Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Principy testování webových aplikací

Bakalářská práce

Autor: Jan Theodor
Studijní program: Aplikovaná informatika

Vedoucí práce: doc. Mgr. Tomáš Kozel, Ph.D.

Hradec Králové Duben 2024

Prohlášení:

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

V Hradci Králové dne 23.4.2024 Jan Theodor

Poděkování:

Děkuji vedoucímu bakalářské práce doc. Mgr. Tomášovi Kozlovi, Ph.D. za metodické vedení práce a konzultace v rámci vedení bakalářské práce. Děkuji za veškeré poznatky, rady a zkušenosti které mi byly předány a pomohly k sepsání této práce.

Abstrakt

Tato bakalářská práce se zaměřuje na úvod do vývoje webových aplikací a problematiku jejich testování. V úvodu jsou představeny základní nativní technologie, ve kterých mohou být webové aplikace vyvíjeny. Dále se práce zabývá testováním webových aplikací, kde jsou popsány různé druhy testů a postupy testování během vývoje. Cílem práce je porovnání dvou vybraných frameworků pro automatické testování webových aplikací: Selenium a Cypress. Pro každý z těchto nástrojů jsou uvedeny jejich výhody, nevýhody a použití. Práce také obsahuje ukázky vzorových testů, které demonstrují jejich použití.

Klíčová slova: Webová aplikace, Testování software, Selenium, Cypress, Automatické testování

Abstract

Title: Principles of web application testing

This bachelor thesis focuses on an introduction to web application development and testing. In the introduction, the basic native technologies in which web applications can be developed are introduced. Next, the thesis deals with web application testing, where different types of tests and testing procedures during development are described. The topic of the thesis is a comparison of two selected frameworks for automated testing of web applications: Selenium and Cypress. For each of these tools, their advantages, disadvantages and use are presented. The thesis also includes sample tests that demonstrate their use.

Keywords: Web application, Software testing, Selenium, Cypress, Automatic testing

Obsah

Seznam obrázků

Seznam tabulek

Seznam zkratek

API	Application Programming Interface
CSS	Cascading Style Sheets
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IDE	Integrated Development Environment
JS	JavaScript
PHP	Hypertext Preprocessor
RC	Remote Control
SEO	Search Engine Optimization
SSL	Secure Sockets Layer
TLS	Transport Layer Security
URL	Uniform Resource Locator
XML	Extensible Markup Language

1 Úvod

Testování webových aplikací se stalo nedílnou součástí vývoje moderní webové aplikace. Dnes, kdy je možné spouštět webový prohlížeč na široké škále zařízení, je nutné, aby se aplikace chovala stejně na všech zařízeních. Pokud na webu dojde k nějaké chybě, ať už je povahy funkční, nebo pouze designové, může návštěvníka odradit od dalšího používání aplikace a v mnohých případech i od potenciálního nákupu. Z těchto důvodu je nutné webové aplikace již během jejich vývoje testovat. K testování aplikací můžeme přistupovat několika způsoby, které jsou v práci vysvětleny. Existuje mnoho frameworků, které mohou práci ulehčit. Bude představeno pár existujících řešení a zvoleno to, které se bude nejvíce hodit pro danou aplikaci. Při zkoumání a porovnávání frameworků bude využita umělá inteligence pro jednodušší a přesnější převod testů mezi použitými nástroji. Závěrečná část práce bude věnována prezentaci ukázky testování aplikace pomocí vybraných testovacích frameworků. Pro tuto ukázku byly zvoleny dva testovací frameworky Cypress a Selenium, které jsou primárně určeny k automatickému testování webových aplikací.

2 Cíl práce

Cílem práce je předat čtenáři komplexní přehled o různých metodách a nástrojích testování, vysvětlit proces vývoje webové aplikace a principy testování, aby byl schopen efektivněji plánovat, provádět a vyhodnocovat testovací procesy ve svých projektech. V práci budou popsány základní technologie nezbytné pro vývoj a testování těchto aplikací, tak i porovnání různých technologií použitelných pro testování, přičemž čtenář se seznámí s jejich výhodami, nevýhodami a možnostmi využití v praxi.

3 Webová aplikace

Tato kapitola je zaměřena na vymezení základních pojmů v oblasti webového vývoje a uvedení čtenáře do obecného povědomí o základních a nezbytných technologiích

3.1 Definice a použití

Dle (1) je webová aplikace taková aplikace, která je uživateli zprostředkována webovým serverem, ke kterému se zpravidla přistupuje přes internetovou síť, případně přes vnitropodnikovou síť intranet.

Podle (2) se webová aplikace označuje jako webové místo, které obsahuje úplně nebo alespoň částečně neurčený obsah. Konečný obsah je zprostředkován až na základě požadavku návštěvníka na server. Jelikož se tyto požadavky mohou měnit jedná se již o dynamickou stránku. (3)

Webové aplikace mají velkou výhodu oproti klasickým aplikacím, kterou je možnost používání aplikace přes webový prohlížeč bez nutnosti instalace dodatečného softwaru. V mnohých případech mohou být i levnější co se týče ceny vývoje. Tyto aplikace se v dnešní době rozšířily prakticky do všech odvětví.

Příklady využití webových aplikací

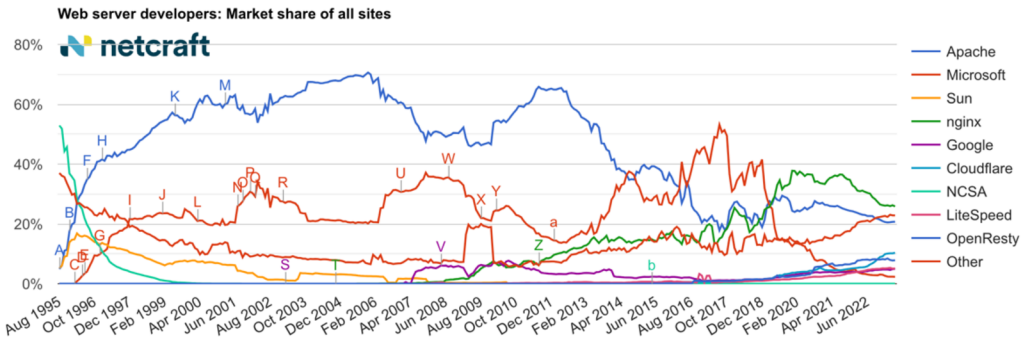
- Prezentační web
- Internetová databáze
    - Diskusní fórum
    - Filmová databáze
- Internetový obchod
- Podnikový intranet

3.2 Server

Server je hardwarový prostředek, který poskytuje služby, data nebo zdroje pro jiné uživatele připojené k celosvětové síti. Jeho hlavním účelem je zprostředkovávat požadavky od zúčastněných, resp. připojených zařízení a následně vyhodnocovat odpovídající služby nebo data. (3) V dnešní době existuje velká řada tzv. serverových řešení. Některé slouží jako datová úložiště, další mohou sloužit jako tzv. game server, který umožňuje propojení uživatelů ve hře a jejich vzájemnou interakci nebo web

server. (3) V našem případě pro nás bude nejdůležitější web server. Jedná se o server, který pomocí HTTP protokolu zprostředkovává uživateli webové stránky.

Aby bylo možné k webový aplikacím přistupovat je třeba mít na serveru nainstalovaný příslušný software. Mezi nejrozšířenější v dnešní době patří Apache HTTP Server nebo Nginx. Obě technologie poskytují stejnou službu, avšak jejich řešení v rámci serveru je zcela odlišné.



Obrázek 1 - Oblíbenost serverových technologií (4)

Na tomto obrázku si můžeme všimnout procentuálního využití jednotlivých technologií, tak jak se v čase vyvíjel jejich trend. Dlouhou dobu byl jedním z nejpoužívanějších softwarů webový server Apache. Nyní se však v průběhu posledních let začíná čím dál více využívat serverové technologie Nginx. (5)

Důvody, pro které se stává Nginx využívanějším řešením, je architektura, ta je založena asynchronním přístupu. Díky tomu je schopen obsloužit více připojení s nízkým využitím paměti. Dalším plusem je reverzní proxy nebo load balancer. (6)

3.3 HTTP(S)

HTTP je základní protokol, jenž zprostředkovává komunikaci na internetu mezi webovým server a koncovým připojeným zařízením. Je určen k přenosu hypertextových dokumentů, kterými jsou HTML, XML nebo další podobné souborové formáty. (7)

Tento typ přenosu bohužel neumožnuje zabezpečení přenosu dat, neumožňuje šifrování dat a ani jejich integritu. To vedlo ke vzniku nové technologie **TLS (dříve SSL)**, která umožňuje bezpečnou komunikaci v rámci internetu. Spojení těchto dvou technologií je označováno jako **HTTPS**. (7)

3.3.1 Princip komunikace

Celá komunikace je založena na principu dotazů. Připojené zařízení posílá dotazy (requesty) směrem k webovému severu a ten posílá zpět odpovědi. Veškeré dotazy jsou na sobě nezávislé a není tak možno stoprocentně určit, zda spolu dva dotazy souvisí. Pro komunikaci se serverem je třeba využíti metod, které nám tuto službu zprostředkovávají. Mezi základní patří GET, POST, PUT, DELETE. Existují i další jako HEAD, TRACE atd. (7)

```
HTTP/1.0 200 OK
Date: Fri, 15 Oct 2004 08:20:25 GMT
Server: Apache/1.3.29 (Unix) PHP/4.3.8
X-Powered-By: PHP/4.3.8
Vary: Accept-Encoding,Cookie
Cache-Control: private, s-maxage=0, max-age=0, must-revalidate
Content-Language: cs
Content-Type: text/html; charset=utf-8
```

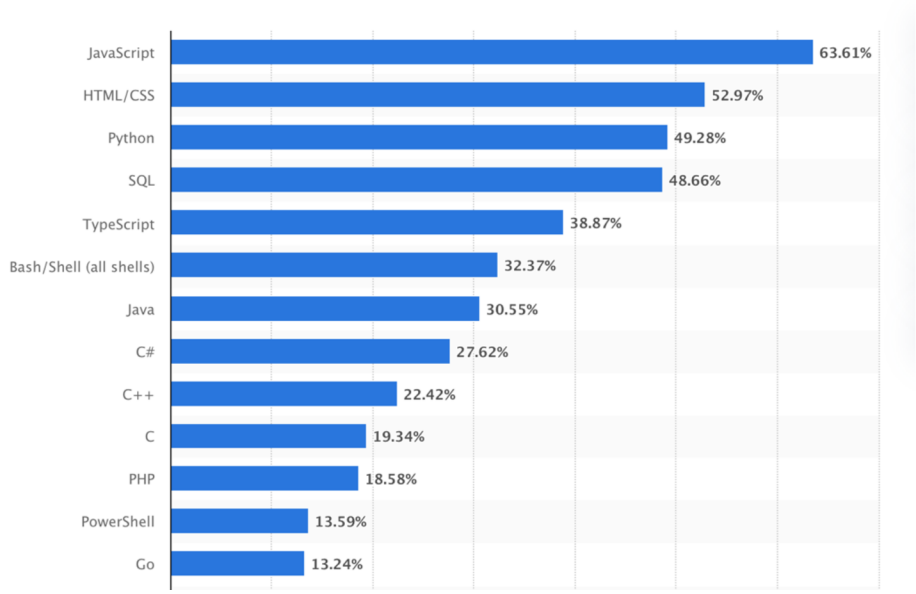
Obrázek 2 - Odpověď serveru (7)

GET je nejvyužívanější metodou ze všech zmíněných. Používá se pro získání dat nad určitým objektem. Může se například jednat o požadavky, které jsou definovány v URL adrese https://www.example.com/?prompt=file. (7)

POST je druhou neméně významnou metodou komunikace se serverem. Tato metoda je využívána při větším přenosu dat, zpravidla více než 512 bajtů nebo pokud není zcela vhodné přenášet data pomocí URL adresy směr k serveru. Využívá se typicky pro odesílání formulářů na webech nebo úpravu dat uložených na serveru. (7)

3.4 Fundamentální webové technologie

Na obrázku níže je vidět oblíbenost základních webových technologií, podle jejich oblíbenosti mezi vývojáři



Obrázek 3 - Oblíbenost webových technologií (8)

3.4.1 HTML

HTML neboli **Hypertext Markup Language**. Jedná se o tzv. značkovací jazyk, který nám pomáhá definovat strukturu webové aplikace. (9) Pomocí tohoto jazyka je možné v aplikaci vytvářet bloky, které jsou zobrazovány uživateli. Jelikož se jedná pouze o značkovací jazyk, bloky postrádají prvky designu. Některé bloky umožnují designovat aplikaci už na úrovni struktury, např. blok s kurzívou. Mimo jiné můžeme v aplikaci díky tomuto jazyku vytvářet např. odkazy nebo umisťovat videa.

