořen na základě testovacího scénáře z tabulky č. 12. Cílem tohoto testu je vytvořit graf typu Utilizace.

V Protractoru byl test vytvořen v souboru `graf.utilizace.spec.ts` (Příloha č. 29). V Cypressu je test součástí souboru `graf.utilizace.spec.js` (Příloha č. 30).

4.4.16 Test Graf – Monitoring

Na základě testovacího scénáře z tabulky č. 13 byl vytvořen test, který vytváří graf z údajů o monitorování.

V Protractoru byl test vytvořen v souboru `graf.monitoring.spec.ts` (Příloha č. 31). V Cypressu je test součástí souboru `graf.monitoring.spec.js` (Příloha č. 32).

4.4.17 Test Graf – Maintenance

Na základě testovacího scénáře z tabulky č. 14 byl vytvořen test, který vytváří graf z údajů o údržbě.

V Protractoru byl test vytvořen v souboru `graf.maintenance.spec.ts` (Příloha č. 33). V Cypressu je test součástí souboru `graf.maintenance.spec.js` (Příloha č. 34).

4.5 Měření rychlosti

Tato kapitola se zaměřuje na měření rychlosti jednotlivých frameworků při využití prohlížečů Google Chrome, Mozilla Firefox a Microsoft Edge. Pro měření rychlosti frameworků bylo využito kvantitativní metriky, která představuje časovou délku trvání provedeného testu v jednotkách sekund. Tato data jsou automaticky součástí výsledku testu, takže pro jejich získání postačí spustit test pomocí příkazu v příkazové řádce. Po dokončení testu je v příkazové řádce vypsán časový interval celého testu.

Měření bylo realizováno na stanici s dostatečným výkonem pro provoz testovacích frameworků viz tabulka č. 15.

Konfigurace	Použitý HW/SW
Procesor	Intel(R) Core(TM) i7-7700 CPU Quad-core (4 Core) 3.60GHz
Paměť	16 GB DDR4, 2400 Mhz
Disk	Intel SSD 600p, M.2 - 256GB
Operační systém	Windows 10 Enerprise

Tabulka 15 - Použitý hardware a software (vlastní zpracování)

Pro účely porovnání byl každý vytvořený test spuštěn desetkrát v každém testovacím frameworku v rámci každého ze tří vybraných webových prohlížečů. Získané časy trvání testů v rámci těchto prohlížečů byly ve výsledku pro účely porovnání zprůměrovány. Pro identifikaci testů ve výsledných tabulkách bude využito zkrácené označení jednotlivých testů viz tabulka č. 16.

Test	Označení
Vytvoření robota	Test1
Načtení dat robota	Test2
Plán pro monitoring	Test3
Plán pro backup	Test4
Graf – Utilizace	Test5
Graf - Monitoring	Test6
Graf – Maintenance	Test7

Tabulka 16 – Zkrácené označení testů (vlastní zpracování)

4.5.1 Google Chrome

Prvním prohlížečem, v kterém bylo provedeno měření automatizovaných testů, byl prohlížeč Google Chrome ve verzi 87. V rámci automatizovaného testování aplikací se jedná o nejpoužívanější webový prohlížeč a veškeré testovací nástroje jsou zaměřeny převážně na tento prohlížeč.

V následující tabulce č. 17 jsou zobrazeny časové výsledky jednotlivých testů v sekundách, které byly získány na základě výsledků spuštěných automatizovaných testů z frameworku Protractor. Získané hodnoty byly za každý test zprůměrovány a poté sečteny. Výsledkem je průměrná doba trvání všech testů nad testovacím frameworkem.

Test	**Pořadí provedených měření [s]**										**Průměr [s]**
	1.	**2.**	**3.**	**4.**	**5.**	**6.**	**7.**	**8.**	**9.**	**10.**	
Test1	4,76	5,03	4,75	4,72	4,68	4,67	5,20	5,06	4,72	4,77	4,8361
Test2	6,78	6,33	6,23	6,59	7,21	6,34	6,28	6,42	6,58	6,66	6,5414
Test3	5,85	6,34	5,80	6,19	6,21	5,78	5,94	5,77	5,93	5,98	5,9785
Test4	4,39	4,26	4,37	4,78	4,23	4,32	4,37	4,37	4,36	4,32	4,3753
Test5	4,48	4,33	4,40	4,31	4,39	4,29	4,43	4,37	4,35	4,29	4,3649
Test6	6,21	5,96	5,98	6,06	6,17	6,11	6,08	6,14	6,33	6,07	6,1126
Test7	3,68	3,52	3,69	3,44	3,53	3,46	3,72	3,51	3,56	3,54	3,5638
Celkem											**35,773**

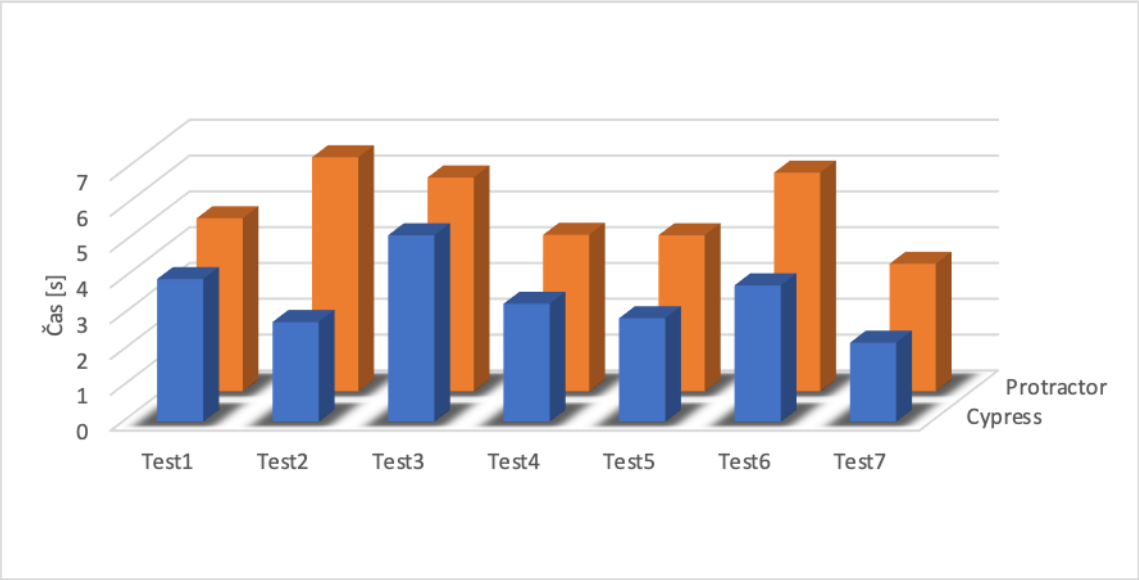
Tabulka 17 - Měření rychlosti nad prohlížečem Google Chrome Protractor (vlastní zpracování)

Stejný proces měření byl realizován také s využitím frameworku Cypress. Naměřená data jsou uvedena v tabulce č. 18.

Test	**Pořadí provedených měření [s]**										**Průměr [s]**
	1.	**2.**	**3.**	**4.**	**5.**	**6.**	**7.**	**8.**	**9.**	**10.**	
Test1	3,91	3,97	3,79	3,92	3,99	3,88	4,12	4,13	4,02	4,08	3,981
Test2	2,81	2,97	2,72	2,78	2,76	2,75	2,77	2,73	2,87	2,65	2,781
Test3	5,7	5,27	5,19	5,13	5,19	5,14	5,1	5,06	5,04	5,22	5,204
Test4	3,66	3,29	3,31	3,44	3,25	3,27	3,16	3,22	3,18	3,12	3,29
Test5	3,42	3,09	2,85	2,76	2,8	2,84	2,87	2,63	2,82	2,78	2,886
Test6	3,83	3,77	3,78	3,83	3,83	3,99	3,7	3,76	3,71	3,86	3,806
Test7	2,78	2,31	2,08	2,22	2,09	2,12	2,1	2,12	2,12	2,09	2,203
Celkem											**24,151**

Tabulka 18 - Měření rychlosti nad prohlížečem Google Chrome Cypress (vlastní zpracování)

Na základě získaných dat z obou testovacích frameworků je patrné, že Cypress byl pro celou sadu testů zhruba o 1/3 rychlejší než Protractor. Pokud se zaměříme na výsledky z jednotlivých testů, tak i zde je vidět, že testy v Cypressu byly ve všech případech rychlejší viz graf na obrázku č. 12. Největší časový rozdíl lze však zaznamenat u druhého testu, kde je rozdíl v průměrných hodnotách přes 3 sekundy. V tomto testu se provádí kontrola počtu záznamů v tabulce, která se zdá být v Protractoru o dost pomalejší než v Cypressu.



Obrázek 12 - Porovnání naměřených hodnot rychlosti nad prohlížečem Google Chrome (vlastní zpracování)

4.5.2 Mozilla Firefox

Dalším prohlížečem, nad kterým bylo provedeno měření automatizovaných testů, byl prohlížeč Mozilla Firefox ve verzi 83. Prvním testovacím frameworkem, který byl měřen nad tímto prohlížečem, byl opět Protractor.

V tabulce č. 19 je možné vidět výsledky z měření jednotlivých testů při využití testovacího frameworku Protractor. Průměrná doba testu se pohybovala zaokrouhleně v rozmezí 4 až 7 sekund. Pokud bychom počítali všechny testy jako jednu celou sadu, tak by průměrná doba trvání celé této sady činila necelých 40 sekund.

Test	Pořadí provedených měření [s]										Průměr [s]
	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	
Test1	5,16	5,09	5,04	5,17	5,46	4,97	5,11	5,12	4,93	5,16	5,1205
Test2	6,58	6,49	6,40	6,48	6,46	6,61	6,43	6,51	6,48	6,54	6,4971
Test3	6,64	6,68	6,91	6,65	6,61	6,80	6,58	6,56	6,64	6,59	6,6677
Test4	5,41	5,59	5,53	5,43	5,40	5,48	5,44	5,47	5,43	5,61	5,4784
Test5	5,38	5,56	5,30	5,29	5,42	5,33	5,34	5,31	5,31	5,30	5,3513
Test6	6,33	6,46	6,27	6,23	6,43	6,45	6,23	6,20	6,40	6,45	6,3436
Test7	4,14	4,14	4,13	4,08	4,13	4,18	4,17	4,20	4,13	4,18	4,1493
Celkem											**39,608**

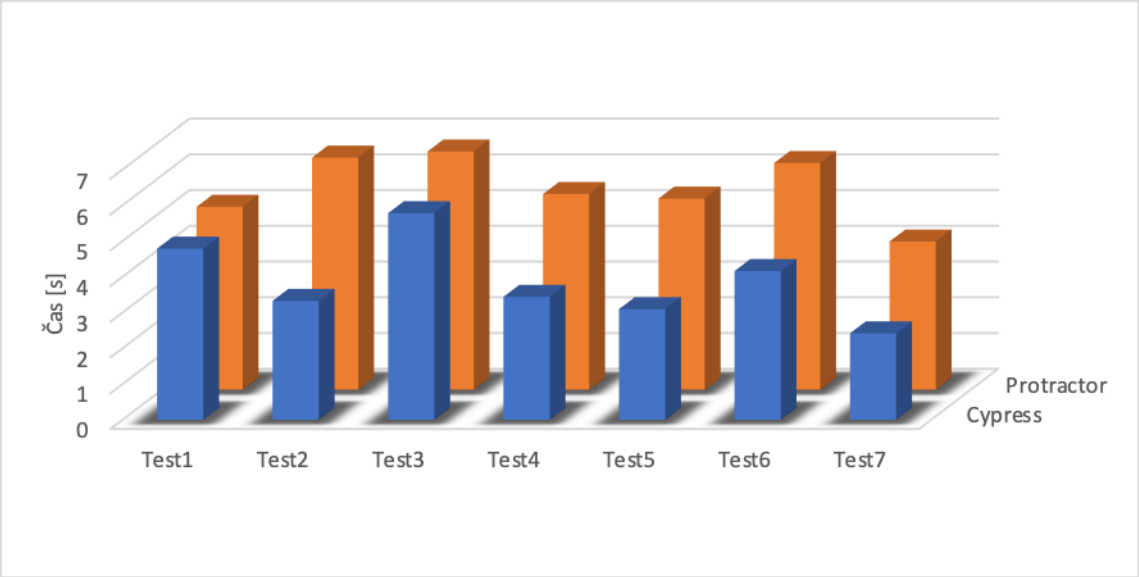
Tabulka 19 - Měření rychlosti nad prohlížečem Mozilla Firefox Protractor (vlastní zpracování)

Dále bylo provedeno měření testů z frameworku Cypress. Data z tohoto měření jsou uvedeny v tabulce č. 20.

Test	Pořadí provedených měření [s]										Průměr [s]
	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	
Test1	4,87	4,68	4,70	4,64	4,89	4,89	4,64	4,88	4,75	4,96	4,79
Test2	3,49	3,41	3,24	3,35	3,28	3,31	3,45	3,47	3,07	3,21	3,328
Test3	5,9	5,84	6,04	5,45	5,71	5,75	5,71	5,81	5,81	5,8	5,782
Test4	3,4	3,42	3,46	3,45	3,4	3,58	3,23	3,52	3,54	3,42	3,442
Test5	3,16	3,11	3,09	3,04	3,04	3,16	3,13	2,99	3,11	3,16	3,099
Test6	4,31	4,21	4,09	3,96	4,06	4,42	4,04	4,21	4,17	4,16	4,163
Test7	2,4	2,4	2,46	2,29	2,38	2,45	2,47	2,51	2,48	2,36	2,42
Celkem											**27,024**

Tabulka 20 - Měření rychlosti nad prohlížečem Mozilla Firefox Cypress (vlastní zpracování)

Z porovnání obou frameworků je opět patrné, že testy v Cypressu byly mnohem rychlejší než testy v Protractoru viz graf na obrázku č. 13. V tomto případě je výsledný rozdíl celé sady skoro deset sekund. Stejně jako u předchozího prohlížeče se zde projevuje výrazný rozdíl u druhého testu, který je zde opět mnohem rychlejší v Cypressu.