3.4.2 CSS

CSS neboli **Cascading Style Sheets.** Kaskádové styly nám umožňují oddělit vzhled stránky od jejich obsahu a struktury. (10) Díky tomuto řešení je možno strukturovaným blokům vytvořit uživatelsky přívětivý vzhled. Blokům lze např. vytvořit nějaké pozadí, rámeček, bravu textu nebo tloušťku písma.

3.4.3 JavaScript

Objektově orientovaný programovací jazyk řízený událostmi, které se ve webové aplikaci objevují. (11) Využívá se pro interakci uživatele s určitými prvky v aplikaci. Typicky se může jednat o konkrétní akci, která se provede po kliknutí na element. Čistý JavaScript bez jakýkoliv knihoven nebo frameworků nevyžaduje funkci serveru, funguje nezávisle na něm i na klientu (tj. v prohlížeči).

JavaScript může být použit i na straně serveru, zejména pokud využíváme knihovny nebo frameworky, které pro své správné fungování vyžadují serverové

řešení. Typicky se jedná o frameworky, které spolupracují například s databázemi nebo zpracovávají média v reálném čase. Aplikace s takovými náročnými požadavky již nejsou schopny běžet pouze na straně klienta a je nutné využít výkon serveru.

Příklady JavaScriptových frameworků

- Rhino
- Node.js
- Express.js
- Next.js

3.4.4 PHP

PHP neboli **Hypertext Preprocessor** je programovací jazyk, který je určen pro tvorbu dynamických webových aplikaci. (12) Na rozdíl od Javascriptu je tento jazyk závislý na serveru a bez něho nemůže fungovat. Skripty jsou prováděny na straně serveru, přičemž k uživateli se dostává již pouze výsledná odpověď. Odpověď je zpracována před samotným načtením stránky, resp. vykreslením uživatelského obsahu. Tato technologie slouží primárně k zpracování dat. Může se jednat o dynamické načítání obsahu např. z databáze nebo souboru. Co se týče vykreslení obsahu, je již nutné využít dalších technologií, které byly zmíněny dříve.

Tento i všechny předešlé jazyky jsou nezávislé na platformě. Mezi operačními systémy se mohou objevit rozdíly, ty však nejsou nijak zásadní pro chod aplikace, ale vždy je potřeba s nimi počítat.

4 Princip testovaní

Princip testování lze rozdělit do několika částí. Tato kapitola popisuje historii a vznik metodik, které v průběhu vývoje aplikací vznikaly, a to jakým způsobem je možné dělit samostatné testování z pohledu aktéra.

4.1 Historie testování

Historie testování programů sahá až do samých počátků vývoje počítačové technologie. První generace programátorů byla často nucena testovat své kódy manuálně, přičemž chyby byly hledány a opravovány ručně.

S postupem času a s rostoucí složitostí počítačových systémů se testování programů stalo systematickým procesem. V 70. letech vznikly první testovací techniky a metody, zaměřené na zvyšování efektivity a účinnosti testování. Nástroje pro automatizaci testování začaly vstupovat do praxe, usnadňující opakované testování a identifikaci chyb. (14)

V 90. letech, s exponenciálním růstem softwarového průmyslu, se testování stalo klíčovou součástí vývojových procesů. Objevily se různé metodologie, včetně testování podle modelu V, který zdůrazňoval důležitost testování v každé fázi vývoje. (15)

V současné době, v éře agilního vývoje a DevOps, testování programů hraje klíčovou roli v zajištění rychlého a spolehlivého dodávání softwaru. Automatizace testování, testování na úrovni jednotek, integrační testování a testování na úrovni uživatelského rozhraní jsou jen některými z moderních přístupů, které vývojáři a testeři využívají k vývoji historie testování programů. (16)

4.2 Úvod do testování

Testováním aplikace se nazývá postup nebo proces, při kterém se určitými postupy kontroluje správný chod výsledné nebo ještě vyvíjené aplikace, tak aby se zajistil její bezproblémový chod a integrita celé aplikace. (17) V případě špatně otestované nebo vůbec netestované aplikace se může stát, že po nasazení do produkčního chodu aplikace začne vykazovat špatné chování a v horším případě únik dat. Všechny takové problémy je proto nutné odhalit již při jejím vývoji.

U menších projektů může a zpravidla testuje aplikaci přímo vývojář, případně zákazník. U větších projektů jsou na tuto práci již najímáni lidé tzv. **testeři**, kteří danou aplikaci testují.

4.3 Postup testování během vývoje

Prvním krokem, který je třeba učinit při vývoji, je stanovit si základní informace kterými jsou **vzhled**, **funkce** a **způsob testování**. Na základě tohoto rozhodnutí se odvíjí i samotná hodnota celé aplikace. Kromě samotného vývoje je **dalším krokem,** jak již bylo zmíněno, i **průběžné testování**, které může odhalit potencionální chyby už v jejich zárodku, a tím se vyvarovat pozdějším problémům. V případě větších projektů se vytváří testovací scénáře, podle kterých se aplikace testuje. (18)

Aby byla aplikace úspěšná je nutné se zaměřit na tato kritéria (18)

1. **Funkčnost** – Všechny části aplikace musí fungovat tak jak mají, respektive dle zadání. Např. odkazy a formuláře.
2. **Přehlednost** – Aplikace musí být přehledná a jednoduše ovladatelná.
3. **Zpracování chyby** – Zde je nutné vyřešit, jak se aplikace zachová, pokud dojde k nějaké chybě.
4. **Integrita** – aplikace musí být odolná vůči útokům z vnější
5. **Náročnost** – U aplikací většího rozsahu může dojít k zpracovávání velkého objemu dat, proto je nutné aplikaci podle toho uzpůsobit.

4.4 Základní kroky testování

4.4.1 Analýza aplikace

Před každým testováním webové aplikace je nutné si správně analyzovat daný software a funkcionalitu, která má být otestována, tak aby byl zajištěn optimální a bezproblémový test aplikace. V tomto případě může tester spolupracovat s vývojáři dané aplikace pro vytvoření správné úvodní analýzy.

4.4.2 Testovací scénář

Po analýze je třeba si vytvořit testovací scénáře. Tyto scénáře mohou simulovat chování (pohyb a interakci s prvky) uživatelů v aplikaci. Testovací scénář by měl obsahovat informace o testování, provedené úkony a vyhodnocení chování samotné aplikace. Ve zkratce se může jednat o posloupnost kroků, které budou provedeny v rámci testování dílčí části aplikace nebo aplikace jako celku.

Příklad testovacího scénáře

Název scénáře: Testování funkčnosti rezervačního systému pro letenky

Cíl scénáře: Otestovat funkčnost rezervačního systému pro letenky, správné integrace platební brány a správné záznamy objednávek v databázi.

Kroky scénáře:

- **Registrace nového uživatele**
  - Uživatel navštíví domovskou stránku rezervačního systému pro letenky.
  - Klikne na tlačítko "Registrace" nebo "Vytvořit účet".
  - Vyplní registrační formulář s povinnými informacemi (jméno, e-mail, heslo, adresa atd.)
  - Ověří si e-mailovou adresu pomocí potvrzovacího e-mailu.
  - Úspěšně se zaregistruje a přesměruje se na domovskou stránku přihlášení.
- **Přihlášení existujícího uživatele**
  - Uživatel navštíví domovskou stránku rezervačního systému pro letenky.
  - Klikne na tlačítko "Přihlásit" nebo "Moje konto".
  - Vyplní přihlašovací formulář s e-mailovou adresou a heslem.
  - Úspěšně se přihlásí a přesměruje se na domovskou stránku rezervačního systému.
- **Vyhledání letů:**
  - Přihlášený uživatel vyplní detaily svého cestování (datumy, destinace atd.).
  - Provede vyhledání dostupných letů podle zadaných kritérií.
  - Prochází různé možnosti letů a vybírá požadovaný let.

- **Rezervace letu:**
    - Uživatel vybere konkrétní let a klikne na tlačítko "Rezervovat".
    - Potvrdí počet cestujících a další podrobnosti rezervace.
    - Vybere požadované možnosti doplňkových služeb (např. přeprava zavazadel).
    - Klikne na tlačítko "Pokračovat k platbě".
- **Platba za rezervaci:**
    - Uživatel vybere způsob platby (např. platební karta, bankovní převod).
    - Vyplní potřebné údaje pro platbu.
    - Potvrdí platbu
    - Obdrží potvrzení o úspěšné rezervaci letu.
- **Ověření objednávky**
    - Správce rezervačního systému se přihlásí do administračního rozhraní.
    - Zkontroluje seznam nových rezervací letů.
    - Ověří, že nová rezervace letu byla úspěšně zaznamenána v systému.

Očekávaný výstup:

- Uživatel úspěšně projde procesem rezervace letu.
- Let byl úspěšně rezervován a platba byla provedena bez problémů.
- Administrátor rezervačního systému ověří, že nová rezervace letu byla zaznamenána v systému.

Závěr: Tento testovací scénář byl vytvořen pro ověření funkčnosti rezervačního systému pro letenky z hlediska uživatele i administrátora. Zahrnuje základní kroky procesu rezervace letu a kontrolu správné integrace platební brány a záznamů objednávek v databázi.

4.4.3 Vyhodnocení testování

Po provedení testovacího scénáře tester pečlivě zaznamená veškeré poznatky, problémy a možná zlepšení, na které během testování narazil. Po vyplnění všech důležitých kroků a údajů, tester vyhodnocuje test, zda byl úspěšný nebo ne. Případně vrací problémy zpět vývojáři k opětovné kontrole funkčnosti a opravě případných chyb.

4.4.4 Rozhodnutí o kvalitě

Posledním a nejdůležitějším krokem pro testera je rozhodnutí o kvalitě výsledné aplikace. V tomto kroku by tester měl rozhodnout, zda je aplikace připravena do reálného provozu nebo je nutné ji ještě opravit/upravit. V případě, že aplikace ještě není připravena do reálného chodu, je nutné chyby opravit a aplikaci opětovně testovat.

4.5 Manuální a automatické testování

Základním rozdělení, podle kterého můžeme testovaní dělit, je podle toho, jak testování aplikace probíhá. V tomto případě máme dvě možnosti.

4.5.1 Manuální testování

První možností, kterou lze využít, je manuální testování. Jedná se o jednodušší a v uvozovkách cenově méně náročnou možnost, jak web testovat. Bohužel s náročností aplikace stoupá i cena testování. U tohoto typu testování jsou potřebné pouze základní schopnosti a povědomí o tom, jak aplikace funguje, a není třeba větší technické znalosti.

Při tomto druhu testování tester (člověk, který web testuje) celou aplikaci prochází, proklikává a interaguje se všemi prvky, které se na webu nachází. Jelikož je v tomto druhu testování hlavním aktérem lidský faktor, může se stát, že něco přehlédne nebo špatně otestuje. Tím se chyba neodhalí, a tak se dostane až k uživateli. (19)

Při použití manuálního testování je důležité pamatovat na to, že tento proces vyžaduje čas a úsilí. Tester musí pečlivě projít každou část webu a prověřit její funkčnost. To může být náročné, zejména pokud je web rozsáhlý nebo obsahuje složité funkce.

Neméně důležitou věcí je mít vhodně vyškolené testery, kteří mají schopnost identifikovat různé typy chyb a chápat uživatelské potřeby a očekávání. Kvalifikovaný tester může pomoci zajistit, že web bude fungovat co nejlépe a bude poskytovat uživatelům pozitivní zkušenost. (19)

Důležité je, aby se manuální testování provádělo pravidelně a systematicky, aby bylo zajištěno, že všechny části webu jsou důkladně otestovány a že se případné chyby odhalí co nejdříve. Tím se minimalizuje riziko vzniku problémů a zlepšuje se celková kvalita a spolehlivost webové aplikace.

4.5.2 Automatické testování

Druhá z možností, jak je webovou aplikaci možné testovat, je automatickými testy. Při tomto druhu testování se využívají speciální programy nebo testovací skripty. Toto testování najde své uplatnění spíše u větších a náročnějších aplikací, protože zde je možnost ušetřit spoustu času a mnohdy i peněz. Jeho nástroje mají většinou jako výsledek tzv. report, zprávu, ve které jsou vypsány veškeré chyby, které software zachytil. Výhodou tohoto řešení oproti manuálnímu testování je neomylnost. Program **vždy** testuje stejně.

Automatické testy se postupně stávají klíčovým prvkem moderního vývoje softwaru, zejména v prostředích, kde je důležitá rychlost nasazení a nutné opakované testování. Díky nim lze automatizovat opakované testovací scénáře, což snižuje časovou náročnost testování a umožňuje vývojářům soustředit se na vývoj nových funkcí a vylepšení aplikace. (20)

Nicméně, je třeba mít na paměti, že automatické testy mají své limity. Ne všechny typy testování lze automatizovat a v některých případech je stále nezbytné provést manuální testování pro kontrolu uživatelského rozhraní a chování aplikace v různých situacích.

Další výzvou při použití automatických testů může být údržba testovacích skriptů a přizpůsobení se změnám v aplikaci. Je důležité pravidelně aktualizovat testovací skripty a sledovat změny v aplikaci, aby byla zachována jejich účinnost.

Mezi nejčastější automatické testy patří (21)

1. **Testování grafického uživatelského rozhraní** – testovací rozhraní vytváří uživatelské události, jako je stisknutí klávesy a na jejich základě testuje aplikaci, zda se chová tak jak má.

2. **Testování na základě API** – tento druh testování mnohdy obchází uživatelské rozhraní a testuje kód jako takový. Zde se může například testovat integrita tříd, pomocí argumentů a výsledných návratových hodnot.

4.6 Začlenění umělé inteligence

Generativní testování umělou inteligencí představuje posun v oblasti testování softwaru, který přináší nové možnosti a efektivitu do procesů vývoje. Tato inovativní metoda umožňuje testerům nebo přímo týmům odpovídajícím za výslednou kvalitu softwaru vytvářet komplexní testovací scénáře a generovat potřebná testovací data (22)

Výhoda generativního testování umělou inteligencí spočívá nejen ve zkrácení doby potřebné k provedení testů, ale také v jeho vysoké flexibilitě a modifikovatelnosti. Díky tomu jsou vývojáři schopni efektivně využívat testy a současně rozvíjet rozsah celého testování na vyšší úrovni. Tímto způsobem mohou lépe reagovat na změny a nové požadavky. Zabezpečí tak vyšší kvalitu výsledného softwaru.

Generativní testování umělou inteligencí umožňuje také automatizaci mnoha rutinních testovacích úkolů, což dále zvyšuje efektivitu testovacího procesu a umožňuje vývojářům věnovat více času kreativním a inovativním aspektům vývoje softwaru. Tímto způsobem se zlepšuje celková kvalita softwarových produktů a zároveň se snižují náklady spojené s testováním a vývojem. (22)

5 Druhy testů během vývoje a aktualizace

Provádění testů již během samotného vývoje je důležitým prvkem moderních aplikací, pro zaručení jejich maximální kvality a spolehlivosti. V této kapitole se zaměříme na druhy testů a jejich začlenění.

5.1 Smoke test

Jedná se o krátký test, který se většinou uskutečňuje ve chvíli, kdy je vývoj aplikace dokončen a je připravena na spuštění. Tento test se zaměřuje na kontrolu všech dílčích části, zda jsou správně implementovány a pracuji tak, jak mají. Test se zaměřuje pouze na hlavní funkce webu, které se tak často neupravují. Pokud není test automatizován musí se provádět ručně. U menších projektů se často provádí pouze tyto testy a další jsou opomíjeny. (23)

5.2 Regresní test

Regresní testovaní je založeno na opětovném testování aplikace po provedení aktualizace nebo implementace nové funkce. Testem se snažíme zjistit, zda „nový kód“, který byl implementován nemá vliv chod předchozích funkcí, které již byly v aplikaci implementovány dříve. (24) Tento druh testu se provádí u aplikací, které podléhají častým aktualizacím a je tak vždy nutné zkontrolovat funkčnost celé aplikace.

Dalšími aktualizacemi aplikace se mohou objevit nové nebo chyby, které byly již v předchozích verzích opraveny. Takové chyby mohou vzniknout nezavedenými nebo naopak špatně zavedenými postupy během verzování aplikace. Dobrým postupem pro lokalizaci takových chyb je právě vytvoření testu, který by odhalil potencionální chyby ihned po uvedení nové verze. Jelikož se jedná o opakovaný test, který se provádí zpravidla vždy po aktualizaci, tak se více než nabízí využití nějakého externího softwaru a tím pádem celkové automatizace testu. (25)

Testování a následná oprava mohou být ekonomicky nákladné. Z toho důvodu se zavádí další metodiky, které napomáhají odhalení chyb už během implementační fáze.

5.3 Funkční a nefunkční test

Funkční test ověřuje, zda aplikace funguje bez problémů jako celek. Během testu se kontrolují veškeré funkce webu, zda fungují správně a odpovídají představě a zadání zákazníka. (26)

Dle webu (2) **funkční testování** ověřuje, zda skutečně aplikace splňuje očekávaní zákazníka a splňuje danou specifikaci. Testují se jednotlivé části webu a jejich chování. Testují se veškeré funkční bloky jako např. tlačítka nebo formuláře, případně i cookies.

Funkční testování nemá přímo za cíl otestovat, zdali aplikace byla správně vyvinuta ve smyslu použití správných technologií, ale zda aplikace odpovídá zadání. To znamená, že nekontrolujeme nebo netestujeme samotný kód aplikace, ale pouze zda se aplikace chová podle zadání.

Nefunkční test tento se na rozdíl od ostatních nezabývá správnou funkcí webu, kterou může přímo ovlivnit uživatel. V tomto testu je hlavně kladen důraz na výkonnostní testování, respektive jak se aplikace chová při velkém zatížení. Díky tomuto testu jsme schopni odhalit, jak je aplikace schopna snášet vysoké zatížení a zda je připravena plnit svůj účel (26).

Podle (2) můžeme nazvat nefunkční testování výkonnostním testováním. Webová aplikace je podrobena nejvyššímu zatížení. Následně zjišťujeme hranice naší aplikace a její chování v zátěži.

V případě, kdy budeme mít e-shop na který chodí tisíce zákazníků každý den, tak na to aplikace musí být přizpůsobena. Je třeba počítat s velkým množství přenášených dat v jeden okamžik.

Tyto testy mají celkový vliv na správnou funkčnost aplikace v dlouhodobém horizontu. Nezřídka se tyto testy opomínají, což vede k pozdějším nepříjemnostem.

5.4 UNIT test

Jedná se o test, který prověřuje chování a funkčnost základních prvků aplikace. Vždy je ověřována malá část, tzv. jednotka (unit). Může se jednat o funkci, metodu, třídu nebo menší ucelený blok, který je možné testovat. Tento druh testu se často nachází na pomezí odpovědnosti testera a programátora. Ve většině případů provádí nebo píše test samotný programátor, který ověřuje, že se kód chová tak jak by měl. Díky své rychlosti a celkové jednoduchosti je tento druh testování mezi programátory oblíben. Testováním na úrovni jednotlivých bloků kódu je velmi snadné izolovat najít a izolovat potencionální chyby, které by v aplikaci mohli vzniknout. (25)

5.5 Integrační test

Integrační test je fáze testování, kdy je třeba otestovat, zda jednotlivé moduly aplikace, které jsou již spojeny ve větší celek, fungují dle očekávání. Tento test se provádí za účelem vyhodnocení kvality kódu a očekávané funkcionality aplikace. (25)

Test je nutné rozdělit na vnitřní a vnější integraci. Test vnitřní integrace spočívá v testování jednotlivých vnitřních částí dané aplikace. Na druhou stranu u vnější integrace je testováno propojení více již hotových řešení, které se skládají jako jednotlivé moduly a tím vytvářejí další a komplexnější aplikaci. Spojení takových modulů není vždy jednoduché, jelikož se jedná o různé moduly (aplikace) od různých vývojářů. (27)

5.6 Systémový test

Systémový test je využíván v pozdějších fázích vývoje aplikace. Zde je kladen důraz na testování z pohledu zákazníka. Jsou vytvořeny jednotlivé testovací scénáře, podle toho, jak by mohla být aplikace po uvedení do provozu využívána. Chyby, které byly nalezeny během testování je třeba opravit. Po opravě chyb následuje další vlna testování, které se opakuje až do opravení všech chyb. Poté již následuje testování na straně zákazníka. (17)

5.7 Akceptační testování

Poslední fází testovaní vyvíjené aplikace je akceptační testování. Zde se role testera ujímá samotný zákazník. Pokud veškeré předchozí testování proběhlo bez větších nedostatků, které by mohli ovlivnit její funkčnost, je aplikace předána na další testování zákazníkovi. Stejně jako v systémovém testování jsou testovány veškeré části aplikace, zda odpovídají představě a zda generují požadované výstupy. V případě, že je tester spokojen s celkovým fungováním aplikace a zároveň nebyla nalezena žádná zásadní chyba, je aplikace připravena k nasazení do reálného provozu. (17)

6 Vyhodnocení kvality

Vyhodnocení kvality testovaného softwaru je nedílnou součástí vývoje moderní webové aplikace i její následné údržby a rozvoje. Pro tento účel byly vytvořeny spousty softwarů, které mohou práci testera ulehčit. Tyto kritéria je možné rozdělit do několika skupin, které jsou v této kapitole popsány.

6.1 Testování funkčnosti

Testování funkčnosti webové aplikace je proces, během kterého se ověřuje, zda aplikace funguje správně a plní očekávané funkce. (28) Cílem je zajistit, že uživatelé mohou aplikaci používat bez problémů a že splňuje stanovené požadavky a specifikace.