Obrázek 13 - Porovnání naměřených hodnot rychlosti nad prohlížečem Mozilla Firefox (vlastní zpracování)

4.5.3 Microsoft Edge

Posledním testovaným prohlížečem byl prohlížeč Microsoft Edge ve verzi 87. V tomto případě je potřeba zmínit, že Protractor neumí pracovat s tímto prohlížečem stejným způsobem jako z předchozími prohlížeči. Pro umožnění spuštění testů nad tímto prohlížečem bylo potřeba v konfiguraci Protractoru zajistit nastartování Selenium Serveru, prostřednictvím kterého byly následně testy provedeny. Výsledky testů z jednotlivých frameworků jsou opět uvedeny v následujících tabulkách.

Test	Pořadí provedených měření [s]										Průměr [s]
	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	
Test1	6,78	6,72	6,64	6,81	6,57	6,70	6,97	6,79	6,77	6,72	6,7457
Test2	7,82	8,05	7,75	7,75	7,81	7,83	7,96	8,14	8,12	8,04	7,9273
Test3	8,31	8,19	7,84	8,43	7,90	7,94	8,13	7,76	7,82	7,82	8,0144
Test4	6,18	5,98	7,03	6,29	6,13	6,32	6,03	6,08	6,36	6,21	6,2591
Test5	6,38	6,29	6,42	6,38	6,34	6,21	6,16	6,08	6,15	6,09	6,2482
Test6	7,57	7,13	6,97	7,94	7,19	6,99	7,47	7,29	7,30	7,27	7,3117
Test7	5,42	5,23	5,14	5,34	5,33	5,22	5,24	5,10	5,21	5,07	5,23
Celkem											**47,736**

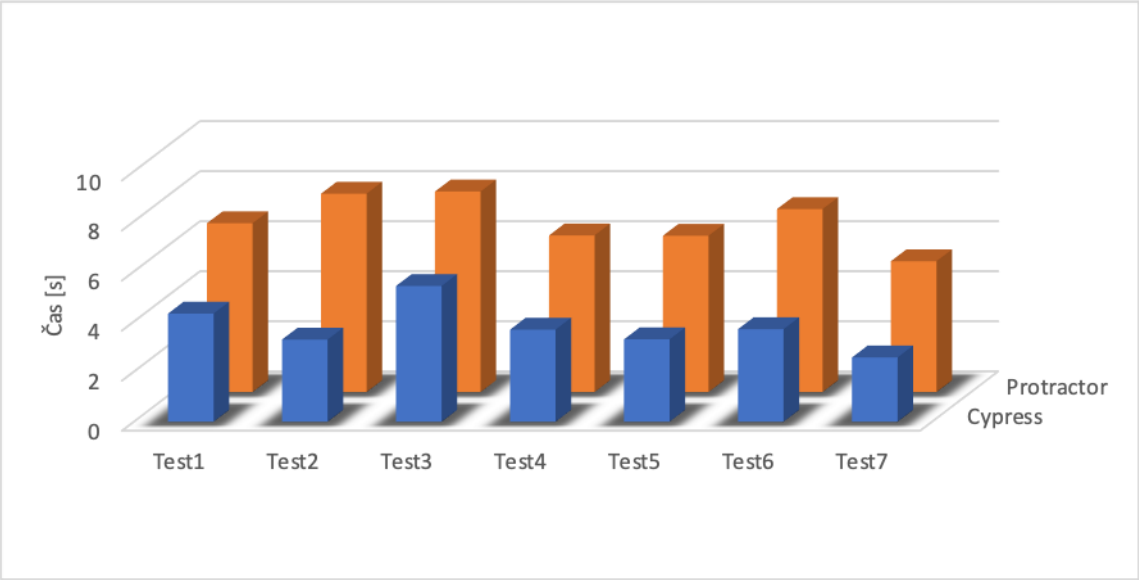
Tabulka 21 – Měření rychlosti nad prohlížečem Microsoft Edge Protractor (vlastní zpracování)

Z výsledků Protractoru uvedených v tabulce č. 21 je patrné, že se jedná o poměrně výrazně pomalejší provedení testů než na předchozích prohlížečích. Nejvyšší průměrná doba testu zde dosahovala 8 sekund. Součet všech průměrných hodnot testů je necelých 48 sekund.

Test	Pořadí provedených měření [s]										Průměr [s]
	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	
Test1	4,32	4,30	4,20	4,32	4,35	4,30	4,31	4,33	4,36	4,41	4,32
Test2	3,28	3,25	3,29	3,2	3,14	3,29	3,5	3,28	3,3	3,31	3,284
Test3	5,4	5,41	5,44	5,31	5,34	5,47	5,41	5,63	5,47	5,37	5,425
Test4	3,62	3,71	3,65	3,61	3,65	3,7	3,69	3,67	3,69	3,82	3,681
Test5	3,45	3,17	3,19	3,22	3,34	3,22	3,35	3,34	3,24	3,41	3,293
Test6	3,77	3,68	3,63	3,66	3,71	3,77	3,77	3,72	3,66	3,62	3,699
Test7	2,62	2,54	2,63	2,6	2,55	2,54	2,53	2,57	2,57	2,54	2,569
Celkem											**26,271**

Tabulka 22 –Měření rychlosti nad prohlížečem Microsoft Edge Cypress (vlastní zpracování)

Výsledky ze Cypressu jsou uvedeny v tabulce č. 22 a opět dokazují, že Cypress je pro testování rychlejším frameworkem. V tomto případě se jedná o největší časový rozdíl na celou sadu. Cypress byl v tomto případě o 18 sekund rychlejší než Protractor. Pokud se podíváme na rozdíl mezi jednotlivými testy, tak každý test v Cypressu byl minimálně o 2 sekundy rychlejší než test v Protractoru viz graf na obrázku č. 14. Výrazně delší doba zpracování testů v Protractoru je v tomto případě zapříčiněna především nepřímou komunikací frameworku s prohlížečem.



Obrázek 14 - Porovnání naměřených hodnot rychlosti nad prohlížečem Microsoft Edge (vlastní zpracování)

4.6 Měření vytížení hardwarových prostředků

Tato kapitola se zaměřuje na měření vytížení hardwarových prostředků v průběhu spuštěného testu nad konkrétním frameworkem a prohlížečem. Pro účely porovnání bylo využito metrik, které představují procentuální vytížení procesoru a vytížení paměti v jednotkách MB/s. Tyto metriky byly získány za použití nástroje SysGauge, který slouží pro monitorování využití hardwarových prostředků a umožňuje měření specifikovat pro jednotlivé procesy. V rámci měření byl využit upravený test pro vytvoření robota. Tento test byl upraven tak, aby v aplikaci neustále vytvářel nové roboty a pro jeho ukončení je nezbytný ruční zásah uživatele. Na základě této úpravy testu bylo umožněno provést měření vytížení hardwarových prostředků po dobu celé jedné minuty.

Samotná realizace měření byla provedena podobným způsobem jako měření rychlosti frameworku z předchozí kapitoly. V tomto případě byl však pro získání výsledků využit pouze jeden konkrétní test, který byl spuštěn opět nad každým z prohlížečů desetkrát.

Pro získání dat k porovnání bylo během provádění testu spuštěno monitorování vytížení hardwarových prostředků, na základě běžících procesů v nástroji SysGauge. V rámci frameworku Protractor byly monitorovány procesy Node, webdriver příslušného prohlížeče a proces tohoto prohlížeče. U Cypressu byly monitorovány pouze proces prohlížeče a proces Cypress. Měření probíhalo v časovém intervalu jedné minuty a třiceti sekund, ale pro výsledné porovnání byly použity hodnoty naměřené pouze v poslední minutě měření. Prvních třicet sekund měření tedy nebylo do porovnání zahrnuto. Toto posunutí bylo provedeno proto, aby výsledky z měření nebyly ovlivněny úvodní inicializací frameworků a prohlížečů.

Výsledky z měření byly pro každý prohlížeč a framework zvlášť zprůměrovány a v případě procesů byly výsledky sečteny do jednoho výstupu.

4.6.1 Google Chrome

První monitorování bylo realizováno nad prohlížečem Google Chrome a frameworkem Protractor. Pro získání požadovaných výstupů bylo monitorování zaměřeno na procesy Node, Chromedriver a Chrome. Výsledky z měření jednotlivých procesů byly sloučeny do jednoho výstupu viz tabulka č. 23.

Měření	Průměr	Minimum	Maximum
Využití procesoru procesem	4.72 %	3 %	7.18 %
Využití paměti procesem	575.2 MB	560.2 MB	588.8 MB

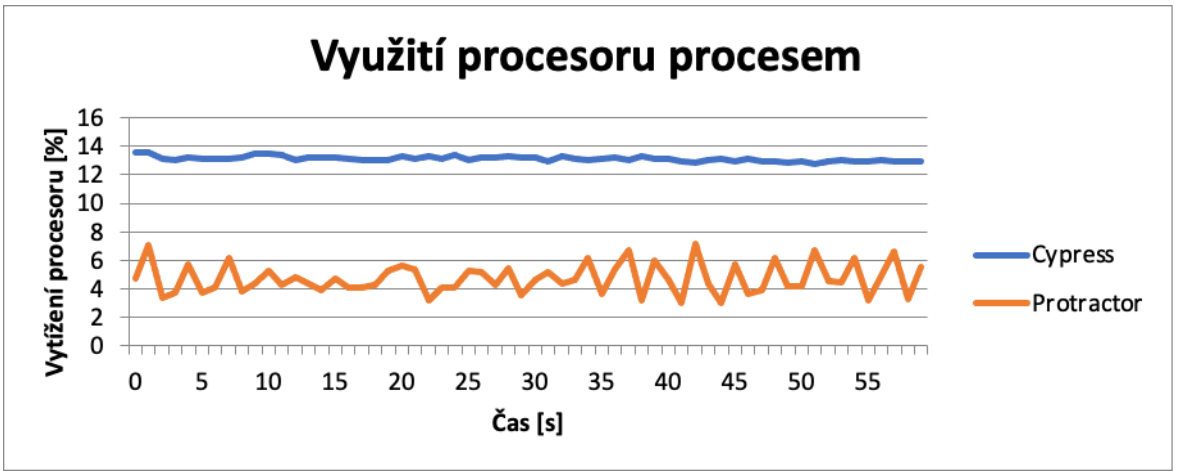
Tabulka 23 - Měření vytížení HW nad Google Chrome Protractor (vlastní zpracování)

Stejné měření bylo realizováno také pro framework Cypress viz tabulka č. 24. V tomto případě však byly monitorovány procesy Cypress a Chrome.

Měření	Průměr	Minimum	Maximum
Využití procesoru procesem	13.13 %	12.78 %	13.59 %
Využití paměti procesem	884.2 MB	853.3 MB	906.5 MB

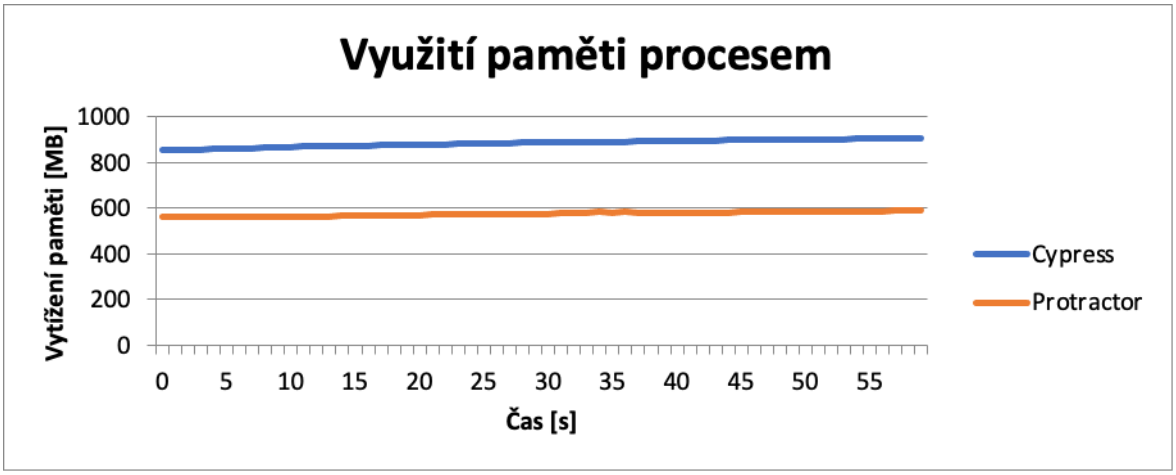
Tabulka 24 - Měření vytížení HW nad Google Chrome Cypress (vlastní zpracování)

Z porovnání uvedených hodnot je patrné, že Cypress má daleko vyšší požadavky na hardwarové prostředky než Protractor. Pokud se zaměříme na využití procesoru procesem, tak rozdíl mezi oběma frameworky v průměrné hodnotě činí až 8 %. Výsledky z měření využití procesoru vystihuje následující graf na obrázku č. 15.



Obrázek 15 - Porovnání vytížení procesoru nad prohlížečem Google Chrome (vlastní zpracování)

V případě vytížení paměti se naměřená data z jednotlivých frameworků v průměrné hodnotě liší o 300 MB. Výsledky opět vystihuje následující graf na obrázku č. 16.



Obrázek 16 - Porovnání vytížení paměti nad prohlížečem Google Chrome (vlastní zpracování)

4.6.2 Mozilla Firefox

Další monitorování bylo realizováno nad prohlížečem Mozilla Firefox. Pro framework Protractor zde byly monitorovány procesy Node, Geckodriver a Firefox. Shrnuté výsledky z měření jsou uvedeny v následující tabulce č. 25.