6.1.1 Selenium

Selenium je open-source testovací nástroj pro automatizaci webových prohlížečů, který nabízí vývojářům a testerům flexibilitu v simulaci interakcí s webovými aplikacemi. Jeho klíčovým prvkem je možnost automatizace různých webových prohlížečů, včetně Chrome, Firefox, Internet Explorer a Safari, což zajišťuje cross-browser kompatibilitu. (29) Selenium podporuje také různé programovací jazyky, jako jsou Java, Python, C# a Ruby, což umožňuje vývojářům pracovat v preferovaném prostředí. Selenium se využívá především k automatizaci testů webových aplikací, od jednotkových testů až po testy přetížení.

WebDriver, který je stěžejním prvkem Selenium, poskytuje programovatelné rozhraní pro ovládání webových prohlížečů. (30) To zahrnuje automatizaci interakcí, jako jsou kliknutí na tlačítka, vyplňování formulářů a navigace na různé stránky.

Díky schopnosti integrace s různými testovacími frameworky, jako jsou JUnit, TestNG a pytest, umožňuje Selenium efektivní správu testů. Navíc podporuje paralelní spouštění testů na více prohlížečích nebo dokonce na různých strojích, což zvyšuje rychlost testovacích cyklů. Selenium poskytuje také různé selektory pro lokalizaci prvků na stránce a usnadňuje práci s webovými elementy. Je podporován aktivní komunitou a nabízí rozsáhlé zdroje, tutoriály a školení, což zvyšuje efektivitu práce vývojářů využívat Selenium. (31)

6.1.2 Cypress

Cypress je open-source nástroj navržený pro **end-to-end** testování webových aplikací, který nabízí moderní a uživatelsky přívětivé prostředí pro automatizaci testů, zejména pro webové aplikace postavené na moderním JavaScriptu. (32)

Jeho instalace a použití jsou jednoduché a snadné díky balíčkovacímu systému npm a přehledné dokumentaci. Co dělá Cypress unikátním, je jeho schopnost poskytovat **realtime** sledování průběhu testů, což umožňuje vývojářům okamžitě vidět každý krok testu a případné chyby. Další výhodou je **automatický reload** (přenačtení) stránky po každé změně kódu, což urychluje testovací iterace. Cypress je optimalizován pro moderní JavaScript frameworky, včetně React, Angular a Vue.js, a podporuje paralelní spouštění testů. (32)

S Cypressem lze snadno navigovat a manipulovat s DOM, a díky automatické detekci chyb mohou uživatelé rychle identifikovat nesrovnalosti v chování aplikace. Navíc podporuje API testování a má vysokou komunitní aktivitu, což znamená pravidelné aktualizace a přidávání nových funkcí. Celkově lze říci, že Cypress poskytuje moderní prostředí pro efektivní automatizaci **end-to-end** testů webových aplikací, přičemž klade důraz na jednoduchost, rychlost a efektivitu diagnostiky chyb. S jeho unikátními vlastnostmi se stal oblíbeným nástrojem mezi vývojáři a testery v oblasti webového vývoje. (32)

6.2 Testování zobrazení

Jelikož je možné přistupovat k webovým aplikacím z různých druhů zařízení, ať už se jedná o počítač, telefon, tablet a další zařízení, je vždy nutné otestovat, jestli se daná aplikace zobrazuje napříč všemi zařízeními, tak jak by měla. V dnešní době již existuje mnoho různých nástrojů, které nám mohou testování velice zjednodušit. Díky těmto nástrojům nemusíme vlastnit desítky zařízení a testovat aplikaci na každém z nich samostatně.

Během testování zobrazení musíme myslet i na programy, prohlížeče, na kterých je aplikace zobrazována. Každý prohlížeč může naši aplikaci zobrazit trochu jinak. V mnohých případech se jedná o kaskádové styly, které způsobují největší problémy. Zatímco prohlížeče jako Chrome, Opera nebo Microsoft Edge běží na jádře Chromium, ostatní prohlížeče jako Firefox nebo Safari mají své vlastní jádro. Pro příklad lze využít kaskádový **selektor :has()**. Zatímco prohlížeče na jádře Chromium

měli plnou podporu pro tento selektor už nějakou dobu, Firefox teprve s podporou přicházel a tím vznikaly problémy napříč prohlížeči. (33)

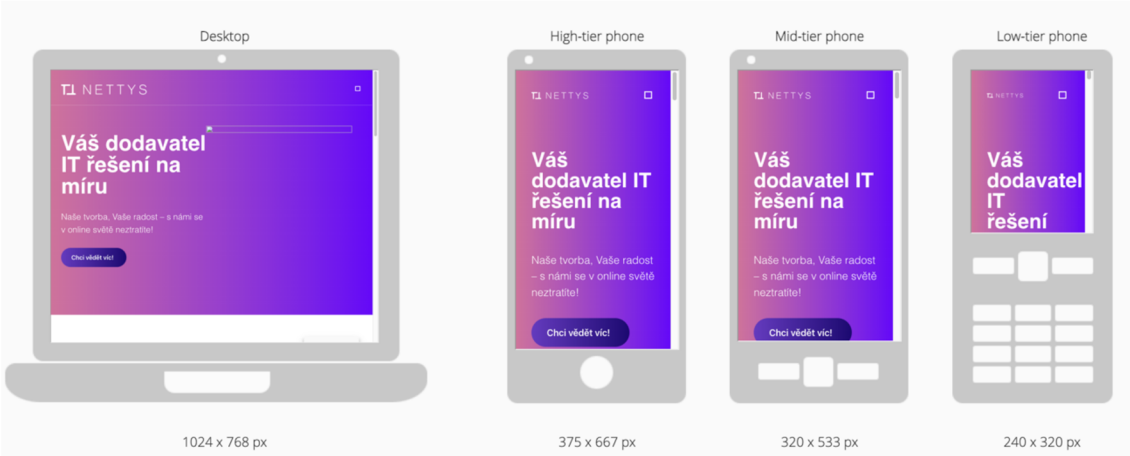
6.2.1 BrowserStack

Online aplikace, která umožnuje testerům zobrazit testovanou webovou aplikaci napříč různými zařízeními. Jedná se o mobilní telefony různých značek, tablety a veškerá rozlišení na desktopu. Zde se nejedná přímo o test aplikace ve smyslu jejího korektního chování, ale o správné zobrazení všech prvků v dané aplikaci.

Výhodou této aplikace je její jednoduchost a přívětivost. Vše funguje online bez jakýkoliv nutných instalací dodatečného softwaru. Testerovi stačí vložit URL adresu nástroje a přepínat se mezi zařízeními na kterých bude aplikace využívána.

6.2.2 MobiReady

Online nástroj, který je možné využít pro testování správného zobrazení webové aplikace. Jedná se o velmi jednoduchý nástroj, který zobrazí danou aplikaci, napříč 4 zařízeními. 3 telefony a desktop. Součástí nástroje je i skóre, které může pomoci k odhalení špatného rozložení webu. To ovlivňuje celkový pocit a vjem uživatele. Součástí výsledného skóre je také kontrola základních prvků na webu, kontrola základních tagů, kontrola správně umístěných skriptů nebo přebytečné kaskádové styly.



Obrázek 4 - Ukázka MobiReady
Zdroj: vlastní zpracování

6.3 Testování SEO a přístupnosti

Testování SEO a přístupnosti webových stránek je klíčovým procesem v oblasti optimalizace webové aplikace. Analýza těchto faktorů je kritická pro zlepšení organického vyhledávání a zajištění, že webová aplikace je přístupná a použitelná pro

všechny uživatele, včetně těch s různými formami tělesného nebo mentálního postižení. (34)

Prvním krokem, který je třeba udělat, je **analýza struktury a kontextového obsahu** aplikace. Pokud má být webová aplikace relevantní z pohledu vyhledávačů je potřeba provést analýzu klíčových slov a technických detailů stránky. Mezi tyto detaily spadá kontrola struktury URL, kódování aplikace, responzivní design a rychlosti načítání. **Obsahová analýza** se zaměřuje na hodnocení relevance a kvality obsahu v rámci klíčových slov a konkrétních vyhledávacích dotazů. Výběr vhodných klíčových slov a vyhledávacích frází, může velmi zásadně ovlivnit pozici ve vyhledávání a obecně zlepšit viditelnost webové aplikace. Díky této analýze jsou firmy lépe schopny cílit na své zákazníky, což vede k vyšším konverzím. (35)

Testování přístupnosti se zaměřuje na hodnocení dostupnosti stránek pro uživatele se zrakovými, sluchovými, motorickými nebo jinými omezeními. To zahrnuje ověření, zda jsou stránky správně interpretovány čtecími zařízeními, zda jsou navigační prvky snadno použitelné pomocí klávesnice a zda jsou dostupné alternativní texty pro multimediální obsah. (34)

Tyto analýzy a testy poskytují komplexní data a hloubkové poznatky, které poslouží jako solidní základ pro formulaci detailních doporučení a strategií. Hlavním cílem je vylepšit výsledky SEO a zajistit optimální přístupnost a uživatelský zážitek na webových stránkách.

6.3.1 Screaming Frog SEO Spider

Nástroj vytvořený společností Screaming Frog pro testování SEO a struktury webových aplikací. Pro použití je nutné si program nainstalovat a následně spouštět na svém vlastním zařízení. Jeho obsluha je velmi jednoduchá. Po zadání URL adresy do vyhledávacího pole, program spustí prohledávání webu, a to až do výše 500 navštívených URL adres, které jsou na webu dostupné. Do těchto adres se započítávají stránky, obrázky nebo soubory nutné pro fungování aplikace, kterými jsou kaskádové styly a skripty. Pro větší množství je třeba si program zakoupit.

Po skončení prohledávání webové aplikace se na obrazovce objeví veškeré URL adresy, které program testoval. Nemusí se nutně jednat pouze o interní adresy, ale i funkčnost odkazů na další aplikace. Během prohledávání také dochází ke kontrole dalších aspektů stránky, kterými jsou například správné pořadí nadpisů, popisky k obrázků, kanonické URL, a **hlavně klíčová slova a fráze pro SEO** (36)

6.4 Testování výkonu

Testování výkonu je proces, který se zaměřuje na ověření rychlosti, škálovatelnosti, stabilitu a obecně na výkonnostní vlastnosti softwarového systému, včetně webových aplikací. Cílem je identifikovat a řešit případné problémy spojené s odezvou systému, jeho zátěží a výkonem za různých podmínek. (37)

6.4.1 Load testing

Load testing nám pomáhá určit, jak se software chová a jak udrží svůj výkon při simulovaných zátěžových podmínkách. Testování může zahrnovat simulaci mnoha uživatelů, zvýšený datový objem nebo náročné operace, které by mohly celý systém zpomalit. Výsledkem toho testu je odhalit hranice našeho systému a díky tomu určit nedostatky, kterými mohou být špatná práce s pamětí, nedostatečný výkon serveru, špatná optimalizace atd. Test se provádí s různými vstupními proměnnými, tak aby bylo možné výsledky následně vyhodnotit a získat potřebné informace. (38)

6.4.2 Stress testing

Jedná se o formu testování výkonu, která se zaměřuje na zkoumání chování softwarového systému za extrémních podmínek, při maximální nebo nadměrné zátěži. Cílem stresového testování je ověřit, jak se systém chová, když je přetížen a přesahuje své normální limity. Testuje, zda systém zvládá nebo se adekvátně chová v neobvyklých nebo extrémních situacích, jako jsou vysoké úrovně zátěže, nedostupnost zdrojů nebo neočekávané chyby. (39)

Díky testování jsme schopni identifikovat potenciální nedostatky v systému a zjistit, kdy a proč selhává, což nám umožňuje předcházet potenciálním problémům ve skutečném provozu. Výsledky testování jsou klíčové pro optimalizaci systému a zajištění jeho robustnosti a odolnosti vůči neočekávaným situacím. Následně můžeme provést úpravy a optimalizace systému tak, aby byl schopen efektivněji zvládat zátěž a minimalizovat riziko výpadků nebo selhání v reálném provozu.

6.4.3 Scalable testing

Škálovatelné testování je důležitým prvkem moderního webového vývoje, které umožňuje efektivní a spolehlivé ověřování funkčnosti a kvality softwarových aplikací. Tento koncept se opírá o princip automatizace testovacích procesů a jejich schopnost přizpůsobit se změnám v rozsahu a složitosti testovaného softwaru. (40) Jedním

z hlavních cílů škálovatelného testování je minimalizace lidského úsilí a času potřebného k provedení testů, což vede k výraznému zvýšení efektivity a úsporám nákladů v průběhu vývoje softwaru.

V praxi se škálovatelné testování projevuje využitím automatizovaných nástrojů, technik a postupů, které umožňují opakované spouštění testovacích scénářů a skriptů na různých úrovních abstrakce softwaru. Využívá se již od jednotkových testů až po integrační či systémové testování. Tento přístup umožňuje efektivní detekci a odhalení chyb, již v brzkých fázích vývoje, což v konečném důsledku velice efektivně snižuje riziko vzniku chyb v produkčním prostředí. Díky tomuto postupu se zvyšuje celková spolehlivost a kvalita vyvíjeného produktu. Důležitou součástí škálovatelného testování je také jeho schopnost přizpůsobit se změnám v požadavcích na testování a zvládnout nárůst objemu testovacích dat a scénářů bez ztráty svojí efektivity a výkonu.

6.4.4 Endurance testing

Endurance testing, také nazýváme jako stability testing nebo soak testing. Jedná se o typ testování vyvíjených systémů, který se zaměřuje na ověření jejich schopnosti udržet stabilní výkon a správnou funkčnost za dlouhodobého provozu nebo při vyšším vytížení v rámci delšího časového horizontu. (41) Cílem endurance testování je identifikovat a řešit problémy, které se mohou objevit během dlouhodobého provozu, jako jsou úniky paměti, chyby v alokaci zdrojů nebo degradace výkonu. Tyto problémy se často neprojeví během krátkodobých nebo nízko zátěžových testů, ale mohou způsobit vážné problémy v produkčním prostředí, pokud nejsou odhaleny a řešeny včas. (42)

Při provádění endurance testů se často simulují reálné pracovní podmínky, kterým bude softwarový systém vystaven v produkčním prostředí, a to buď pomocí automatických nástrojů nebo manuálně vytvořených scénářů. Tento typ testování je zvláště důležitý pro kritické systémy, které musí fungovat nepřetržitě bez výpadků, jako jsou například systémy pro online bankovnictví nebo lékařské informační systémy. Výsledky endurance testů poskytují vývojářům a testerům důležité informace o stabilitě a spolehlivosti systému za dlouhodobého provozu, což umožňuje identifikovat a řešit případné nedostatky ještě před nasazením do produkčního prostředí a minimalizovat tak riziko vzniku výpadků nebo chyb během reálného provozu.

6.4.5 PageSpeed Insights

Komplexní nástroj, který byl vytvořen společností Google, je určen k analýze výkonu webových stránek. Tento nástroj poskytuje uživatelům vhled do toho, jak se jejich webové stránky prezentují a fungují, což jim umožňuje optimalizovat jejich výkon a zlepšit uživatelskou zkušenost. Zaměřuje se na čtyři klíčová kritéria: **výkon**, **přístupnost**, **doporučené postupy** a **SEO**. Každé z těchto kritérií je ohodnoceno skóre z rozmezí 0-100, platí čím vyšší číslo tím lépe.

Zaměříme se na **výkonnostní analýzu**, ta slouží k testování rychlosti načítání webové aplikace a rychlosti odpovědi serveru na které je aplikace hostována. Metrika je rozdělena do 2 částí. První je výkon aplikace na mobilním zařízení, druhá na počítači.

Jednou z metrik, kterou nástroj měří je **First Contentful Paint (FCP)**. Díky tomuto testu lze zjistit, jak rychle se vykreslí první ucelený obsah na stránku. (43)
Další metrikou je **Largest Contentful Paint (LCP)**, pomocí tohoto lze jednoduše optimalizovat část aplikace, která způsobuje příliš dlouhé načítání. **Cumulative Layout Shift (CLS)** ukazuje změny v rozložení aplikace během načítání. (43)

Pod metrikami jsou vidět podrobné informace a body ke zlepšení. Typicky se jedná o špatně optimalizované obrázky, tyto obrázky mají mnohdy zbytečně velkou velikost nebo u nich není definovaná výška a šířka.

Druhé kritérium je **přístupnost**, ta zkoumá, jak dobře je stránka přístupná uživatelům se zdravotním postižením, s důrazem na správné používání značek a atributů pro přístupnost.

Třetím kritériem jsou **doporučené postupy**. Ty hodnotí soulad webové stránky s doporučenými postupy, zahrnující například používání bezpečného protokolu HTTPS a správnou konfiguraci serveru.

Posledním kritériem je **SEO**, které hodnotí optimalizaci stránky pro vyhledávače, s doporučeními pro vytváření unikátních titulků, meta popisků a optimální struktury stránky. (43)

6.5 Testování API

Testování API je zásadním procesem při vývoji softwaru, který využívá pro své fungování komunikaci mezi různými systémy. Cílem tohoto testu je ověřit správnou funkčnost, výkonnost a bezpečnost rozhraní. (44) Což je klíčové pro zajištění kvality a spolehlivosti celkového softwarového ekosystému, v případě že naše aplikace je závislá na aplikacích třetích stran nebo někdo jiný je závislý na naší aplikaci.

Prvním krokem, jenž musíme udělat v testování API je plánování. To zahrnuje stanovení cílů, rozsahu a priorit testů. Důležité je pečlivě analyzovat požadavky a specifikace API, aby bylo možné efektivně navrhnout testovací scénáře. Správné navržení testovacích scénářů je kritické, protože každá aplikace, resp. jejich API mohou být odlišné. Tyto scénáře poté pokrývají různé aspekty funkčnosti, použití rozhraní, HTTP metody nebo různé typy vstupních dat a jejich očekávané výstupy.

Automatizace těchto testů hraje zásadní roli zejména při opakovaném provádění rozsáhlých testovacích sad. K tomuto účelu byly vytvořeny různé nástroje, mezi něž patří například **Postman**. Tato aplikace umožňuje automatizovat testování a poskytuje možnosti automatizovaného ověřování funkčnosti, výkonnosti a bezpečnosti API.

6.5.1 Postman

Postman je vývojářský nástroj, který umožňuje efektivní přístup k práci s API, testování a dokumentaci rozhraní pro programování aplikací (API). Umožňuje snadné vytváření API požadavků pomocí různých HTTP metod, kterými jsou GET, POST a PUT. A následně jejich odesílání na cílovou adresu URL s přidáním parametrů a hlaviček podle potřeby. Odpovědi od serveru jsou ihned zobrazeny na obrazovce, což usnadňuje rychlé testování a diagnostiku případného problému.

Další z funkcí, kterou aplikace nabízí je možnost organizace souvisejících požadavků do **kolekcí** pro lepší orientaci, přehlednost a spolupráci v týmu. K dotazům a kolekcím je možné přidávat poznámky. (45) Díky tomu je práce s nástrojem ještě efektivnější. Poté už můžeme snadno vytvářet a spouštět automatické testy, které ověří správné chování.

Rozdělení práce mezi různými servery umožňují tzv. **prostředí**. Tímto způsobem mohou uživatelé snadno přepínat mezi různými servery a provádět testy API bez nutnosti ručně měnit konfigurace nebo upravovat požadavky. To usnadňuje práci s různými verzemi, testováními a vývojovými a produkčními prostředími.

Důležitou a neméně podstatnou funkcí je také **generátor dokumentace API** z existujících požadavků a kolekcí, což usnadňuje sdílení informací v týmu a dokumentaci API. **Postman Cloud** pak umožňuje sdílení a synchronizaci kolekcí a prostředí mezi uživateli, exportování a importování dat a poskytuje nástroje pro sledování výkonu API, což je důležité pro diagnostiku a optimalizaci. (45)

7 Instalace testovacích nástrojů

Tato kapitola je věnována představení instalace jednotlivých testovacích frameworků, od jejich stažení až po samostatné spuštění.

7.1 Selenium

Nástroj pro automatické testování webových aplikací. Skládá z několika navzájem se doplňujících komponent, kterými jsou **Selenium IDE, Selenium RC, Selenium WebDriver a Selenium Grid**. Tento program je vytvořen v programovacím jazyce Java a je možné ho spouštět na různých platformách. (46)

7.1.1 Instalace Java Development Kit (JDK)

Jedná se o sadu nástrojů potřebných pro vývoj aplikací v jazyce Java. Obsahuje kompilátor, který převádí zdrojový kód Java do bytecode, a také další nástroje, které umožňují vývojářům vytvářet, ladit a spouštět Java aplikace. Před instalací Selenium je nejprve nutné JDK stáhnout ze stránek společnosti Oracle a následně nainstalovat. (47)

7.1.2 Nastavení proměnných prostředí

Po úspěšné instalaci JDK je nutné nastavit proměnné prostředí tak, aby systém věděl, kde se JDK nachází. Nastavit je třeba dvě základní proměnné JAVA_HOME a PATH. JAVA_HOME ukazuje na kořenový adresář nainstalovaného JDK a PATH obsahuje cestu k binárním souborů JDK.

7.1.3 Stažení a instalace Selenium WebDriver

Základním stavebním kamenem pro automatizaci testování webových aplikací v jazyce Java je **Selenium WebDriver**. Pro jeho integraci do projektu je vhodné použít správce závislostí **Maven**, který umožňuje snadnou správu externích knihoven. Pomocí závislosti na Selenium WebDriver v **pom.xml** projektu lze jednoduše stáhnout a integrovat tuto knihovnu do projektu.

7.1.4 Stažení a konfigurace webového ovladače

Pro správnou interakci s webovým prohlížečem pomocí Selenium je nezbytné mít nainstalovaný odpovídající webový ovladač. V případě testování webových aplikací s použitím prohlížeče **Google Chrome** je potřeba stáhnout ovladač **ChromeDriver**, který umožňuje ovládání prohlížeče Chrome pomocí Selenium WebDriver. Po stažení je nutné specifikovat cestu k ovladači v konfiguraci testů. (48)

Tímto způsobem je možné úspěšně implementovat a nakonfigurovat Selenium v jazyce Java pro automatizované testování webových aplikací. Pečlivá instalace a konfigurace Selenium je klíčová pro úspěšné provedení automatizovaných testů.