Měření	Průměr	Minimum	Maximum
Využití procesoru procesem	2.44 %	1.58%	4.05 %
Využití paměti procesem	817.49 MB	769.43 MB	857.45 MB

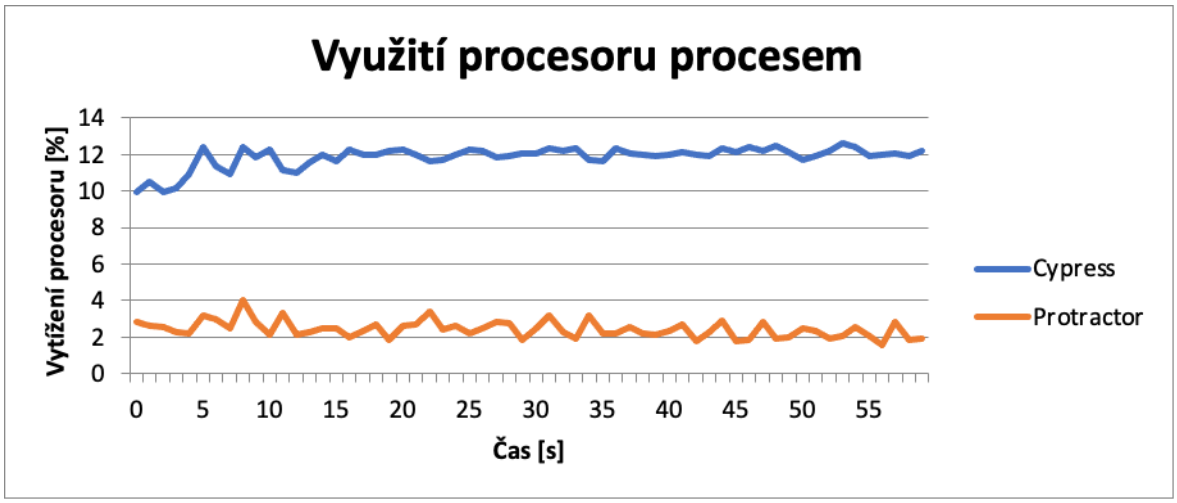
Tabulka 25 - Měření vytížení HW nad Mozilla Firefox Protractor (vlastní zpracování)

V rámci Cypressu bylo monitorování realizováno nad procesy Cypress a Firefox. Výsledky měření Cypressu byly horší než v případě Protractoru viz tabulka č. 26.

Měření	Průměr	Minimum	Maximum
Využití procesoru procesem	11.84 %	9.9 %	12.59 %
Využití paměti procesem	1280.1 MB	1159.2 MB	1333 MB

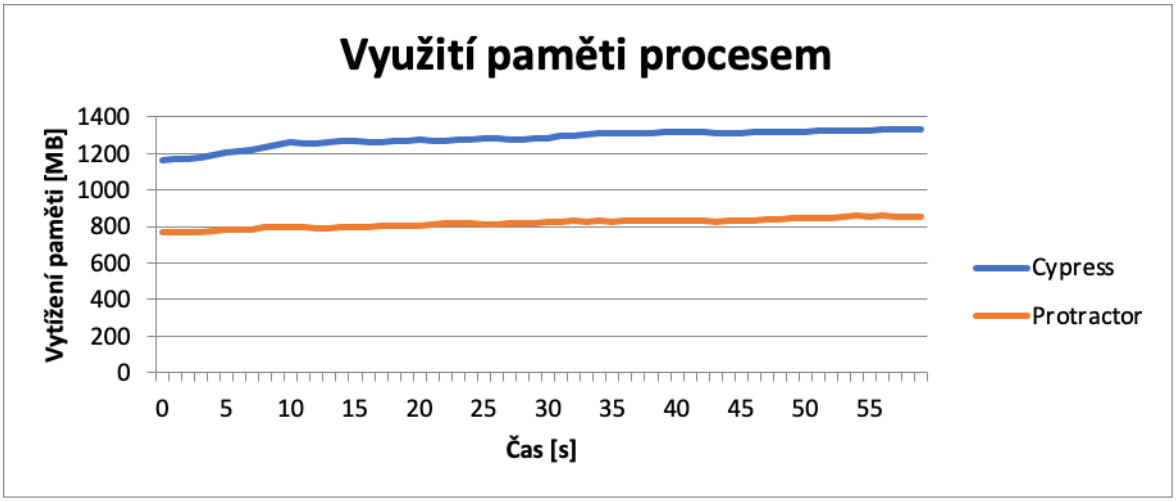
Tabulka 26 - Měření vytížení HW nad Mozilla Firefox Cypress (vlastní zpracování)

Porovnáním jednotlivých hodnot z tabulek lze zjistit, že i v prohlížeči Firefox má Cypress vyšší nároky na hardware viz graf na obrázku č. 17. Rozdíl ve vytížení procesoru je v případě porovnání průměrných hodnot přes 9 %.



Obrázek 17 - Porovnání vytížení procesoru nad prohlížečem Mozilla Firefox (vlastní zpracování)

V případě porovnání vytížení paměti se jedná o rozdíl až 600 MB u průměrných hodnot. Na základě naměřených výsledků lze rovněž konstatovat, že Firefox vytěžoval pamět mnohem více než předchozí měřený prohlížeč viz graf na obrázku č. 18.



Obrázek 18 - Porovnání vytížení paměti nad prohlížečem Mozilla Firefox (vlastní zpracování)

4.6.3 Microsoft Edge

Poslední měření bylo provedeno nad prohlížečem Microsoft Edge. V případě frameworku Protractor byly monitorovány procesy Node, Msedgedriver a Msedge. Základní ukazatele z měření jsou opět součástí tabulky č. 27.

Měření	Průměr	Minimum	Maximum
Využití procesoru procesem	4.72 %	3 %	8.95 %
Využití paměti procesem	702.71 MB	618.2 MB	743.9 MB

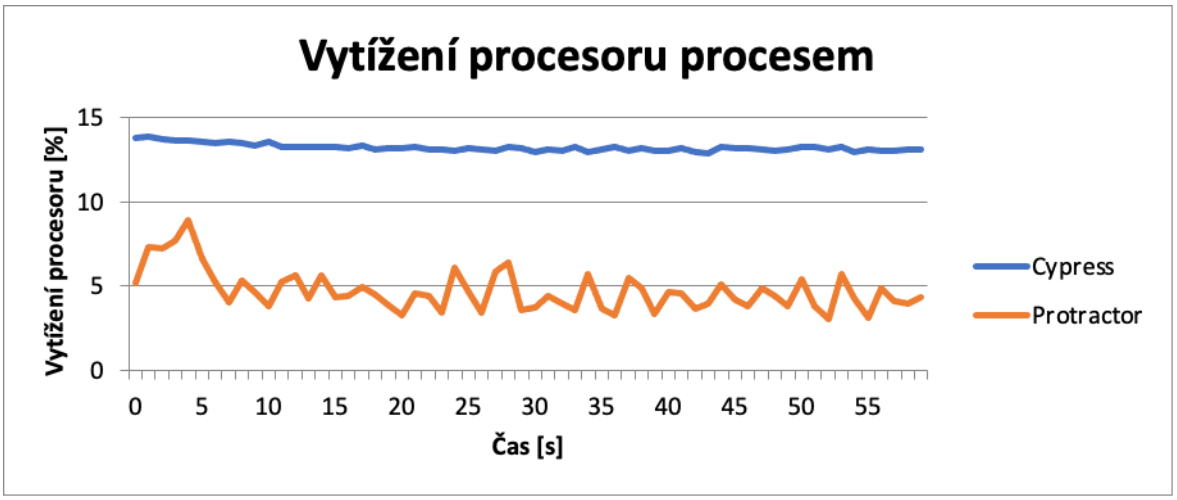
Tabulka 27 - Měření vytížení HW nad Microsoft Edge Protractor (vlastní zpracování)

Pro Cypress byly monitorovány procesy Cypress a Msedge. I v tomto prohlížeči byly výsledky Cypressu horší než v případě Protractoru viz tabulka č. 28.

Měření	Průměr	Minimum	Maximum
Využití procesoru procesem	13.22 %	12.88 %	13.88 %
Využití paměti procesem	885 MB	846.2 MB	912.7 MB

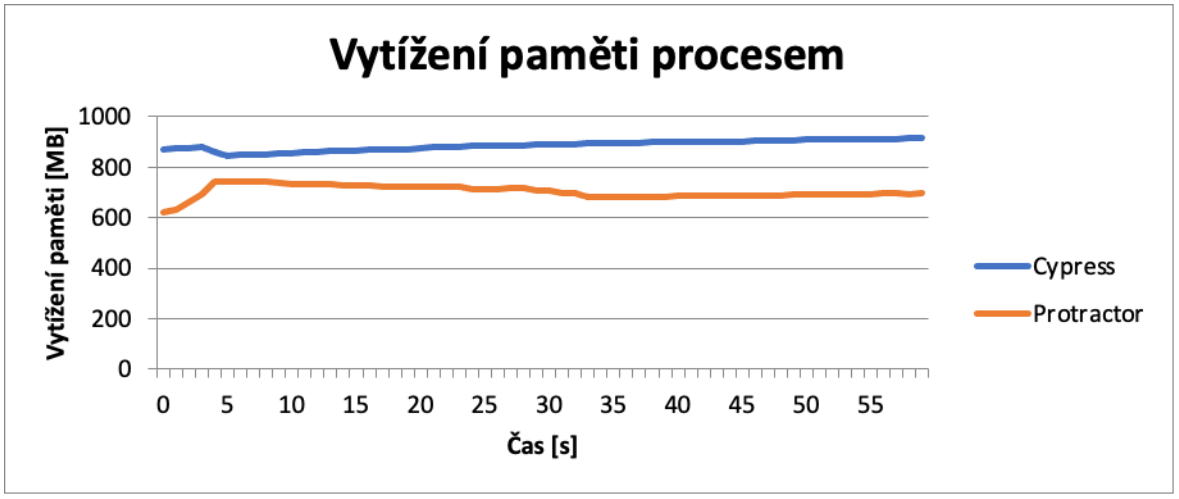
Tabulka 28 - Měření vytížení HW nad Microsoft Edge Cypress (vlastní zpracování)

Výsledky měření prokázaly v případě Protractoru nižší hardwarové požadavky na provoz testů viz graf na obrázku č. 19. V případě vytížení procesoru se zde jedná o rozdíl 8,5 %.



Obrázek 19 - Porovnání vytížení procesoru nad prohlížečem Microsoft Edge (vlastní zpracování)

Rozdíl ve vytížení paměti zde nebyl tak veliký jako u předchozích prohlížečů, což bylo pravděpodobně zapříčiněno jiným způsobem komunikace s prohlížečem v Protractoru. I tak však rozdíl mezi frameworky činil při porovnání průměrných hodnot 180 MB viz graf na obrázku č. 20.



Obrázek 20 - Porovnání vytížení paměti nad prohlížečem Microsoft Edge (vlastní zpracování)

5 Výsledky a diskuse

V této kapitole je provedeno porovnání frameworků na základě naměřených hodnot vybraných kritérií. Pro toto porovnání byla využita vícekriteriální analýza variant formou bodovací metody s vahami, které byly vypočteny pomocí bodovací metody. Před aplikováním vícekriteriální analýzy variant byla zpracována naměřená data tak, aby mohla být aplikována na následné vyhodnocení.

5.1 Naměřená data

Každé kritérium bylo monitorováno nad třemi webovými prohlížeči. Pro účely porovnání byla pro všechny prohlížeče určena stejná důležitost. V praxi totiž bývá častým požadavkem otestování aplikace na nejčastěji používaných webových prohlížečích. U kritéria rychlosti frameworku, kdy se měřil čas úspěšného provedení testů v jednotkách sekund, byla provedena sumarizace těchto výsledků z jednotlivých prohlížečů na základě konkrétního frameworku viz tabulka č. 29.

Rychlost testů [s]	Google Chrome	Mozilla Firefox	Microsoft Edge	Celkem
Protractor	35,773	39,608	47,736	**123,12**
Cypress	24,151	27,024	26,271	**77,45**

Tabulka 29 - Porovnání průměrných rychlostí testů (vlastní zpracování)

V rámci dalšího měření bylo monitorováno vytížení hardwarových prostředků pro procesor a paměť. I v tomto případě je zapotřebí data za jednotlivé prohlížeče sloučit do jednoho výstupu pro konkrétní framework. Zde ovšem není vhodné provádět sumarizaci naměřených dat, ale pro naměřená data v jednotlivých prohlížečích byl použit jejich průměr. V případě vytížení procesoru byl pro porovnání využit průměr z průměrných hodnot naměřených v rámci jednotlivých prohlížečů viz tabulka č. 30.

Vytížení procesoru [%]	Google Chrome	Mozilla Firefox	Microsoft Edge	Průměr
Protractor	4,72	2,44	4,7	**3,95**
Cypress	13,13	11,84	13,22	**12,73**

Tabulka 30 - Porovnání průměrného vytížení procesoru (vlastní zpracování)

Pro vytížení paměti byl použit stejný postup jako u vytížení procesoru, kdy se pro porovnání využil průměr z průměrných hodnot viz tabulka č. 31.

Vytížení paměti [MB/s]	Google Chrome	Mozilla Firefox	Microsoft Edge	Průměr
Protractor	575,2	817,49	702,71	**698,47**
Cypress	884,2	1280,1	885	**1016,43**

Tabulka 31 - Porovnání průměrného vytížení paměti (vlastní zpracování)

5.2 Vícekriteriální analýza variant

Na základě naměřených dat byla provedena vícekriteriální analýza variant pomocí bodovací metody s vahami. Výpočet vah jednotlivých kritériích byl proveden také pomocí bodovací metody. Bodování jednotlivých kritérií a výsledný výpočet byl rozdělen do tří skupin. Tyto skupiny představují různé varianty preferencí měřených parametrů.

Pro vícekriteriální analýzu variant byla využita data, která byla specifikována v předchozí kapitole viz tabulka č. 32.

	Rychlost testů [s]	Vytížení procesoru [%]	Vytížení paměti [MB/s]
Protractor	123,117	3,95	698,47
Cypress	77,446	12,73	1016,43
Váhy	0,62	0,23	0,15

Tabulka 32 - Data pro vícekriteriální analýzu (vlastní zpracování)

5.2.1 Preference rychlosti provedení testů

V prvním výpočtu byla jako nejdůležitější kritérium zvolena rychlost prováděných testů. Toto kritérium bylo ohodnoceno 10 body. Druhým obodovaným kritériem bylo vytížení procesoru, které bylo ohodnoceno 2 body. Posledním kritériem je vytížení paměti, které bylo ohodnoceno 1 bodem. Vypočtené váhy jsou součástí tabulky č. 33.

Kritérium	Body	Váhy
Rychlost testů	10	0,77
Vytížení procesoru	2	0,15
Vytížení paměti	1	0,08

Tabulka 33 - Stanovení vah při preferenci rychlosti provedení testů (vlastní zpracování)

Všechna kritéria z tabulky mají minimalizační povahu a body pro výsledný výpočet byly rozděleny viz následující tabulka č 34.

	Rychlost testů [s]	Vytížení procesoru [%]	Vytížení paměti [MB/s]	Skalární součin
Protractor	6	10	10	**6,92**
Cypress	10	3	5	**8,54**
Váhy	0,77	0,15	0,08	

Tabulka 34 - Vícekriteriální analýza variant při preferenci rychlosti provedení testů (vlastní zpracování)

Na základě výsledku vícekriteriální analýzy variant lze prohlásit, že Cypress je v případě takto zvolených a ohodnocených kritérií vhodnější variantou pro testování vybrané webové aplikace v Angularu.

5.2.2 Preference vytížení hardwarových prostředků

V druhém výpočtu bylo jako nejdůležitější kritérium zvoleno vytížení hardwarových prostředků při vykonávání testů. V tomto případě bylo ohodnoceno kritérium vytížení procesoru 10 body a vytížení paměti 8 body. Zbývající kritérium představující rychlost testů bylo ohodnoceno 3 body. Vypočtené váhy jsou uvedeny v tabulce č. 35.

Kritérium	Body	Váhy
Rychlost testů	3	0,14
Vytížení procesoru	10	0,48
Vytížení paměti	8	0,38

Tabulka 35 - Stanovení vah při preferenci vytížení hardwarových prostředků (vlastní zpracování)

Pro výpočet vícekriteriální analýzy bylo využito stejného obodování jednotlivých výsledků měření jako u předchozího výpočtu. Jedinou změnou v tabulce jsou hodnoty vypočtených vah a výsledný skalární součin viz tabulka č. 36.

	Rychlost testů [s]	Vytížení procesoru [%]	Vytížení paměti [MB/s]	Skalární součin
Protractor	6	10	10	**9,43**
Cypress	10	3	5	**4,76**
Váhy	0,14	0,48	0,38	

Tabulka 36 - Vícekriteriální analýza variant při preferenci vytížení hardwarových prostředků (vlastní zpracování)

Z výsledků vícekriteriální analýzy variant je patrné, že při preferenci vytížení hardwarových prostředků je vhodnějším frameworkem pro testování Protractor.

5.2.3 Preference vyvážení důležitosti měřených parametrů

Poslední výpočet je zaměřen na vyváženost všech kritérií. Pro výpočet vah zde byla kritéria rychlost testů a vytížení procesoru ohodnocena 10 body. Kritérium vytížení paměti bylo ohodnoceno 8 body. Vypočtené váhy jsou uvedeny v následující tabulce č. 37.

Kritérium	Body	Váhy
Rychlost testů	10	0,36
Vytížení procesoru	10	0,36
Vytížení paměti	8	0,29

Tabulka 37 - Stanovení vah při vyvážené preferenci (vlastní zpracování)

Vypočtené hodnoty vah byly opět vloženy do tabulky č. 38 s obodovanými výsledky z měření pro výpočet skalárního součinu.

	Rychlost testů [s]	Vytížení procesoru [%]	Vytížení paměti [MB/s]	Skalární součin
Protractor	6	10	10	**8,57**
Cypress	10	3	5	**6,07**
Váhy	0,36	0,36	0,29	

Tabulka 38 - Vícekriteriální analýza variant při vyvážené preferenci (vlastní zpracování)

Z tabulky vyplývá, že pokud nebudeme výrazně preferovat některé z kritérií, tak je vhodnějším frameworkem pro testování Protractor.

5.3 Shrnutí získaných poznatků

Po shrnutí získaných výsledků z provedených testů vychází v případě porovnání na základě doby trvání jednotlivých testů nad zvoleným typem Angular aplikací Cypress lepší volbou než Protractor. Pro prohlášení, že je Cypress vždy rychlejší variantou než Protractor, by však bylo potřeba rozsáhlejšího pozorování. Dále z provedených testů vyplynulo, že výhoda Cypressu v podobě rychlosti prováděných testů přináší nevýhodu v podobě vyššího vytížení hardwarových prostředků. K výběru konkrétního frameworku pro testování však může posloužit ještě mnoho jiných aspektů než jen rychlost provedení testů nebo vytížení hardwarových prostředků. V následující části bylo provedeno zhodnocení frameworků dle konfigurace, syntaxe, podpory a vývoje. Na závěr byly porovnány zjištěné skutečnosti s názory jiných autorů.

5.3.1 Konfigurace frameworku

Při porovnání obou frameworků z pohledu konfigurace vychází, že konfigurace Protractoru je náročnější než konfigurace Cypressu. Tato skutečnost ovšem není zapříčiněna tím, že by konfigurace Protractoru byla složitá, ale tím, že Cypress lze bez potřeby další konfigurace použít rovnou po jeho instalaci. Tato vlastnost může být pro mnoho uživatelů přínosná, ale má i svá negativa v podobě defaultního spouštění funkcí, které nemusí být aktuálně zapotřebí. Jako příklad lze uvést nahrávání videa z průběhu testů. V tomto případě se jedná o funkčnost, která nemusí být a zpravidla ani není vždy

vyžadována. Pro vyhodnocení testů ve většině případů postačí report, případně vytvořené screenshoty.

Pokud je zapotřebí provést specifickou konfiguraci frameworku, je v případě Protractoru k dispozici základní konfigurační soubor s minimálním defaultním nastavením. Podrobnější konfiguraci je však nutno dohledat v dokumentaci a následně požadované parametry vložit do konfiguračního souboru. Naproti tomu v Cypressu jsou veškeré dostupné konfigurace už součástí jeho instalace. Změnu nastavení lze tak provést pomocí aplikace Cypress, která je součástí instalace frameworku. V této aplikaci se všechny konfigurační parametry přehledně nabízí s přednastavenými defaultními hodnotami. Požadované konfigurační parametry tak lze snadno přenést do konfiguračního souboru a nastavit jejich hodnotu dle potřeby.

V rámci konfigurace je zapotřebí zmínit také nastavení používání jednotlivých prohlížečů. Protractor využívá pro prohlížeče Google Chrome a Mozilla Firefox v podstatě stejnou konfiguraci s přímým přístupem k ovladači prohlížeče. Problém nastává v případě prohlížeče Microsoft Edge, pro který nelze využit stejný přístup k ovladačům prohlížeče jako v předchozím případě. Konfigurace Protractoru pro tento prohlížeč vyžaduje nastartování Selenium Serveru, pomocí kterého dochází ke komunikaci s prohlížečem v průběhu vykonávání testů. Cypress žádnou specifickou konfiguraci pro konkrétní typ prohlížeče nevyžaduje. Sám si dokáže vyhledat instalované prohlížeče a přímo s nimi dokáže pracovat.

5.3.2 Syntaxe

Syntaxe obou frameworků je velmi podobná. Struktura a délka zdrojových kódů jednotlivých testů byla v obou frameworcích téměř stejná. V některých případech měl Cypress menší počet řádků kódu, ale rozdíl byl nepatrný. Hlavní rozdíl je tak především v identifikaci objektů, kde Protractor nabízí mnoho příkazů (např. by.id, by.className, by.buttonText atd.), kterými lze objekty identifikovat. To činí psaní kódu v tomto frameworku náročnějším. Výsledný kód je však snadno čitelný, protože příkaz přímo obsahuje atribut, který je k identifikaci použit. Cypress se naopak snaží počet těchto příkazů redukovat na minimum tím, že využívá příkazu `get()` a `contains()`, kterými lze nahradit většinu způsobů identifikace objektů v Protractoru. V rámci těchto příkazů se

definuje typ atributu a jeho hodnota, která charakterizuje hledaný objekt. Pro specifikaci použitého typu atributu jsou definovány speciální znaky (např. dvojkřížek, tečka atd.).

5.3.3 Podpora a vývoj

Z pohledu podpory by na tom měl být Protractor lépe, protože od jeho vydání uplynulo již přes 8 let. Přestože je v tomto směru Cypress podstatně mladší, je na tom ale v této oblasti mnohem lépe. Pokud porovnáme tutoriály dostupné na webových stránkách obou frameworků, tak tutoriál Cypressu je výrazně lépe zpracovaný a poskytuje daleko podrobnější informace.

Z hlediska intervalu vydávání nových verzí je na tom Cypress ještě mnohem lépe než v případě tutoriálu. Například v průběhu roku 2020 vycházela nová verze každý měsíc. Protractor měl ve stejném roce pouze tři nové verze za celý rok.

5.4 Porovnání získaných zkušeností s názory jiných autorů

Tato část porovnává získané zkušenosti s názory autorů uvedených v teoretické části práce. Některé klady a zápory jednotlivých frameworků byly uvedeny v předchozích kapitolách, a z toho důvodu už nebudou v této kapitole dále řešeny.

U frameworku Protractor se často uvádí, že je nejvhodnějším frameworkem pro testování aplikací v Angularu. V rámci porovnávání frameworků nad Angularem byl zjištěn pouze jeden významný rozdíl mezi frameworky ve prospěch Protractoru, a tím je vytížení hardwarových prostředku při vykonávání testů. Ve všech ostatních případech byly frameworky totožné, nebo byl naopak lepším frameworkem Cypress.

Jako další z výhod Protractoru se uvádí možnost paralelního spouštění testů. Protractor skutečně nabízí možnost spuštění jednotlivých testů paralelně, což může v některých případech ušetřit čas při testování. Jelikož Cypress tuto funkci neposkytuje, nebyla při porovnání frameworků využita.

Protractoru bývá někdy vytýkána úroveň uživatelské dokumentace. Při jeho konfiguraci pro účely této práce bylo zapotřebí kromě oficiální dokumentace Protractoru využít i dalších dostupných zdrojů. Problémem byla například jeho konfigurace pro spouštění testů v prohlížeči Microsoft Edge. V tomto konkrétním případě bylo zapotřebí využít postupů získaných mimo uživatelskou dokumentaci, protože ta se touto konfigurací

téměř nezabývá. U Cypressu naopak nebylo nutné využít jiných zdrojů než dostupné oficiální dokumentace.

Další řešenou oblastí při porovnání frameworků bývá způsob a podpora reportování. Protractor v základu neobsahuje žádné uživatelsky přívětivé reporty a pro jejich získání je zapotřebí instalace pluginů v podobě NPM balíčků. Cypress naopak v základu obsahuje reportér, který poskytuje přehledný výstup zobrazovaný přímo v prohlížeči již při běhu testu. Pokud je zapotřebí reporty uchovávat, lze stejně jako u Protractoru použít dostupných pluginů, nebo zpoplatněnou komponentu Cypress Dashboard.

6 Závěr

Testování hraje významnou roli v oblasti vývoje softwaru. Před jeho zahájením je vždy zapotřebí vybrat vhodný nástroj, kterým bude aplikace testována. Výběr vhodného nástroje může záviset na mnoha aspektech. Jedním z hlavních aspektů je platforma testované aplikace. Při vývoji webových aplikací patří v současnosti mezi oblíbené platformy Angular. K testování aplikací na této platformě je možné využít mnoho nástrojů. Častým nástrojem pro testování nad Angularem je framework Protractor, který byl primárně vyvinut právě pro tuto platformu. V rámci testování webových aplikací je dalším velmi rozšířeným nástrojem framework Cypress. Tento nástroj sice není primárně zaměřen na platformu Angular, ale pro testování aplikací na této platformě je rovněž velmi dobrou volbou.

V teoretické části byla provedena charakteristika základních pojmů v oblasti testování softwaru s následným zaměřením na testování pomocí frameworků Protractor a Cypress. Úvodem teoretické části byly představeny metody testování dle jejich cílů. Poté byly uvedeny jednotlivé způsoby testování. Následně byla představena platforma Angular včetně způsobu testování aplikací na této platformě. Teoretická část se dále věnuje popisu testovacích frameworků Protractor a Cypress z pohledu architektury, konfigurace, syntaxe, nahrávání testů a reportování testů. Závěrem bylo shrnuto porovnání frameworků z pohledu jiných autorů.

Praktická část se zaměřuje na vyhodnocení testů vytvořených v jednotlivých frameworcích. Toto vyhodnocení bylo provedeno na základě vybraných metrik, které byly následně využity pro porovnání testovacích frameworků Protractor a Cypress. Na úvod této části byla provedena konfigurace testovacích frameworků. Poté byly vytvořeny testovací scénáře, podle kterých byly vytvořeny automatizované testy. Následně bylo realizováno měření rychlosti formou spuštění vytvořených testů nad prohlížeči Google Chrome, Mozilla Firefox a Microsoft Edge. Dále bylo realizováno měření vytížení hardwarových prostředků v průběhu vykonávání testů nad uvedenými prohlížeči. Poté bylo provedeno porovnání frameworků na základě výsledků z jednotlivých měření. Dle získaných hodnot bylo zjištěno, že Cypress byl ve všech prohlížečích mnohem rychlejším frameworkem než Protractor. V rámci měření vytížení hardwarových prostředků byl však úspornějším frameworkem Protractor. Výsledky byly dále zpracovány prostřednictvím vícekriteriální analýzy variant s využitím bodovací metody s vahami. Výpočet byl rozdělen do tří skupin s různými preferencemi pro jednotlivá kritéria. Na základě výsledků z vícekriteriální analýzy variant lze prohlásit, že Cypress je vhodnějším frameworkem tam, kde je prioritou

rychlost provádění testů a vytížení hardwarových prostředků není omezujícím faktorem. V ostatních případech, kdy není prioritou rychlost, je důležité vytížení hardwarových prostředků, nebo mají všechna kritéria stejnou prioritu, je vhodnějším frameworkem Protractor. Další část se zaměřila na vyhodnocení získaných poznatků v oblasti konfigurace, syntaxe, podpory a vývoje. Na závěr byly získané zkušenosti porovnány s názory jiných autorů zmíněných v teoretické části této práce.