7.2 Cypress

Před samotnou instalací testovacího nástroje Cypress je nutné nainstalovat prostředí Node.js. Prvním krokem, který je třeba udělat, je stáhnout a nainstalovat ideálně nejnovější verzi Node.js z oficiálních stránek. Díky již vytvořenému instalačnímu balíčku je instalace velmi jednoduchá. Součástí Node.js je také npm. Tento správce balíčků pro vývojáře umožňuje vyhledávat, instalovat nebo odstraňovat balíčky. Balíčky jsou umístěny v centrálním repositáři, který je pro vývojáře volně dostupný. Po úspěšném dokončení instalace je důležité ověření funkčnosti Node.js a npm. Pro ověření je třeba spustit příkazový řádek nebo terminál, v závislosti na operačním systému. K tomu slouží 2 základní příkazy node -v a npm -v. Díky nim lze zjistit aktuální verze Node.js a npm. Pokud tyto příkazy vracejí očekávané verze, je Node.js funkční a instalace Cypress může pokračovat. (49)

7.2.1 Instalace Cypress

Po úspěšné instalaci všech potřebných technologií je možné přistoupit k instalaci samotného nástroje. Opět je třeba využít příkazového řádku, jelikož instalace Cypress probíhá právě pomocí správce balíčků **npm**. Příkazový řádek je nutné otevřít přímo ve složce, kde se má nástroj nainstalovat a zadat instalační příkaz **npm install cypress --save-dev.** Pokud instalace proběhla v pořádku můžeme spustit uživatelské prostředí pomocí příkazu **npx cypress open.**

7.2.2 Nastavení a spuštění testů

V tuto chvíli je již vše nainstalované a připravené pro testování. Dalším krokem, který je nutné udělat, je připravit si testovací scénáře. Testovací scénáře se nacházejí ve složce **cypress/e2e**, tu je v první řadě nutné vytvořit. Poté se do ní budou umisťovat testovací soubory. Po umístění souborů do složky je již možné spouštět testy přímo z uživatelského prostředí a sledovat jejich průběh. (50)

8 Ukázka testování

Tato kapitola je zaměřena na ukázku automatického testování. Veškeré funkce, které jsou zmíněny v kapitole a jejich popis pochází z dokumentace testovacích frameworků.

8.1 Test přihlašovacího formuláře

Název scénáře: Ověření funkcionality přihlašovacího formuláře
Cíl scénáře: Zjistit, zda přihlašovací formulář na webové stránce funguje korektně a uživatel se může úspěšně přihlásit bez zobrazení chybových zpráv.
Kroky scénáře:

1. Přístup na stránku s formulářem:
    1.1. Uživatel otevře webovou stránku s přihlašovacím formulářem.
2. Vyplnění platných údajů:
    2.1. Uživatel vyplní pole "Uživatelské jméno" platným uživatelským jménem.
    2.2. Uživatel vyplní pole "Heslo" platným heslem.
3. Odeslání formuláře:
    3.1. Uživatel klikne na tlačítko "Přihlásit".
4. Ověření úspěšného přihlášení:
    4.1. Uživatel ověří, že se po úspěšném přihlášení nezobrazuje žádná chybová zpráva.

Očekávaný výstup: Uživatel se úspěšně přihlásí na webovou stránku bez zobrazení chybových zpráv.
Závěr: Tento testovací scénář byl vytvořen pro ověření funkcionality přihlašovacího formuláře na webové stránce. Zahrnuje kroky procesu přihlášení a kontrolu správného chování formuláře v reakci na poskytnuté platné údaje.

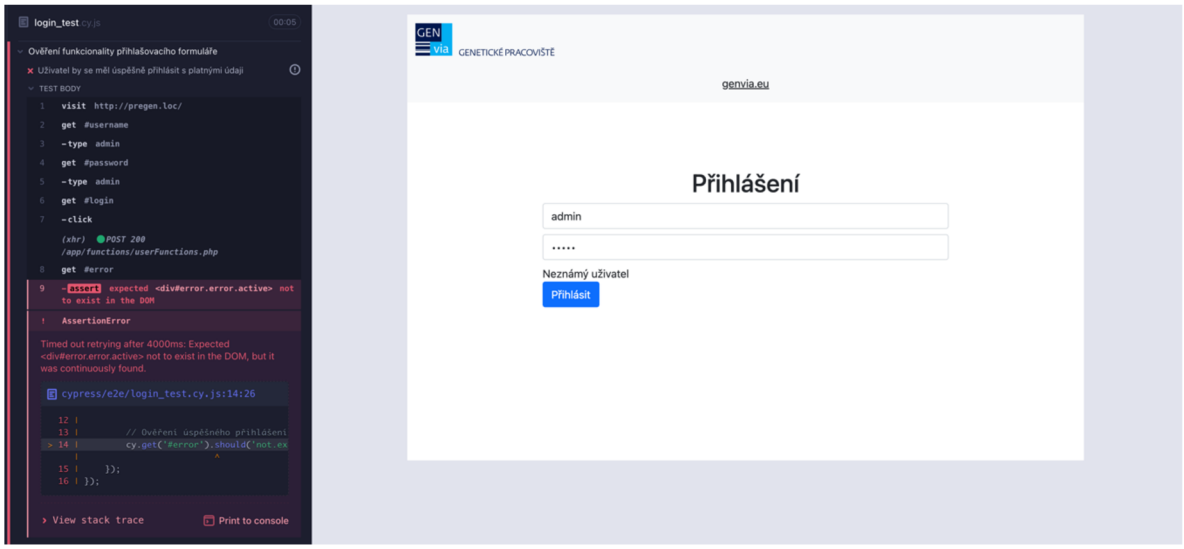
8.1.1 Cypress

```
describe( title: 'Ověření funkcionality přihlašovacího formuláře', fn: () : void => {
  it( title: 'Uživatel by se měl úspěšně přihlásit s platnými údaji', config: () : void => {
      // Přístup na stránku s formulářem
      cy.visit( url: 'http://pregen.loc/');

      // Vyplnění platných údajů
      cy.get( selector: '#username').type( text: 'admin');
      cy.get( selector: '#password').type( text: 'admin');

      // Odeslání formuláře
      cy.get( selector: '#login').click();

      // Ověření úspěšného přihlášení
      cy.get( selector: '#error').should( chainer: 'not.exist');
  });
});
```

Obrázek 5 - Ukázka testování přihlášení v Cypress
Zdroj: vlastní zpracování

Jednotlivé kroky testu jsou reprezentovány funkcemi nástroje Cypress.io v nástavbě jazyka JavaScript. Funkční část celého testu začíná funkcí *cy.visit()* ta nám umožňuje přístup na webovou stránku na které se nachází testovaný přihlašovací formulář. Po otevření dané URL, test přechází k další funkci *cy.get().type()*, ta slouží k vyplnění námi zadaných přihlašovacích údajů do formuláře. Po vyplnění údajů je provedeno odeslání formuláře pomocí funkce *cy.get().click().* Závěrečným krokem je ověření úspěšného přihlášení, což se děje prostřednictvím funkce *cy.get().should()*, která kontroluje, zda na stránce není přítomna chybová zpráva.

Níže uvedený obrázek ukazuje situaci, kdy došlo k **chybnému přihlášení**. Místo očekávaného přesměrování do aplikace po zadání správných přístupových údajů se objevila hláška "Neznámý uživatel". Tuto událost zaznamenal i nástroj Cypress, jak je patrné v levém sloupci, kde je zaznamenáván průběh testování v reálném čase. Konkrétně jsme se setkali s chybou, kterou kontroluje funkce *cy.get('#error').should('not.exist').* Tento scénář vedl k zadání nesprávných údajů a následnému zobrazení chybového hlášení.



Obrázek 6 - Ukázka zachycení chyby v Cypress
Zdroj: vlastní zpracování

8.1.2 Selenium

Tento testovací kód automatizuje ověření správné funkcionality webového přihlašovacího formuláře v nástroji Selenium. Nejprve se pomocí anotace *@Before* nastaví cesta k ovladači pro Chrome a konfigurují se další možnosti prohlížeče. Poté je vytvořena instance třídy ChromeDriver s použitím předchozích nastavení a přiřadí se do proměnné "driver". Samotný test, označený anotací *@Test*, začíná navigací na URL adresu webové stránky s přihlašovacím formulářem. Následně jsou vyplněny platné přihlašovací údaje a formulář je odeslán. Poté se pomocí metody *assertFalse* ověřuje, zda na stránce není zobrazena chybová zpráva, což signalizuje úspěšné přihlášení. Po dokončení testu, označeného anotací *@After*, je relace s WebDriverem ukončena, což zahrnuje i zavření prohlížeče Chrome.

Chybné přihlášení je identifikováno přítomností chybové zprávy na webové stránce po pokusu o přihlášení. Testovací kód ověřuje tuto situaci pomocí metody *assertFalse*, která kontroluje, zda není zobrazen prvek s identifikátorem "error". V případě, že je chybová zpráva zobrazena, test selže a vyhodí chybu, jelikož očekává, že prvek s chybou nebude přítomen. Tento výstup je pak zobrazen v konzoli vývojového prostředí.

```java
@Before
public void setUp() {
    // Nastavení WebDriveru pro Chrome
    String chromeDriverPath = "/Users/jantheodor/IdeaProjects//chromedriver-mac-arm64/chromedriver";
    System.setProperty("webdriver.chrome.driver", chromeDriverPath);
    ChromeOptions options = new ChromeOptions();
    options.addArguments("--start-maximized");
    options.setCapability( capabilityName: "chrome.verbose",  value: false);
    options.setCapability( capabilityName: "whitelistedIps",  value: "");
    this.driver = new ChromeDriver(options); // Assign to class member this.driver
}

@Test
public void testSuccessfulLogin() {
    // Přístup na stránku s formulářem
    driver.get("http://pregen.loc/");

    // Vyplnění platných údajů
    WebElement usernameField = driver.findElement(By.id("username"));
    usernameField.sendKeys( ...charSequences: "admin");

    WebElement passwordField = driver.findElement(By.id("password"));
    passwordField.sendKeys( ...charSequences: "admin");

    // Odeslání formuláře
    WebElement loginButton = driver.findElement(By.id("login"));
    loginButton.click();

    // Ověření úspěšného přihlášení
    assertFalse(driver.findElement(By.id("error")).isDisplayed());
    }

    @After
    public void tearDown() {
        // Ukončení relace WebDriveru
        if (driver != null) {
            driver.quit();
        }
}
```

Obrázek 7 - Ukázka testování přihlášení v Selenium
Zdroj: vlastní zpracování

8.2 Test akce 3+1

Druhým testem, který bude demonstrován je testování akce na internetovém obchodu. Jedná se o mnohem komplexnější test, který již zahrnuje i asynchronní načítání.

Název scénáře: Ověření funkcionality akce 3+1
Cíl scénáře: Zjistit, zda akce 3+1 na webové stránce funguje korektně a cena produktů se správně snižuje.
Kroky scénáře:

1. Přístup na webovou stránku:
    1.1. Uživatel navštíví webovou stránku http://pepe.nettys.cz/.
2. Výběr provozovny:
    2.1. Uživatel klikne na první tlačítko pro výběr města.
3. Přidání produktu do košíku:
    3.1. Uživatel klikne na produkt v sekci “Tomatový základ” a vybere pizzu ve variantě střední.
    3.2. Uživatel vyčká, až se zobrazí počet položek v košíku.
4. Odeslání objednávky:
    4.1. Uživatel klikne na tlačítko pro odeslání objednávky
5. Ověření přesměrování:
    5.1. Uživatel ověří, že byl přesměrován na stránku košíku (URL obsahuje "/kosik.php").
    5.2. Ujistí se, že se košík načetl úspěšně.
6. Ověření akce 3+1:
    6.1. Uživatel ověří, že v košíku je zobrazena akce 3+1.
    6.2. Uživatel se ujistí, že **akce 3+1** je aktivní a cena produktů byla správně snížena.

Očekávaný výstup: Uživatel úspěšně přidá produkt do košíku, bude přesměrován na stránku košíku a zjistí, že cena produktů se po aplikaci akce 3+1 správně snížila.
Závěr: Tento testovací scénář slouží k ověření funkcionality akce 3+1 na webové stránce. Zahrnuje kroky od výběru produktu až po ověření, že cena produktů byla snížena podle očekávání.