7 Seznam použitých zdrojů

1. ŽÁRA, Ondřej. JavaScript: programátorské techniky a webové technologie. Computer Press, 2015. ISBN: 978-80-251-4573-9.
2. STEPHENS, Matt; ROSENBERG, Doug. Testování softwaru řízené návrhem. Computer Press, 2017. ISBN: 978-80-251-3607-2.
3. PAGE, Alan; ROLLISON, Bj; JOHNSTON, Ken. Jak testuje software Microsoft. Computer Press, Albatros Media as, 2017. ISBN: 978-80-251-2869-5.
4. MALKUSOVÁ, Tereza. Testování použitelnosti díl 2: Testuji, tedy jsem. *AITOM* [online]. 2015 [cit. 2020-12-03]. Dostupné z: https://www.aitom.cz/co-je-noveho/testovani-pouzitelnosti-dil-2
5. SOMMERVILLE, Ian. Softwarové inženýrství. Computer Press, Albatros Media as, 2017. ISBN: 978-80-251-3826-7.
6. ROUDENSKÝ, Petr; HAVLÍČKOVÁ, Anna. Řízení kvality softwaru. Computer Press, Albatros Media as, 2017. ISBN: 978-80-251-4519-7.
7. NIDHRA, Srinivas; DONDETI, Jagruthi. Black box and white box testing techniques-a literature review. International Journal of Embedded Systems and Applications (IJESA) [online], 2012 [cit. 2020.12.05]. Dostupné z: https://www.academia.edu/13570104/BLACK_BOX_AND_WHITE_BOX_TESTING_TECHNIQUES_A_LITERATURE_REVIEW
8. HUSSAIN, Taraq; SINGH, Satyaveer. A Comparative Study of Software Testing Techniques Viz. White Box Testing Black Box Testing and Grey Box Testing. (IJAPRR), 2015. ISSN: 2350-1294.
9. SELECKÝ, Matúš. Penetrační testy a exploitace. Computer Press, Albatros Media as, 2017. ISBN: 978-80-251-3752-9.
10. BECK, Kent. Programování řízené testy. 1. vydání. Praha: Grada, 2004. Moderní programování. ISBN 80-247-0901-5.
11. DZIWOKI, Michał. What's the difference between AngularJS and Angular? *Gorrion* [online]. 2019 [cit. 2021-01-05]. Dostupné z: https://gorrion.io/blog/angularjs-vs-angular
12. CHAPLIN, Tim. AngularJS Test-driven Development. Packt Publishing Ltd, 2015. ISBN 978-1-78439-883-5.

13. LIM, Greg. Beginning Angular 2 With Typescript. CreateSpace, Scottsdale, 2017. ISBN: 978-1-5429-1667-7.
14. PALMER, Jesse, et al. Testing Angular Applications. Manning, 2018. ISBN: 978-1-61729-364-1.
15. DARWIN, Peter Bacon; KOZLOWSKI, Pawel. Mastering Web Application Development with AngularJS. Packt Publishing Ltd, 2013. ISBN 978-1-78216-182-0.
16. KUPCEWICZ, Tymoteusz. Which JavaScript UI Testing Framework to Use in 2020? *Netguru* [online]. 2020 [cit. 2021-01-07]. Dostupné z: https://www.netguru.com/codestories/which-javascript-ui-testing-framework-to-use-in-2020
17. Cypress vs Protractor vs Selenium-WebDriver vs TestCafe vs WebdriverIO vs Nightwatch. *Npm trends* [online]. 2021 [cit. 2021-03-10]. Dostupné z: https://www.npmtrends.com/cypress-vs-protractor-vs-selenium-webdriver-vs-testcafe-vs-webdriverio-vs-nightwatch
18. ENISHETTI, Ranjith Kumar. Protractor Testing Tutorial: Automation Tool Framework. *Guru99* [online]. 2021 [cit. 2021-02-25]. Dostupné z: https://www.guru99.com/protractor-testing.html
19. MATYKA, Kuba. How to start working with Protractor and run your first E2E test. *Neoteric* [online]. 2021 [cit. 2021-01-10]. Dostupné z: https://neoteric.eu/blog/start-to-start-working-with-protractor-and-run-your-first-e2e-test/
20. Protractor. Setting Up the Selenium Server. *Protractor* [online]. 2021 [cit. 2021-01-10]. Dostupné z: https://www.protractortest.org/#/server-setup
21. RAGONHA, Paulo. Jasmine JavaScript Testing. Packt Publishing Ltd, 2013. ISBN 978-1-78528-204-1.
22. Protractor. Using Page Objects to Organize Tests. *Protractor* [online]. 2021 [cit. 2021-01-10]. Dostupné z: https://www.protractortest.org/#/page-objects
23. BAJPAI, Raunak. How to run protractor tests parallely. *Medium* [online]. 2018 [cit. 2021-01-20]. Dostupné z: https://medium.com/@bajpairaunak123/how-to-run-protractor-tests-parallely-99241af5d2f8
24. CHEN, Steve Long. Configure Protractor to Run on Firefox with Multiple Capability. *A Steve Long Chen Blog* [online]. 2018 [cit. 2021-01-22]. Dostupné z:

http://blogs.stevelongchen.com/2018/09/23/configure-protractor-to-run-on-firefox-with-multiple-capability/

25. Tutorials Point. Protractor - Protractor And Selenium Server. *Tutorialspoint.com* [online]. 2021 [cit. 2021-01-22]. Dostupné z: https://www.tutorialspoint.com/protractor/protractor_and_selenium_server.htm
26. Protractor. Timeouts. *Protractor* [online]. 2021 [cit. 2021-01-10]. Dostupné z: https://www.protractortest.org/#/timeouts
27. Protractor. Choosing a Framework. *Protractor* [online]. 2021 [cit. 2021-01-10]. Dostupné z: https://www.protractortest.org/#/frameworks
28. Protractor. Using Locators. *Protractor* [online]. 2021 [cit. 2021-01-10]. Dostupné z: https://www.protractortest.org/#/locators
29. Protractor. ExpectedConditions. *Protractor* [online]. 2021 [cit. 2021-01-10]. Dostupné z: https://www.protractortest.org/#/api?view=ProtractorExpectedConditions
30. KHADEMI, Amir. Tired of Writing E2E Test? Try Recorders. *Medium* [online]. 2018 [cit. 2020-12-10]. Dostupné z: https://khademi.medium.com/tired-of-writing-e2e-test-try-recorders-b90ad940bb9c
31. CASTRO, Tulio. End to End (e2e) – Angular Testing – Protractor vs Cypress. *Tech Blog* [online]. 2018 [cit. 2020-12-15]. Dostupné z: https://techblog.fexcofts.com/2018/09/24/end-to-end-e2e-angular-testing-protractor-vs-cypress/
32. Software Testing Material. Generate Reports using Protractor Beautiful Reporter. *Software Testing Material* [online]. 2020 [cit. 2021-02-10]. Dostupné z: https://www.softwaretestingmaterial.com/protractor-beautiful-reporter/
33. Cypress. How it works. *Cypress* [online]. 2020 [cit. 2020-12-20]. Dostupné z: https://www.cypress.io/how-it-works
34. Cypress. Writing and Organizing Tests. *Cypress* [online]. 2021 [cit. 2021-01-17]. Dostupné z: https://docs.cypress.io/guides/core-concepts/writing-and-organizing-tests.html
35. HALDER, Krishnendu. Automated Testing with Cypress and Mocha. *Medium* [online]. 2020 [cit. 2021-01-15]. Dostupné z: https://medium.com/swlh/automated-testing-with-cypress-17bf74bfd97d

36. Cypress. Bundled Tools. *Cypress* [online]. 2021 [cit. 2021-01-21]. Dostupné z: https://docs.cypress.io/guides/references/bundled-tools.html
37. Tools QA. Cypress Tutorial. *Tools QA* [online]. 2021 [cit. 2021-01-17]. Dostupné z: https://www.toolsqa.com/cypress-tutorial/
38. Cypress. Configuration. *Cypress* [online]. 2021 [cit. 2021-01-20]. Dostupné z: https://docs.cypress.io/guides/references/configuration.html
39. Cypress. Launching Browsers. *Cypress* [online]. 2021 [cit. 2021-01-20]. Dostupné z: https://docs.cypress.io/guides/guides/launching-browsers.html
40. Cypress. Get. *Cypress* [online]. 2021 [cit. 2021-01-21]. Dostupné z: https://docs.cypress.io/api/commands/get.html
41. Cypress. Actions. *Cypress* [online]. 2021 [cit. 2021-01-21]. Dostupné z: https://example.cypress.io/commands/actions
42. Cypress. Wait. *Cypress* [online]. 2021 [cit. 2021-01-22]. Dostupné z: https://docs.cypress.io/api/commands/wait.html
43. Medium. Cypress Recorder — write Cypress tests much easier and faster. *Medium* [online]. 2019 [cit. 2020-12-10]. Dostupné z: https://medium.com/@gxcadr/cypress-recorder-write-cypress-tests-much-easier-and-faster-6ed70b712bff
44. Cypress. The Test Runner. *Cypress* [online]. 2021 [cit. 2021-02-12]. Dostupné z: https://docs.cypress.io/guides/core-concepts/test-runner
45. GRUBER, Adam. Mochawesome. *GitHub* [online]. 2021 [cit. 2021-02-25]. Dostupné z: https://github.com/adamgruber/mochawesome-report-generator
46. GOYAL, Sahil. Comparison between Selenium, Protractor, Cypress, and WebdriverIO. *Medium* [online]. 2020 [cit. 2021-03-05]. Dostupné z: https://medium.com/@sahil.goyal2/comparison-between-selenium-protractor-cypress-and-webdriverio-7786fc90ee09

Přílohy

Příloha č. 1: Identifikace objektů na úvodní stránce v Protractoru (vlastní zpracování)

```
import {by, element} from 'protractor';

export class MainPage {

  aPlans() {
    return element(by.cssContainingText('a', 'Plans'));
  }

  aCharts() {
    return element(by.cssContainingText('a', 'Charts'));
  }

  aLogin() {
    return element(by.cssContainingText('a', 'Login'));
  }

  checkboxMonitoring() {
    return element(by.css('yas-refresh p-checkbox'));
  }

  buttonCreateRobot() {
    return element(by.buttonText('Robot'));
  }
}
```

Příloha č. 2: Identifikace objektů na úvodní stránce v Cypressu (vlastní zpracování)

```
export class MainPage {

  aPlans() {
    return cy.contains('Plans');
  }

  aCharts() {
    return cy.contains('Charts');
  }

  aLogin() {
    return cy.contains('Login');
  }

  checkboxMonitoring() {
    return cy.get('yas-refresh p-checkbox');
  }

  buttonCreateRobot() {
    return cy.get('button').contains('Robot');
  }
}
```

Příloha č. 3: Identifikace objektů na přihlašovacím dialogu v Protractoru (vlastní zpracování)

```
import {by, element} from 'protractor';
import {MainPage} from './main.page';

export class LoginPage {

  inputUsername() {
    return element(by.id('username'));
  }

  inputPassword() {
    return element(by.id('password'));
  }

  buttonSignIn() {
    return element(by.buttonText('Sign in'));
  }

  prihlasUzivatele(jmeno: string, heslo: string) {
    const mainPage = new MainPage();

    mainPage.aLogin().click();
    this.inputUsername().sendKeys(jmeno);
    this.inputPassword().sendKeys(heslo);
    this.buttonSignIn().click();
  }
}
```

Příloha č. 4: Identifikace objektů na přihlašovacím dialogu v Cypressu (vlastní zpracování)

```
import {MainPage} from './main.page';

export class LoginPage {

  inputUsername() {
    return cy.get('#username');
  }

  inputPassword() {
    return cy.get('#password');
  }

  buttonSignIn() {
    return cy.contains('Sign in');
  }

  prihlasUzivatele(jmeno, heslo) {
    const mainPage = new MainPage();

    mainPage.aLogin().click();
    this.inputUsername().type(jmeno);
    this.inputPassword().type(heslo);
    this.buttonSignIn().click();
  }
}
```

Příloha č. 5: Identifikace objektů na dialogu nového robota v Protractoru (vlastní zpracování)

```
import {by, element} from 'protractor';

export class RobotDialogPage {

  inputName() {
    return element(by.name('name'));
  }

  dropdownModel() {
    return element(by.name('cRobotModel'));
  }

  dropdownTransfer() {
    return element(by.name('cTransferModel'));
  }

  dropdownLocation() {
    return element(by.name('robotLocationModel'));
  }

  inputIpAdress() {
    return element(by.name('ipAddress'));
  }

  inputSerialNumber() {
    return element(by.name('serial'));
  }

  inputQueryOncePerMs() {
    return element(by.name('nextQuestionTime'));
  }

  inputMaxConnection() {
    return element(by.name('maxConnection'));
  }

  buttonSave() {
    return element(by.buttonText('Save'));
  }
}
```

Příloha č. 6: Identifikace objektů na dialogu nového robota v Cypressu (vlastní zpracování)

```
export class RobotDialogPage {

  inputName() {
    return cy.get('input[name="name"]');
  }

  dropdownModel() {
    return cy.get('[name="cRobotModel"]');
  }

  dropdownTransfer() {
    return cy.get('[name="cTransferModel"]');
  }

  dropdownLocation() {
    return cy.get('[name="robotLocationModel"]');
  }

  inputIpAdress() {
    return cy.get('[name="ipAddress"]');
  }

  inputSerialNumber() {
    return cy.get('[name="serial"]');
  }

  inputQueryOncePerMs() {
    return cy.get('[name="nextQuestionTime"]');
  }

  inputMaxConnection() {
    return cy.get('[name="maxConnection"]');
  }

  buttonSave() {
    return cy.contains('Save');
  }
}
```

Příloha č. 7: Identifikace objektů na detailu robota v Protractoru (vlastní zpracování)

```
import {by, element} from 'protractor';

export class RobotPage {

  tableReadDataSignal() {
    return element(by.css('yas-robot-info-read-data-signal table'));
  }

  tableReadDataRegister() {
    return element(by.css('yas-robot-info-read-data-register table'));
  }

  tableReadDataVariables() {
    return element(by.css('yas-robot-info-read-data-variables table'));
  }

  aRobot(name: string) {
    return element(by.cssContainingText('a', name));
  }

  buttonByText(text: string) {
    return element(by.cssContainingText('button span', text));
  }

  rozklikniZalozku(zalozkaText: string) {
    element(by.cssContainingText('a', zalozkaText)).click();
  }
}
```

Příloha č. 8: Identifikace objektů na detailu robota v Cypressu (vlastní zpracování)

```
export class RobotPage {

  tableReadDataSignal() {
    return cy.get('yas-robot-info-read-data-signal table');
  }

  tableReadDataRegister() {
    return cy.get('yas-robot-info-read-data-register table');
  }

  tableReadDataVariables() {
    return cy.get('yas-robot-info-read-data-variables table');
  }

  aRobot(name) {
    return cy.contains(name);
  }

  buttonByText(text) {
    return cy.get('button').contains(text);
  }

  rozklikniZalozku(zalozkaText) {
    cy.contains(zalozkaText).click();
  }
}
```

Příloha č. 9: Identifikace objektů stránce s plány v Protractoru (vlastní zpracování)

```
import {by, element} from 'protractor';

export class PlansPage {

  buttonCreateBackupPlan() {
    return element(by.cssContainingText('button', 'Backup plan'));
  }

  buttonCreateMonitoringPlan() {
    return element(by.cssContainingText('button', 'Monitoring plan'));
  }
}
```

Příloha č. 10: Identifikace objektů stránce s plány v Cypressu (vlastní zpracování)

```
export class PlansPage {

  buttonCreateBackupPlan() {
    return cy.get('button').contains('Backup plan');
  }

  buttonCreateMonitoringPlan() {
    return cy.get('button').contains('Monitoring plan');
  }
}
```

Příloha č. 11: Identifikace objektů stránce zálohovacího plánu v Protractoru (vlastní zpracování)

```
import {by, element} from 'protractor';

export class PlanBackupPage {

  inputDescription() {
    return element(by.name('name'));
  }

  dropdownRobot() {
    return element(by.css('yas-robot-backup-group-seznam p-dropdown'));
  }

  radioButtonEveryDay() {
    return element(by.cssContainingText('p-radiobutton', 'Every day'));
  }

  inputTime() {
    return element(by.name('time'));
  }

  buttonAddRobot() {
    return element(by.partialButtonText('Robot'));
  }

  buttonFileType() {
    return element(by.partialButtonText('File type'));
  }

  multiselectFileType() {
    return element(by.css('p-multiselect'));
  }

  checkboxSelectAll() {
    return element.all(by.css('div[role=\'checkbox\']')).first();
   }

  buttonSave() {
    return element(by.buttonText('Save'));
  }
}
```

Příloha č. 12: Identifikace objektů stránce zálohovacího plánu v Cypressu (vlastní zpracování)

```
export class PlanBackupPage {

  inputDescription() {
    return cy.get('[name="name"]');
  }

  dropdownRobot() {
    return cy.get('yas-robot-backup-group-seznam p-dropdown');
  }

  radioButtonEveryDay() {
    return cy.get('p-radiobutton').contains('Every day');
  }

  inputTime() {
    return cy.get('[name="time"]');
  }

  buttonAddRobot() {
    return cy.get('button').contains('Robot');
  }

  buttonFileType() {
    return cy.get('button').contains('File type');
  }

  multiselectFileType() {
    return cy.get('p-multiselect');
  }

  checkboxSelectAll() {
    return cy.get('div[role="checkbox"]').first();
  }

  buttonSave() {
    return cy.contains('Save');
  }
}
```

Příloha č. 13: Identifikace objektů na stránce monitorovacího plánu v Protractoru (vlastní zpracování)

```
import {by, element} from 'protractor';

export class PlanMonitoringPage {

  inputDescription() {
    return element(by.css('input[type="text"]'));
  }

  dropdownRobot() {
    return element(by.css('yas-robot-signal-group-seznam p-dropdown'));
  }

  buttonAddRobot() {
    return element(by.partialButtonText('Robot'));
  }

  accordion() {
    return element(by.id('p-accordiontab-0'));
  }

  tabpanelVariables() {
    return element(by.id('p-tabpanel-1-label'));
  }

  inputMonitoringTime() {
    return element(by.name('askAgain'));
  }

  buttonAddVariable() {
    return element(by.partialButtonText('Variable'));
  }

  inputVariable(index: number) {
    return element.all(by.css('yas-variable-auto-complete p-autocomplete
span input')).get(index);
  }

  buttonSave() {
    return element(by.partialButtonText('Save'));
  }
}
```

Příloha č. 14: Identifikace objektů na stránce monitorovacího plánu v Cypressu (vlastní zpracování)

```
export class PlanMonitoringPage {

  inputDescription() {
    return cy.get('input[type="text"]');
  }

  dropdownRobot() {
    return cy.get('yas-robot-signal-group-seznam p-dropdown')
  }

  buttonAddRobot() {
    return cy.get('button').contains('Robot');
  }

  accordion() {
    return cy.get('#p-accordiontab-0');
  }

  tabpanelVariables() {
    return cy.get('#p-tabpanel-1-label');
  }

  inputMonitoringTime() {
    return cy.get('[name="askAgain"]');
  }

  buttonAddVariable() {
    return cy.get('button').contains('Variable');
  }

  inputVariable() {
    return cy.get('p-autocomplete input').last();
  }

  buttonSave() {
    return cy.contains('Save');
  }
}
```

Příloha č. 15: Identifikace objektů na stránce s evidencí grafů v Protractoru (vlastní zpracování)

```
import {by, element} from 'protractor';

export class ChartsPage {

  buttonCreateChart() {
    return element(by.cssContainingText('button', 'Chart'));
  }
}
```

Příloha č. 16: Identifikace objektů na stránce s evidencí grafů v Cypressu (vlastní zpracování)

```
export class ChartsPage {

  buttonCreateChart() {
    return cy.get('yas-charts button').contains('Chart');
  }
}
```

Příloha č. 17: Identifikace objektů na stránce pro vytvoření grafu v Protractoru (vlastní zpracování)

```
import {by, element} from 'protractor';

export class CreateChartPage {

  inputName() {
    return element(by.id('name'));
  }

  spanUtilization() {
    return element(by.cssContainingText('span', 'Utilization'));
  }

  dropdownPlan() {
    return element(by.css('yas-plan-selector p-dropdown'));
  }

  buttonPlan() {
    return element(by.css('p-button[label=\'Plan\']'));
  }

  spanAddVariable(text: string) {
    return element(by.cssContainingText('span', text));
  }

  dropdownRange() {
    return element(by.css('yas-chart-preview-settings p-dropdown'));
  }

  buttonSave() {
    return element(by.buttonText('Save'));
  }

  spanDoughnut() {
    return element(by.cssContainingText('span', 'Doughnut'));
  }

  selectButtonByText(text: string) {
    return element(by.cssContainingText('p-selectbutton span', text));
  }

  accordion() {
    return element(by.id('p-accordiontab-0'));
  }
}
```

Příloha č. 18: Identifikace objektů stránce pro vytvoření grafu v Cypressu (vlastní zpracování)

```
export class CreateChartPage {

  inputName() {
    return cy.get('#name');
  }

  spanUtilization() {
    return cy.get('span').contains('Utilization');
  }

  dropdownPlan() {
    return cy.get('yas-plan-selector p-dropdown');
  }

  buttonPlan() {
    return cy.get('p-button').contains('Plan');
  }

  spanAddVariable(text) {
    return element(by.cssContainingText('span', text));
  }

  dropdownRange() {
    return cy.get('yas-chart-preview-settings p-dropdown');
  }

  buttonSave() {
    return cy.contains('Save');
  }

  spanDoughnut() {
    return cy.get('span').contains('Doughnut');
  }

  selectButtonByText(text) {
    return cy.get('p-selectbutton span').contains(text);
  }

  accordion() {
    return cy.get('#p-accordiontab-0');
  }
}
```

Příloha č. 19: Obecné funkce v Protractoru (vlastní zpracování)

```
import {by, element, ElementFinder, protractor} from 'protractor';

/*
Výběr dat z dropdownu, který se aktivuje kliknutím na objekt label.
@param objekt - objekt labelu pro aktivaci dropdownu
@param nazev - název položky z dropdownu
*/
export function vyberZLabelDropdownuText(objekt: any, nazev: string) {
  objekt.click().then(() => { // aktivace dropdownu
    element(by.cssContainingText('li span', nazev)).click(); // výběr
položky z dropdownu
  });
}

/*
Výběr dat z dropdownu, který se aktivuje kliknutím na objekt label.
@param objekt - objekt labelu pro aktivaci dropdownu
@param text - text vkládaný do inputu
@param nazev - název položky z dropdownu
*/
export function vyberZInputDropdownuText(objekt: any, text: string,
nazev: string) {
  objekt.sendKeys(text).then(() => { // aktivace dropdownu
    element(by.cssContainingText('li span', nazev)).click(); // výběr
položky z dropdownu
  });
}

/*
Funkce vrací počet řádek z obdržené tabulky
@param ef - standardni popis objektu - trida.metoda(popis)
export function tabulkaPocetRadku(ef: ElementFinder) {
  return ef.all(by.css('tr')).count();
}
```

Příloha č. 20: Obecné funkce v Cypressu (vlastní zpracování)

```
/*
Výběr dat z dropdownu, který se aktivuje kliknutím na objekt label.
@param objekt - objekt labelu pro aktivaci dropdownu
@param nazev - název položky z dropdownu
*/
export function vyberZLabelDropdownuText(objekt, nazev) {
  objekt.click().then(() => { // aktivace dropdownu
    cy.get('li span').contains(nazev).click(); // výběr položky z
dropdownu
  });
}

/*
Výběr dat z dropdownu, který se aktivuje kliknutím na objekt label.
@param objekt - objekt labelu pro aktivaci dropdownu
@param text - text vkládaný do inputu
@param nazev - název položky z dropdownu
*/
export function vyberZInputDropdownuText(objekt, text, nazev) {
  objekt.type(text).then(() => { // aktivace dropdownu
    cy.get('li span').contains(nazev).click(); // výběr položky z
dropdownu
  });
}

/*
Funkce vrací počet řádek z obdržené tabulky
@param ef - standardni popis objektu - trida.metoda(popis)
@return vraci pocet radku v tabulce (number)
*/
export function tabulkaPocetRadku(ef) {
  return ef.find('tr').its('length');
}
```

Příloha č. 21: Test vytvoření robota v Protractoru (vlastní zpracování)

```
import {browser} from 'protractor';
import {MainPage} from '../pages/main.page';
import {PrihlaseniCase} from '../pages/commons/prihlaseni.case';
import {
  vyberZLabelDropdownuText
} from '../utils/util';
import {RobotDialogPage} from '../pages/robot.dialog.page';

const mainPage = new MainPage();
const prihlaseniCase = new PrihlaseniCase();
const robotDialogPage = new RobotDialogPage();

describe('Vytvoření robota', () => {
  const robotName = 'MOTOMAN - TEST';
  const robotModel = 'FS100';
  const robotTransfer = 'Ethernet/Http';
  const robotIp = '10.0.0.10';
  const robotSerialNumber = '123456789';
  const robotLocation = 'Default location';
  const robotQueryOncePerMs = 30;
  const robotMaxConnection = 10;

  it('Spuštění aplikace', () => {
    browser.get('');
  });

  it('Zrušení aktivní volby Monitoring (vypnutí přenačítání stránky)',
() => {
    mainPage.checkboxMonitoring().click();
  });

  it('Přihlášení do aplikace', () => {
    prihlaseniCase.prihlasUzivatele('test', '123');
  });

  it('Vytvoření robota', () => {
    mainPage.buttonCreateRobot().click();
  });

  describe('Vyplnění dialogu Create robot', () => {
    it('Vyplnění položky Name', () => {
      robotDialogPage.inputName().sendKeys(robotName);
    });

    it('Výběr položky Model', () => {
      vyberZLabelDropdownuText(robotDialogPage.dropdownModel(),
robotModel);
    });

    it('Výběr položky Transfer', () => {
      vyberZLabelDropdownuText(robotDialogPage.dropdownTransfer(),
robotTransfer);
    });
```

```
    it('Vyplnění položky IP adress', () => {
      robotDialogPage.inputIpAdress().sendKeys(robotIp);
    });

    it('Vyplnění položky Serial number', () => {
      robotDialogPage.inputSerialNumber().sendKeys(robotSerialNumber);
    });

    it('Výběr položky Location', () => {
      vyberZLabelDropdownuText(robotDialogPage.dropdownLocation(),
robotLocation);
    });

    it('Vyplnění položky Query once per ms', () => {
robotDialogPage.inputQueryOncePerMs().sendKeys(robotQueryOncePerMs);
    });

    it('Vyplnění položky Max connection', () => {
robotDialogPage.inputMaxConnection().sendKeys(robotMaxConnection);
    });

    it('Uložení robota', () => {
      robotDialogPage.buttonSave().click();
    });
  });
});
```

Příloha č. 22: Test vytvoření robota v Cypressu (vlastní zpracování)

```
import {MainPage} from './pages/main.page';
import {LoginPage} from './pages/login.page';
import {vyberZLabelDropdownuText} from './util';
import {RobotDialogPage} from './pages/robot.dialog.page';

const mainPage = new MainPage();
const loginPage = new LoginPage();
const robotDialogPage = new RobotDialogPage();

describe('Vytvoření robota', () => {
  const robotName = 'MOTOMAN - TEST';
  const robotModel = 'FS100';
  const robotTransfer = 'Ethernet/Http';
  const robotIp = '10.0.0.10';
  const robotSerialNumber = '123456789';
  const robotLocation = 'Default location';
  const robotQueryOncePerMs = 30;
  const robotMaxConnection = 10;

  it('Spuštění aplikace', () => {
    cy.visit('');
  });

  it('Zrušení aktivní volby Monitoring (vypnutí přenačítání stránky)',
() => {
    mainPage.checkboxMonitoring().click();
  });

  it('Přihlášení do aplikace', () => {
    loginPage.prihlasUzivatele('test', '123');
  });

  it('Vytvoření robota', () => {
    mainPage.buttonCreateRobot().click();
  });

  describe('Vyplnění dialogu Create robot', () => {
    it('Vyplnění položky Name', () => {
      robotDialogPage.inputName().type(robotName);
    });

    it('Výběr položky Model', () => {
      vyberZLabelDropdownuText(robotDialogPage.dropdownModel(),
robotModel);
    });

    it('Výběr položky Transfer', () => {
      vyberZLabelDropdownuText(robotDialogPage.dropdownTransfer(),
robotTransfer);
    });

    it('Vyplnění položky IP adress', () => {
      robotDialogPage.inputIpAdress().type(robotIp);
    });

    it('Vyplnění položky Serial number', () => {
      robotDialogPage.inputSerialNumber().type(robotSerialNumber);
    });
```

```
    it('Výběr položky Location', () => {
      vyberZLabelDropdownuText(robotDialogPage.dropdownLocation(),
robotLocation);
    });

    it('Vyplnění položky Query once per ms', () => {
      robotDialogPage.inputQueryOncePerMs().type(robotQueryOncePerMs);
    });

    it('Vyplnění položky Max connection', () => {
      robotDialogPage.inputMaxConnection().type(robotMaxConnection);
    });

    it('Uložení robota', () => {
      robotDialogPage.buttonSave().click();
    });
  });
});
```