Ve třetím bodě testovacího scénáře je již implementováno asynchronní načítání, dochází zde k načtení položek menu pro konkrétní kategorii. Zde se již velice výrazným způsobem rozchází celkový princip implementace automatického testu.

Cypress již ze své podstaty je uzpůsoben pro takovýto princip načítání, automaticky čeká určitý čas bez další složitější implementace.

Selenium bohužel spoléhá na již načtený DOM a je tak nutné takovému načítání přizpůsobit. Na obrázku je znázorněný jeden z možných způsobů implementace.

```
WebDriverWait waitKosik = new WebDriverWait(driver,  timeOutInSeconds: 60);
WebElement submitButton = driver.findElement(By.cssSelector(".modal--items--submit.modal--items--submit--margin"));
Actions actions = new Actions(driver);
actions.moveToElement(submitButton).click().perform();
waitKosik.until(ExpectedConditions.urlContains( fraction: "/kosik.php"));
```

Obrázek 8 - Ukázka čekání v Selenium
Zdroj: vlastní zpracování

Nejprve je nutné inicializovat **WebDriverWait**, který umožňuje čekání na určitý stav webové stránky s maximálním časovým limitem 60 sekund. Po uplynutí dané doby je vyhledán prvek pomocí CSS selektoru. Pokud je prvek v limitu nalezen provede se další část kódu. Tou je kliknutí na daný element a následné přesměrování do košíku.

9 Porovnání nástrojů

Tato kapitola je zaměřena porovnání různých testovacích nástrojů. To bude provedeno na základě klíčových kritérií, která mají vliv na efektivitu a praktickou užitečnost těchto nástrojů. Kritéria, která budou použita pro hodnocení, zahrnují pohodlí použití, výkon, flexibilitu a náklady.

Porovnání bude zaměřeno na jednotlivé aspekty každého kritéria a poskytne komplexní přehled. Toto porovnání bude založeno na poznatcích, které byly získány během práce s těmito nástroji a studiem dokumentace testovacích nástrojů.

9.1 Pohodlí použití

Cypress se vyznačuje svým jednoduchým a intuitivním API, což ho činí vhodným pro uživatele jak na začátečnické, tak na pokročilé úrovni. Jeho snadná syntaxe a přímý přístup k psaní testů snižují dobu a náklady spojené se zaškolováním nových členů týmu. (32) Významnou výhodou nástroje **Cypress** je využití programovacího jazyka JavaScript jako hlavního jazyka pro tvorbu testů, což je zejména v oblasti vývoje webových aplikací značným přínosem. Tento přístup poskytuje značné výhody pro týmy, které již mají zkušenosti s JavaScriptem nebo s ním aktivně pracují. V dnešní době existuje mnoho aplikací, které využívají JavaScript jako svůj hlavní programovací jazyk.

Použití stejného jazyka pro vývoj webových aplikací i pro testování umožňuje efektivnější využití zdrojů. To umožňuje vývojářům a testerům sdílet svůj kód a lépe se orientovat v jeho implementaci, což výrazně usnadňuje práci v týmu.

Další výhodou nástroje je možnost spouštění testů přímo v prohlížeči, čímž se eliminuje potřeba konfigurace a správy dalších externích nástrojů. Tento přístup usnadňuje rychlé ladění a debugování testů. (51)

Velkou výhodou testovacího nástroje Selenium je jeho vícejazyčnost. Na rozdíl od Cypress, který omezuje uživatele na jediný programovací jazyk, Selenium nabízí flexibilitu tím, že nepřikládá výlučnou prioritu žádnému konkrétnímu programovacímu jazyku. Tím umožňuje testerům pracovat v jejich preferovaném jazyce bez nutnosti učit se dalšímu.

9.2 Výkon

Testovací prostředí **Cypress** nabízí výhodu rychlosti díky tomu, že funguje ve stejném prostředí jako testovaná aplikace. To umožňuje synchronní spouštění testů, což vede

k výraznému zrychlení procesu testování. Princip synchronního spouštění testů zajišťuje, že každý příkaz je vykonáván ve specifickém pořadí a testy čekají na splnění akcí před pokračováním. (32) Tento přístup vede k přesnějším a spolehlivějším testům, což je zvláště užitečné při testování moderních webových aplikací s interaktivními uživatelskými rozhraními. Celkově tak tento synchronní přístup výrazně zlepšuje efektivitu testování a umožňuje dosažení vyššího výkonu v testovacím procesu oproti nástroji Selenium.

Rozšířené možnosti konfigurace v nástroji Selenium představují klíčový prvek umožňující vývojářům a testovacím týmům přizpůsobit a optimalizovat své testovací scénáře a prostředí podle specifických potřeb a požadavků projektu. Díky těmto možnostem lze nastavit různé parametry a konfigurace pro ovladače prohlížečů, což může ovlivnit způsob, jakým jsou testy spouštěny a jak jsou ovladače prohlížečů využívány. Tato funkcionalita je užitečná pro optimalizaci výkonu a stabilitu testů, zejména při použití různých prohlížečů.

Selenium provádí testy asynchronně, což umožňuje běh testů paralelně a bez nutnosti čekat na dokončení předchozího kroku před spuštěním dalšího. Tento přístup může vést ke zlepšení výkonu a rychlosti testování, zejména při provádění rozsáhlejších testovacích sestav. (31)

9.3 Flexibilita

Flexibilita nástroje Cypress se projevuje v jeho schopnosti integrovat s různými nástroji a službami, jako jsou platformy pro kontinuální integraci a doručování (Jenkins, Travis CI), správa chyb (JIRA, GitHub Issues) nebo notifikační služby (Slack). Tato možnost integrace umožňuje snadné začlenění testování do již existujícího vývojového procesu. (52)

Další výhodou, která činí z nástroje Cypress velmi flexibilní framework, je jeho podpora různých typů testů. Mezi ně patří jednotkové testy, integrační testy až po end-to-end testy. (32) Tím umožňuje pokrýt širokou škálu scénářů a přístupů k testování, což přispívá k jeho univerzálnímu využití a přizpůsobení potřebám konkrétního projektu.

Podobně jako Cypress, i Selenium nabízí širokou škálu metod pro výběr prvků na stránce, včetně CSS selektorů, selektorů XPath, ID a dalších. Tato flexibilita umožňuje snadnou manipulaci s prvky a jejich efektivní integraci do testovacích scénářů.

Integrace Selenium s různými nástroji a frameworky, jako jsou TestNG (pro Java), NUnit (pro C#), PyTest (pro Python) a další, je klíčovým faktorem při výběru vhodného testovacího prostředí pro konkrétní projekt. Tato možnost integrace umožňuje efektivní začlenění Selenium do existujícího vývojového procesu a zajištění kompatibility s preferovanými programovacími jazyky a frameworky v týmu. (31)

9.4 Náklady

Cypress i Selenium jsou nástroje poskytované pod open-source licencemi, což znamená, že jejich základní verze jsou zdarma k použití a modifikaci. Tyto verze nabízejí základní funkce pro automatizaci testů webových aplikací a jsou vhodné pro většinu běžných projektů.

S využitím pokročilých nebo prémiových funkcí a služeb však mohou vzniknout náklady. Pro efektivní využívání těchto nástrojů může být zapotřebí školení a podpora pro tým, což může zahrnovat placené kurzy, konzultace s odborníky nebo zakoupení podpory od poskytovatelů nástrojů.

Cypress Cloud je webová služba poskytující centralizované místo pro sledování, správu a analýzu testů spuštěných v Cypress. Sleduje stav testů, poskytuje detailní zprávy o chybách a analyzuje výkon testů. Lze ji integrovat s nástroji pro kontinuální integraci a nasazení (CI/CD), což umožňuje automatizované spouštění testů jako součást vývojového procesu. (53)

Cypress Recorder je rozšíření pro prohlížeč, které umožňuje snadněji nahrávat interakce s webovou aplikací a generovat testovací scénáře, což usnadňuje tvorbu testů bez psaní kódu ručně.

Selenoid je alternativní implementací Selenium Gridu navrženou pro optimalizaci pro kontejnery Docker. Tento nástroj umožňuje distribuci testovacích úloh na různé počítače pro paralelní provádění testů. Pro úspěšné nasazení a správu Selenoidu je nezbytné mít znalosti Dockeru a správy kontejnerů. (54)

Selenium IDE je nástroj pro tvorbu testovacích skriptů pomocí grafického uživatelského rozhraní (GUI). (55) Poskytuje snadnou tvorbu testů bez nutnosti psaní kódu, což může zvýšit produktivitu týmu testovacích inženýrů. Avšak pro některé pokročilé funkce a integrace může být zapotřebí placené rozšíření. Přestože Selenium IDE může být vhodný pro rychlou tvorbu a spouštění testů, je důležité zvážit, zda jsou placené funkce a integrace nezbytné pro potřeby projektu a zda jsou v souladu s jeho cíli.

9.5 Zpracování chyb

Cypress nabízí schopnost opakování akcí provedených nad prvky, což snižuje počet nestabilních testů. To znamená, že pokud dojde k chybě při interakci s prvkem, automaticky zkusí akci znovu, což může vést k úspěšnému dokončení testu.

Na druhé straně, **Selenium** je flexibilní a umožňuje širokou škálu konfigurací, což může být výhodou při řešení složitých problémů. Nicméně, Selenium může být náročnější na nastavení a může vyžadovat více úsilí pro správné zacházení s chybami. (56)

Z hlediska zpracování chyb je tedy Cypress často považován za jednodušší a efektivnější nástroj, zatímco Selenium nabízí větší flexibilitu a kontrolu. (57)

9.6 Jednoduchost ovládání

Cypress je přístupný i pro začátečníky a jeho architektura a design jsou zaměřeny na to, aby bylo snadné se naučit, jak s ním pracovat a jak ho efektivně využívat. To znamená, že učící se křivka pro **Cypress** je obecně považována za méně strmou (58)

Na druhou stranu, při pohledu na Selenium, vidíme nástroj, který je flexibilní a umožňuje širokou škálu konfigurací. To může být výhodou při řešení složitých problémů. Nicméně, Selenium může být náročnější na nastavení a řešení chyb. To znamená, že učící se křivka pro Selenium může být strmější.

Cypress je dobrou volbou pro ty kteří hledají komplexní a zároveň velmi šikovný nástroj pro testování webových aplikací. Pokud ale hledáme nástroj, který je flexibilní a více konfigurovatelný **Selenium** je lepší volbou.

9.7 Práce s asynchronním načítáním

Cypress se vyznačuje tím, že automaticky zvládá čekání. To znamená, že není třeba explicitně říkat Cypressu, aby čekal, až bude prvek interaktivní, udělá to automaticky. Tato vlastnost je jedním z důvodů, proč tým Cypress tvrdí, že jejich testy jsou údajně spolehlivější než jejich protějšky v Selenium. (59)

Tato vlastnost je velmi významná, protože asynchronní načítání prvků je běžným jevem v moderních webových aplikacích. Bez automatického zvládání čekání by testeři museli ručně řídit čekání na načtení prvků, což by mohlo vést k chybám a nestabilitě testů.

Na druhé straně spektra je **Selenium**, i když poskytuje mnoho nástrojů, pro automatizaci. Jeho testy jsou náročné na psaní. To je způsobeno tím, že Selenium se

spoléhá na rozhraní pro interakce s prohlížečem, což zavádí zpoždění v komunikaci, které může ovlivnit tempo akcí a ověřování. (60)

Toto zpoždění může být problematické při testování aplikací, které intenzivně využívají asynchronní operace, jako je načítání dat z API nebo dynamické načítání prvků. Bez správného řízení čekání může dojít k falešným selháním testů, což může vést k nesprávným závěrům o kvalitě kódu.