Příloha č. 23: Test načtení dat robota v Protractoru (vlastní zpracování)

```
import {browser} from 'protractor';
import {PrihlaseniCase} from '../pages/commons/prihlaseni.case';
import {RobotPage} from '../pages/robot.page';
import {
  tabulkaPocetRadku,
} from '../utils/util';
import {MainPage} from '../pages/main.page';

const robotPage = new RobotPage();
const mainPage = new MainPage();
const prihlaseniCase = new PrihlaseniCase();

describe('Načtení dat robota', () => {
  const robotName = 'MOTOMAN - TEST';

  it('Spuštění aplikace', () => {
    browser.get('');
  });

  it('Zrušení aktivní volby Monitoring (vypnutí přenačítání stránky)',
() => {
    mainPage.checkboxMonitoring().click();
  });

  it('Přihlášení do aplikace', () => {
    prihlaseniCase.prihlasUzivatele('test', '123');
  });

  it('Výběr robota', () => {
    robotPage.aRobot(robotName).click();
  });

  it('Záložka Read data from robot', () => {
    robotPage.rozklikniZalozku('Read data from robot');
  });

  describe('Named signals', () => {
    it('Přepnutí na podzáložku Named signals', () => {
      robotPage.rozklikniZalozku('Named signals');
    });

    it('Tlačítko Update signals from file', () => {
      robotPage.buttonByText('Update signals from file').click();
    });

    it('Kontrola záznamů v tabulce', () => {
      tabulkaPocetRadku(robotPage.tableReadDataSignal()).then((pocet:
any) => {
        expect(pocet).toBeGreaterThan(0);
      });
    });
  });

  describe('Named Registers', () => {
    it('Přepnutí na podzáložku Named Registers', () => {
      robotPage.rozklikniZalozku('Named registers');
    });
```

```
    it('Tlačítko Update registers from file', () => {
      robotPage.buttonByText('Update registers from file').click();
    });

    it('Kontrola záznamů v tabulce', () => {
     tabulkaPocetRadku(robotPage.tableReadDataRegister()).then((pocet:
any) => {
        expect(pocet).toBeGreaterThan(0);
      });
    });
  });

  describe('Named variables', () => {
    it('Přepnutí na podzáložku Named variables', () => {
      robotPage.rozklikniZalozku('Named variables');
    });

    it('Tlačítko Update variable names from file', () => {
      robotPage.buttonByText('Update variable names from
file').click();
    });

    it('Kontrola záznamů v tabulce', () => {
    tabulkaPocetRadku(robotPage.tableReadDataVariables()).then((pocet:
any) => {
         expect(pocet).toBeGreaterThan(0);
      });
    });
  });
});
```

Příloha č. 24: Test načtení dat robota v Cypressu (vlastní zpracování)

```
import {LoginPage} from './pages/login.page';
import {RobotPage} from './pages/robot.page';
import {MainPage} from './pages/main.page';
import { tabulkaPocetRadku } from './util';

const robotPage = new RobotPage();
const mainPage = new MainPage();
const loginPage = new LoginPage();

describe('Načtení dat robota', () => {
  const robotName = 'MOTOMAN - TEST';

  it('Spuštění aplikace', () => {
    cy.visit('');
  });

  it('Zrušení aktivní volby Monitoring (vypnutí přenačítání stránky)',
() => {
    mainPage.checkboxMonitoring().click();
  });

  it('Přihlášení do aplikace', () => {
    loginPage.prihlasUzivatele('test', '123');
  });

  it('Výběr robota', () => {
    robotPage.aRobot(robotName).click();
  });

  it('Záložka Read data from robot', () => {
    robotPage.rozklikniZalozku('Read data');
  });

  describe('Named signals', () => {
    it('Přepnutí na podzáložku Named signals', () => {
      robotPage.rozklikniZalozku('Named signals');
    });

    it('Tlačítko Update signals from file', () => {
      robotPage.buttonByText('Update signals from file').click();
    });

    it('Kontrola záznamů v tabulce', () => {
    tabulkaPocetRadku(robotPage.tableReadDataSignal()).should('be.gt',
    0);
    });
  });

  describe('Named Registers', () => {
    it('Přepnutí na podzáložku Named Registers', () => {
      robotPage.rozklikniZalozku('Named registers');
    });

    it('Tlačítko Update registers from file', () => {
      robotPage.buttonByText('Update registers from file').click();
    });
```

```
    it('Kontrola záznamů v tabulce', () => {
    tabulkaPocetRadku(robotPage.tableReadDataRegister()).should('be.gt
    ', 0);
    });
  });

  describe('Named variables', () => {
    it('Přepnutí na podzáložku Named variables', () => {
      robotPage.rozklikniZalozku('Named variables');
    });

    it('Tlačítko Update variable names from file', () => {
      robotPage.buttonByText('Update variable names from
file').click();
    });

    it('Kontrola záznamů v tabulce', () => {
    tabulkaPocetRadku(robotPage.tableReadDataVariables()).should('be.g
    t', 0);
    });
  });
});
```

Příloha č. 25: Test vytvoření plánu monitoringu v Protractoru (vlastní zpracování)

```
import {browser, ExpectedConditions} from 'protractor';
import {MainPage} from '../pages/main.page';
import {PlanMonitoringPage} from '../pages/plan.monitoring.page';
import {PrihlaseniCase} from '../pages/commons/prihlaseni.case';
import {PlansPage} from '../pages/Plans.page';
import {
  vyberZInputDropdownuText,
  vyberZLabelDropdownuText
} from '../utils/util';

const monitoringPage = new PlanMonitoringPage();
const mainPage = new MainPage();
const prihlaseniCase = new PrihlaseniCase();
const plansPage = new PlansPage();

describe('Plán pro monitoring', () => {
  const robotName = 'MOTOMAN - TEST';
  const description = 'Plán - ' + robotName;
  const monitoringTime = '20';
  const variable1 = '0 - PocetKusu - DOUBLE';
  const variable2 = '1 - PocetUkonu - DOUBLE';

  it('Spuštění aplikace', () => {
    browser.get('');
  });

  it('Zrušení aktivní volby Monitoring (vypnutí přenačítání stránky)',
() => {
    mainPage.checkboxMonitoring().click();
  });

  it('Přihlášení do aplikace', () => {
    prihlaseniCase.prihlasUzivatele('test', '123');
  });

  it('Přechod na formulář Plans', () => {
    mainPage.aPlans().click();
  });

  it('Tlačítko Monitoring plan', () => {
    plansPage.buttonCreateMonitoringPlan().click();
  });

  describe('Založení monitoringu', () => {
    it('Vyplnění položky Description', () => {
      monitoringPage.inputDescription().sendKeys(description);
    });

    it('Výběr položky z dropdownu Add Robot', () => {
      vyberZLabelDropdownuText(monitoringPage.dropdownRobot(),
robotName);
    });

   it('Potvrzení výběru položky tlačítkem "+ Robot"', () => {
      monitoringPage.buttonAddRobot().click();
    });
```

```
    it('Rozbalení accordionu', () => {
      monitoringPage.accordion().click();
    });

    it('Vyplnění položky Monitoring time', () => {
    browser.wait(ExpectedConditions.elementToBeClickable(monitoringPag
    e.inputMonitoringTime()), 5000).then(() => {
        monitoringPage.inputMonitoringTime().sendKeys(monitoringTime);
      });
    });

    it('Otevření záložky "Variables"', () => {
      monitoringPage.tabpanelVariables().click();
    });

    it('Tlačítko Add variable', () => {
      monitoringPage.buttonAddVariable().click();
    });

    it('Přidání proměnné PocetKusu', () => {
      vyberZInputDropdownuText(monitoringPage.inputVariable(0),
'pocet', variable1);
    });

    it('Tlačítko Add variable', () => {
      monitoringPage.buttonAddVariable().click();
    });

    it('Přidání proměnné PocetUkonu', () => {
      vyberZInputDropdownuText(monitoringPage.inputVariable(1),
'pocet', variable2);
    });

    it('Uložení monitoringu', () => {
      monitoringPage.buttonSave().click();
    });
  });
});
```

Příloha č. 26: Test vytvoření plánu monitoringu v Cypressu (vlastní zpracování)

```
import {MainPage} from './pages/main.page';
import {PlanMonitoringPage} from './pages/plan.monitoring.page';
import {LoginPage} from './pages/login.page';
import {PlansPage} from './pages/plans.page';
import {vyberZInputDropdownuText, vyberZLabelDropdownuText} from
'./util';

const monitoringPage = new PlanMonitoringPage();
const mainPage = new MainPage();
const loginPage = new LoginPage();
const plansPage = new PlansPage();

describe('Plán pro monitoring', () => {
  const robotName = 'MOTOMAN - TEST';
  const description = 'Plán - ' + robotName;
  const monitoringTime = '20';
  const variable1 = '0 - PocetKusu - DOUBLE';
  const variable2 = '1 - PocetUkonu - DOUBLE';

  it('Spuštění aplikace', () => {
    cy.visit('');
  });

  it('Zrušení aktivní volby Monitoring (vypnutí přenačítání stránky)',
() => {
    mainPage.checkboxMonitoring().click();
  });

  it('Přihlášení do aplikace', () => {
    loginPage.prihlasUzivatele('test', '123');
  });

  it('Přechod na formulář Plans', () => {
    mainPage.aPlans().click();
  });

  it('Tlačítko Monitoring plan', () => {
    plansPage.buttonCreateMonitoringPlan().click();
  });

  describe('Založení monitoringu', () => {
    it('Vyplnění položky Description', () => {
      monitoringPage.inputDescription().type(description);
    });

    it('Výběr položky z dropdownu Add Robot', () => {
      vyberZLabelDropdownuText(monitoringPage.dropdownRobot(),
robotName);
    });

    it('Potvrzení výběru položky tlačítkem "+ Robot"', () => {
      monitoringPage.buttonAddRobot().click();
    });

    it('Rozbalení accordionu', () => {
      monitoringPage.accordion().click();
    });
```

```
    it('Vyplnění položky Monitoring time', () => {
      monitoringPage.inputMonitoringTime().type(monitoringTime);
    });

    it('Otevření záložky "Variables"', () => {
      monitoringPage.tabpanelVariables().click();
    });

    it('Tlačítko Add variable', () => {
      monitoringPage.buttonAddVariable().click();
    });

    it('Přidání proměnné PocetKusu', () => {
      vyberZInputDropdownuText(monitoringPage.inputVariable(),
'pocet', variable1);
    });

    it('Tlačítko Add variable', () => {
      monitoringPage.buttonAddVariable().click().then(() => {
        cy.wait(150);
      })
    });

    it('Přidání proměnné PocetUkonu', () => {
      vyberZInputDropdownuText(monitoringPage.inputVariable(),
'pocet', variable2);
    });

    it('Uložení monitoringu', () => {
      monitoringPage.buttonSave().click();
    });
  });
});
```

Příloha č. 27: Test vytvoření backup plánu v Protractoru (vlastní zpracování)

```
import {browser} from 'protractor';
import {MainPage} from '../pages/main.page';
import {PrihlaseniCase} from '../pages/commons/prihlaseni.case';
import {PlansPage} from '../pages/Plans.page';
import {vyberZLabelDropdownuText} from '../utils/util';
import {PlanBackupPage} from '../pages/plan.backup.page';

const mainPage = new MainPage();
const prihlaseniCase = new PrihlaseniCase();
const plansPage = new PlansPage();
const backupPage = new PlanBackupPage();

describe('Plán pro backup', () => {
  const robotName = 'MOTOMAN - TEST';
  const description = 'Backup - ' + robotName;
  const time = '1000AM';

  it('Spuštění aplikace', () => {
    browser.get('');
  });

  it('Zrušení aktivní volby Monitoring (vypnutí přenačítání stránky)',
() => {
    mainPage.checkboxMonitoring().click();
  });

  it('Přihlášení do aplikace', () => {
    prihlaseniCase.prihlasUzivatele('test', '123');
  });

  it('Přechod na stránku Plans', () => {
    mainPage.aPlans().click();
  });

  it('Tlačítko Backup plan', () => {
    plansPage.buttonCreateBackupPlan().click();
  });

  describe('Založení Backup plan', () => {
    it('Vyplnění položky Description', () => {
      backupPage.inputDescription().sendKeys(description);
    });

    it('Výběr položky z dropdownu Add Robot', () => {
      vyberZLabelDropdownuText(backupPage.dropdownRobot(), robotName);
    });

    it('Potvrzení výběru položky tlačítkem "+ Robot"', () => {
      backupPage.buttonAddRobot().click();
    });

    it('Výběr Every day', () => {
      backupPage.radioButtonEveryDay().click();
    });
```

```
    it('Vyplnění času', () => {
      backupPage.inputTime().sendKeys(time);
    });

    it('Tlačítko Add folder type', () => {
      backupPage.buttonFileType().click();
    });

    it('Výběr všech dostupných typů', () => {
      backupPage.multiselectFileType().click().then(() => {
        backupPage.checkboxSelectAll().click().then(() => {
        });
      });
    });

    it('Uložení backup tlačítkem Save', () => {
      backupPage.buttonSave().click();
    });
  });
});
```

Příloha č. 28: Test vytvoření backup plánu v Cypressu (vlastní zpracování)

```
import {MainPage} from './pages/main.page';
import {LoginPage} from './pages/login.page';
import {PlansPage} from './pages/Plans.page';
import {vyberZLabelDropdownuText} from './util';
import {PlanBackupPage} from './pages/plan.backup.page';

const mainPage = new MainPage();
const loginPage = new LoginPage();
const plansPage = new PlansPage();
const backupPage = new PlanBackupPage();

describe('Plán pro backup', () => {
  const robotName = 'MOTOMAN - TEST';
  const description = 'Backup - ' + robotName;
  const time = '10:00';

  it('Spuštění aplikace', () => {
    cy.visit('');
  });

  it('Zrušení aktivní volby Monitoring (vypnutí přenačítání stránky)',
() => {
    mainPage.checkboxMonitoring().click();
  });

  it('Přihlášení do aplikace', () => {
    loginPage.prihlasUzivatele('test', '123');
  });

  it('Přechod na stránku Plans', () => {
    mainPage.aPlans().click();
  });

  it('Tlačítko Backup plan', () => {
    plansPage.buttonCreateBackupPlan().click();
  });

  describe('Založení Backup plan', () => {
    it('Vyplnění položky Description', () => {
      backupPage.inputDescription().type(description);
    });

    it('Výběr položky z dropdownu Add Robot', () => {
      vyberZLabelDropdownuText(backupPage.dropdownRobot(), robotName);
    });

    it('Potvrzení výběru položky tlačítkem "+ Robot"', () => {
      backupPage.buttonAddRobot().click();
    });

    it('Výběr Every day', () => {
      backupPage.radioButtonEveryDay().click();
    });

    it('Vyplnění času', () => {
      backupPage.inputTime().type(time);
    });
```

```
    it('Tlačítko Add folder type', () => {
      backupPage.buttonFileType().click();
    });

    it('Výběr všech dostupných typů', () => {
      backupPage.multiselectFileType().click().then(() => {
        backupPage.checkboxSelectAll().click().then(() => {
        });
      });
    });

    it('Uložení backup tlačítkem Save', () => {
      backupPage.buttonSave().click();
    });
  });
});
```

Příloha č. 29: Test vytvoření grafu Utilizace v Protractoru (vlastní zpracování)

```
import {browser} from 'protractor';
import {MainPage} from '../pages/main.page';
import {PrihlaseniCase} from '../pages/commons/prihlaseni.case';
import {ChartsPage} from '../pages/charts.page';
import {CreateChartPage} from '../pages/create.chart.page';
import {vyberZLabelDropdownuText} from '../utils/util';

const mainPage = new MainPage();
const prihlaseniCase = new PrihlaseniCase();
const chartsPage = new ChartsPage();
const createChartsPage = new CreateChartPage();

describe('Graf - Utilizace', () => {
  const chartName = 'Graf Util Test';
  const plan = 'Engine hours plan';
  const range = 'This week';

  it('Spuštění aplikace', () => {
    browser.get('');
  });

  it('Zrušení aktivní volby Monitoring (vypnutí přenačítání stránky)',
() => {
    mainPage.checkboxMonitoring().click();
  });

  it('Přihlášení do aplikace', () => {
    prihlaseniCase.prihlasUzivatele('test', '123');
  });

  it('Přechod na stránku Charts', () => {
    mainPage.aCharts().click();
  });

  describe('Vytvoření nového grafu', () => {
    it('Tlačítko Chart', () => {
      chartsPage.buttonCreateChart().click();
    });

    it('Zadání name', () => {
      createChartsPage.inputName().sendKeys(chartName);
    });

    it('Výběr tlačítka Utilization', () => {
      createChartsPage.spanUtilization().click();
    });

    it('Výběr typu plánu', () => {
      vyberZLabelDropdownuText(createChartsPage.dropdownPlan(), plan);
    });

    it('Vytvoření plánu', () => {
      createChartsPage.buttonPlan().click();
    });
```

```
    it('Výběr rozsahu', () => {
    vyberZLabelDropdownuText(createChartsPage.dropdownRange(), range);
    });

    it('Výběr tlačítka Hour', () => {
      createChartsPage.selectButtonByText('Hour').click();
    });

    it('Uložení tlačítkem Save', () => {
      createChartsPage.buttonSave().click();
    });
  });
});
```

Příloha č. 30: Test vytvoření grafu Utilizace v Cypressu (vlastní zpracování)

```
import {MainPage} from './pages/main.page';
import {LoginPage} from './pages/login.page';
import {ChartsPage} from './pages/charts.page';
import {CreateChartPage} from './pages/create.chart.page';
import {vyberZLabelDropdownuText} from './util';

const mainPage = new MainPage();
const loginPage = new LoginPage();
const chartsPage = new ChartsPage();
const createChartsPage = new CreateChartPage();

describe('Graf - Utilizace', () => {
  const chartName = 'Graf Util Test';
  const plan = 'Engine hours plan';
  const range = 'This week';

  it('Spuštění aplikace', () => {
    cy.visit('');
  });

  it('Zrušení aktivní volby Monitoring (vypnutí přenačítání stránky)',
() => {
    mainPage.checkboxMonitoring().click();
  });

  it('Přihlášení do aplikace', () => {
    loginPage.prihlasUzivatele('test', '123');
  });

  it('Přechod na stránku Charts', () => {
    mainPage.aCharts().click();
  });

  describe('Vytvoření nového grafu', () => {
    it('Tlačítko Chart', () => {
      chartsPage.buttonCreateChart().click();
    });

    it('Zadání name', () => {
      createChartsPage.inputName().type(chartName);
    });

    it('Výběr tlačítka Utilization', () => {
      createChartsPage.spanUtilization().click();
    });

    it('Výběr typu plánu', () => {
      vyberZLabelDropdownuText(createChartsPage.dropdownPlan(), plan);
    });

    it('Vytvoření plánu', () => {
      createChartsPage.buttonPlan().click();
    });
```

```
    it('Výběr rozsahu', () => {
      vyberZLabelDropdownuText(createChartsPage.dropdownRange(),
range);
    });

    it('Výběr tlačítka Hour', () => {
      createChartsPage.selectButtonByText('Hour').click();
    });

    it('Uložení tlačítkem Save', () => {
      createChartsPage.buttonSave().click();
    });
  });
});
```

Příloha č. 31: Test vytvoření grafu Monitoringu v Protractoru (vlastní zpracování)

```
import {browser} from 'protractor';
import {MainPage} from '../pages/main.page';
import {PrihlaseniCase} from '../pages/commons/prihlaseni.case';
import {ChartsPage} from '../pages/charts.page';
import {CreateChartPage} from '../pages/create.chart.page';
import {vyberZLabelDropdownuText} from '../utils/util';

const mainPage = new MainPage();
const prihlaseniCase = new PrihlaseniCase();
const chartsPage = new ChartsPage();
const createChartsPage = new CreateChartPage();

describe('Graf - Monitoring', () => {
  const robotName = 'MOTOMAN';
  const chartName = 'Graf Monit';
  const plan = 'Plán - ' + robotName;
  const range = 'This week';
  const variable1 = 'PocetKusu';
  const variable2 = 'PocetUkonu';

  it('Spuštění aplikace', () => {
    browser.get('');
  });

  it('Zrušení aktivní volby Monitoring (vypnutí přenačítání stránky)',
() => {
    mainPage.checkboxMonitoring().click();
  });

  it('Přihlášení do aplikace', () => {
    prihlaseniCase.prihlasUzivatele('test', '123');
  });

  it('Přechod na stránku Charts', () => {
    mainPage.aCharts().click();
  });

  describe('Vytvoření nového grafu', () => {
    it('Tlačítko Create chart', () => {
      chartsPage.buttonCreateChart().click();
    });

    it('Zadání name', () => {
      createChartsPage.inputName().sendKeys(chartName);
    });

    it('Výběr tlačítka Plan', () => {
      createChartsPage.selectButtonByText('Plan').click();
    });

    it('Výběr plánu', () => {
      vyberZLabelDropdownuText(createChartsPage.dropdownPlan(), plan);
    });
```

```
    it('Přidání plánu', () => {
      createChartsPage.buttonPlan().click();
    });

    it('Rozbalení accordionu', () => {
      createChartsPage.accordion().click();
    });

    it('Přidání proměnné PocetKusu', () => {
      createChartsPage.spanAddVariable(variable1).click();
    });

    it('Přidání proměnné PocetUkonu', () => {
      createChartsPage.spanAddVariable(variable2).click();
    });

    it('Výběr rozsahu', () => {
      vyberZLabelDropdownuText(createChartsPage.dropdownRange(),
range);
    });

    it('Výběr tlačítka Hour', () => {
      createChartsPage.selectButtonByText('Hour').click();
    });

    it('Uložení tlačítkem Save', () => {
      createChartsPage.buttonSave().click();
    });
  });
});
```

Příloha č. 32: Test vytvoření grafu Monitoringu v Cypressu (vlastní zpracování)

```
import {MainPage} from './pages/main.page';
import {LoginPage} from './pages/login.page';
import {ChartsPage} from './pages/charts.page';
import {CreateChartPage} from './pages/create.chart.page';
import {vyberZLabelDropdownuText} from './util';

const mainPage = new MainPage();
const loginPage = new LoginPage();
const chartsPage = new ChartsPage();
const createChartsPage = new CreateChartPage();

describe('Graf - Monitoring', () => {
  const robotName = 'MOTOMAN';
  const chartName = 'Graf Monit';
  const plan = 'Plán - ' + robotName;
  const range = 'This week';
  const variable1 = 'PocetKusu';
  const variable2 = 'PocetUkonu';

  it('Spuštění aplikace', () => {
    cy.visit('');
  });
```

```
  it('Zrušení aktivní volby Monitoring (vypnutí přenačítání stránky)',
() => {
    mainPage.checkboxMonitoring().click();
  });

  it('Přihlášení do aplikace', () => {
    loginPage.prihlasUzivatele('test', '123');
  });

  it('Přechod na stránku Charts', () => {
    mainPage.aCharts().click();
  });

  describe('Vytvoření nového grafu', () => {
    it('Tlačítko Create chart', () => {
      chartsPage.buttonCreateChart().click();
    });

    it('Zadání name', () => {
      createChartsPage.inputName().type(chartName);
    });

    it('Výběr tlačítka Plan', () => {
      createChartsPage.selectButtonByText('Plan').click();
    });

    it('Výběr plánu', () => {
      vyberZLabelDropdownuText(createChartsPage.dropdownPlan(), plan);
    });

    it('Přidání plánu', () => {
      createChartsPage.buttonPlan().click();
    });

    it('Rozbalení accordionu', () => {
      createChartsPage.accordion().click();
    });

    it('Přidání proměnné PocetKusu', () => {
      createChartsPage.spanAddVariable(variable1).click();
    });

    it('Přidání proměnné PocetUkonu', () => {
      createChartsPage.spanAddVariable(variable2).click();
    });

    it('Výběr rozsahu', () => {
      vyberZLabelDropdownuText(createChartsPage.dropdownRange(),
range);
    });

    it('Výběr tlačítka Hour', () => {
      createChartsPage.selectButtonByText('Hour').click();
    });

    it('Uložení tlačítkem Save', () => {
      createChartsPage.buttonSave().click();
    });
  });
});
```

Příloha č. 33: Test vytvoření grafu Maintenance v Protractoru (vlastní zpracování)

```
import {browser} from 'protractor';
import {MainPage} from '../pages/main.page';
import {PrihlaseniCase} from '../pages/commons/prihlaseni.case';
import {ChartsPage} from '../pages/charts.page';
import {CreateChartPage} from '../pages/create.chart.page';

const mainPage = new MainPage();
const prihlaseniCase = new PrihlaseniCase();
const chartsPage = new ChartsPage();
const createChartsPage = new CreateChartPage();

describe('Graf - Maintenance', () => {
  const chartName = 'Graf Maint';

  it('Spuštění aplikace', () => {
    browser.get('');
  });

  it('Zrušení aktivní volby Monitoring (vypnutí přenačítání stránky)',
() => {
    mainPage.checkboxMonitoring().click();
  });

  it('Přihlášení do aplikace', () => {
    prihlaseniCase.prihlasUzivatele('test', '123');
  });

  it('Přechod na stránku Charts', () => {
    mainPage.aCharts().click();
  });

  describe('Vytvoření nového grafu', () => {
    it('Tlačítko Chart', () => {
      chartsPage.buttonCreateChart().click();
    });

    it('Zadání name', () => {
      createChartsPage.inputName().sendKeys(chartName);
    });

    it('Výběr tlačítka Maintenance', () => {
      createChartsPage.selectButtonByText('Maintenance').click();
    });

    it('Výběr typu grafu Doughnut', () => {
      createChartsPage.spanDoughnut().click();
    });

    it('Uložení tlačítkem Save', () => {
      createChartsPage.buttonSave().click();
    });
  });
});
```

Příloha č. 34: Test vytvoření grafu Maintenance v Cypressu (vlastní zpracování)

```
import {MainPage} from './pages/main.page';
import {LoginPage} from './pages/login.page';
import {ChartsPage} from './pages/charts.page';
import {CreateChartPage} from './pages/create.chart.page';

const mainPage = new MainPage();
const loginPage = new LoginPage();
const chartsPage = new ChartsPage();
const createChartsPage = new CreateChartPage();

describe('Graf - Maintenance', () => {
  const chartName = 'Graf Maint';

  it('Spuštění aplikace', () => {
    cy.visit('');
  });

  it('Zrušení aktivní volby Monitoring (vypnutí přenačítání stránky)',
() => {
    mainPage.checkboxMonitoring().click();
  });

  it('Přihlášení do aplikace', () => {
    loginPage.prihlasUzivatele('test', '123');
  });

  it('Přechod na stránku Charts', () => {
    mainPage.aCharts().click();
  });

  describe('Vytvoření nového grafu', () => {
    it('Tlačítko Chart', () => {
      chartsPage.buttonCreateChart().click();
    });

    it('Zadání name', () => {
      createChartsPage.inputName().type(chartName);
    });

    it('Výběr tlačítka Maintenance', () => {
      createChartsPage.selectButtonByText('Maintenance').click();
    });

    it('Výběr typu grafu Doughnut', () => {
      createChartsPage.spanDoughnut().click();
    });

    it('Uložení tlačítkem Save', () => {
      createChartsPage.buttonSave().click();
    });
  });
});
```