Při porovnání **Cypress** a **Selenium** v kontextu asynchronního načítání je tedy jasné, že oba nástroje mají své silné stránky a slabiny. Cypress se vyznačuje **automatickým** zvládáním čekání, což z něj činí ideální nástroj pro testování moderních webových aplikací s intenzivním využitím asynchronních operací.

10 Shrnutí výsledků

Tato tabulka byla vytvořena jako součást hodnocení a analýzy testovacích softwarů Cypress a Selenium. Hodnocení jednotlivých kritérií je shrnuto v následující tabulce. Každé z kritérií bylo vyhodnoceno na základě praktických testů a experimentů, které byly provedeny během procesu zkoumání. Hodnocení je na stupnici od 1 do 5, přičemž 1 je nejlepší a 5 je nejhorší. Přestože hodnocení může mít subjektivní povahu, byla vynaložena maximální snaha o zachování objektivity a spravedlnosti při srovnávání obou nástrojů.

Kritérium	***Cypress***	***Selenium***
Pohodlí použití	2	3
Výkon	3	2
Flexibilita	4	2
Náklady	3	3
Zpracování chyby	2	3
Jednoduchost ovládání	2	3
Práce s asynchronním načítáními	2	4
Celkové hodnocení	18	20

Tabulka 1 - porovnání frameworků
Zdroj: vlastní zpracování

Dle vyhodnocení se Cypress jeví jako lepší volba, i když rozdíl je malý. Celkové skóre pro Cypress je 18, pro Selenium to je 20. Důležitým prvkem, který rozdílově ovlivňuje hodnocení, je práce s asynchronním načítání, které je v dnešní době již prakticky implementováno ve všech komplexnějších aplikacích. Je důležité zdůraznit, že tyto výsledky mohou být ovlivněny specifickými potřebami daného projektu, pro který je testovací software využit.

Oba testovací nástroje mají své silné a slabé stránky, které je nutné zvážit při výběru software pro konkrétní projekt.

11 Závěry a doporučení

Práce se zaměřila na představení základních principů testování webových aplikací, ať už během vývoje nebo při následném rozvoji.

V úvodu práce byly představeny fundamentálních technologie, které jsou klíčové pro vývoj webu. Byl vysvětlen rozdíl mezi manuálním a automatickým testováním, jeho výhody a nevýhody. Následně byly představeny jednotlivé kroky testování, různé druhy testů a jejich návaznost na fázi vývoje. Během toho bylo představeno i několik aplikací, které dokážou práci testera zjednodušit a zefektivnit.

Závěrečná část práce byla se věnována porovnání 2 odlišných testovacích nástrojů, Cypress a Selenium. Toto porovnání bylo vyhodnoceno v tabulce (tabulka 1), díky které bylo možné vybrat vhodnější software, v závislosti na testovaných aplikacích. „Vítězem“ hodnocení se stal **Cypress**. Stal se tak díky své jednoduchosti a přehlednosti. Jeho schopnost asynchronního zpracování je v dnešní době moderních webových aplikací velmi významná.

Přínosem této práce je nejen lepší porozumění základních principů testování webových aplikací, ale také poskytnutí uceleného pohledu na problematiku testování v kontextu vývoje webových aplikací. Prací byl předán čtenářům komplexní přehled o různých metodách a nástrojích testování, aby byla navýšena jejich schopnost efektivněji plánovat, provádět a vyhodnocovat testovací procesy ve svých projektech.

12 Seznam použité literatury

1. Wikipedia. Webová aplikace. *Wikipedia: the free encyclopedia.* [Online] Wikimedia Foundation, 2001. [Citace: 2022-08-16. 08 2022.] https://cs.wikipedia.org/wiki/Webov%C3%A1_aplikace.

2. Adobe. Informace o webových aplikacích. *Adobe.* [Online] Adobe, 2021. [Citace: 2022-08-29. 2022 08.] https://helpx.adobe.com/cz/dreamweaver/using/web-applications.html.

3. Server. *Wikipedie: Otevřená encyklopedie.* [Online] Wikipedie: Otevřená encyklopedie. [Citace: 02. 03 2024.] https://cs.wikipedia.org/w/index.php?title=Server&oldid=23162866.

4. June 2023 Web Server Survey . *Netcraft.* [Online] Netcraft. [Citace: 20. 02 2023.] https://www.netcraft.com/blog/june-2023-web-server-survey/.

5. Janík, David. Apache vs Nginx. *Váš-Hosting.* [Online] Váš-Hosting, 30. 04 2019. [Citace: 03. 03 2024.] https://www.vas-hosting.cz/blog-apache-vs-nginx.

6. Wikipedie, Přispěvatelé. Nginx. *Wikipedie: Otevřená encyklopedie.* [Online] Wikipedie: Otevřená encyklopedie. [Citace: 02. 03 2024.] https://cs.wikipedia.org/w/index.php?title=Nginx&oldid=23038997.

7. Wikipedie. Hypertext Transfer Protocol. *Wikipedie: Otevřená encyklopedie.* [Online] 2001. [Citace: 2024-03-17. 03 2024.] WWW: https://cs.wikipedia.org/w/index.php?title=Hypertext_Transfer_Protocol&oldid=23031168.

8. Most used programming languages among developers worldwide as of 2023. *Statista.* [Online] Statista. [Citace: 10. 02 2024.] https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/.

9. Wikipedia. Hypertext Markup Language. *Wikipedia: the free encyclopedia.* [Online] Wikipedia: the free encyclopedia, 2001. [Citace: 2022-08-29. 08 2022.] https://cs.wikipedia.org/wiki/Hypertext_Markup_Language.

10. —. Kaskádové styly. *Wikipedia: the free encyclopedia.* [Online] Wikimedia Foundation, 2001. [Citace: 2022-08-29. 08 2022.] https://cs.wikipedia.org/wiki/Kask%C3%A1dov%C3%A9_styly.

11. —. JavaScript. *Wikipedia: the free encyclopedia.* [Online] Wikimedia Foundation, 2001. [Citace: 2022-08-29. 08 2022.] https://cs.wikipedia.org/wiki/JavaScript.

12. —. PHP. *Wikipedia: the free encyclopedia.* [Online] Wikimedia Foundation, 2001. [Citace: 2022-08-29. 08 2022.] https://cs.wikipedia.org/wiki/PHP.
13. Turingův test. *Wikipedie: Otevřená encyklopedie.* [Online] 2001. [Citace: 2024-03-18. 03 2024.] WWW: https://cs.wikipedia.org/w/index.php?title=Turing%C5%AFv_test&oldid=23532982.
14. Wikipedie, Přispěvatelé. Metodika vývoje softwaru. *Metodika vývoje softwaru.* [Online] Wikipedie: Otevřená encyklopedie. [Citace: 08. 02 2024.] https://cs.wikipedia.org/w/index.php?title=Metodika_v%C3%BDvoje_softwaru&oldid=21654912.
15. —. V-model. *Wikipedie: Otevřená encyklopedie.* [Online] Wikipedie: Otevřená encyklopedie. [Citace: 09. 02 2024.] https://en.wikipedia.org/w/index.php?title=V-model&oldid=1211955529.
16. Wikipedie. DevOps. *Wikipedie: Otevřená encyklopedie.* [Online] 2023. 10 16. [Citace: 01. 04 2024.] https://cs.wikipedia.org/w/index.php?title=DevOps&oldid=23275683.
17. Wikipedie, Přispěvatelé. Testování softwaru. *Wikipedie: Otevřená encyklopedie.* [Online] Wikipedie: Otevřená encyklopedie. [Citace: 10. 03 2024.] https://cs.wikipedia.org/w/index.php?title=Testov%C3%A1n%C3%AD_softwaru&oldid=23725316.
18. Koďousková, Barbora. JAK NA TESTOVÁNÍ WEBŮ A WEBOVÝCH APLIKACÍ. *Rascasone.* [Online] Rascasone, 2022. [Citace: 2022-08-18. 08 2022.] https://www.rascasone.com/cs/blog/7-tipu-jak-na-testovani-webu-internetovych-stranek-a-mobilnich-aplikaci.
19. Manuální testování – co to je, typy, postupy, přístupy, nástroje a další! *Zaptest.* [Online] Zaptest. [Citace: 10. 03 2024.] https://www.zaptest.com/cs/manualni-testovani-co-to-je-typy-postupy-pristupy-nastroje-a-dalsi.
20. Automatizované testování. *Testování software.* [Online] Testování software. [Citace: 11. 03 2024.] http://testovanisoftwaru.cz/automatizovane-testovani/.
21. Wikipedie, Přispěvatelé. Automatizace testování. *Wikipedie: Otevřená encyklopedie.* [Online] Wikipedie: Otevřená encyklopedie. [Citace: 12. 03 2024.] https://cs.wikipedia.org/w/index.php?title=Automatizace_testov%C3%A1n%C3%AD&oldid=22954634.
22. Vliv kopilotů a generativní umělé inteligence na RPA a testování softwaru: současnost a budoucnost. *Zaptest.com.* [Online] 2024. [Citace: 2022-08-18. 08 2022.]

https://www.zaptest.com/cs/vliv-kopilotu-a-generativni-umele-inteligence-na-rpa-a-tes.
23. Smoke testy. *Testování softwaru.* [Online] 2011. [Citace: 2022-08-29. 08 2022.] http://testovanisoftwaru.cz/automatizovane-testovani/.
24. Progresní a regresní testy. *Testování softwaru.* [Online] 2011. [Citace: 2022-08-29. 08 2022.] http://testovanisoftwaru.cz/metodika-testovani/druhy-typy-a-kategorie-testu/progresni-a-regresni-testy/.
25. Druhy testování v procesu vývoje SW. *SWTestovani.cz.* [Online] 2024. [Citace: 2024-03-18. 03 2024.] http://test.swtestovani.cz/index.php?option=com_content&view=article&id=18:druhy-testovani-v-procesu-vyvoje-sw&catid=3:zaklady&Itemid=11.
26. Funkční a nefunkční testy. *Testování softwaru.* [Online] 2011. [Citace: 2022-08-29. 08 2022.] http://testovanisoftwaru.cz/metodika-testovani/druhy-typy-a-kategorie-testu/funkcni-a-nefunkcni-testy/.
27. Wikipedie. Integrační testování . *Wikipedie: Otevřená encyklopedie.* [Online] 2023. [Citace: 2024-03-18. 03 2024.] https://cs.wikipedia.org/w/index.php?title=Integra%C4%8Dn%C3%AD_testov%C3%A1n%C3%AD&oldid=23135583.
28. Testování webových aplikací – hluboký ponor do testování webových aplikací, typy, procesy, automatizace, nástroje a další! *Zeptest.* [Online] Zaptest. [Citace: 10. 03 2024.] https://www.zaptest.com/cs/testovani-webovych-aplikaci-hluboky-ponor-do-testovani-webovych-aplikaci-typy-procesy-automatizace-nastroje-a-dalsi.
29. Selenium Overview. *Selenium.* [Online] Selenium. [Citace: 12. 03 2024.] https://www.selenium.dev/documentation/overview/.
30. Getting started with WebDriver. *Selenium.dev.* [Online] https://www.selenium.dev/documentation/webdriver/getting_started/. [Citace: 2024-03-18. 03 2024.]
31. Selenium Testing. *BrowserStack.* [Online] BrowserStack. [Citace: 01. 04 2024.] https://www.browserstack.com/selenium.
32. Why Cypress? *Cypress Docs.* [Online] Cypress. [Citace: 29. 03 2024.] https://docs.cypress.io/guides/overview/why-cypress.
33. mdn web docs. *Mozilla.* [Online] Mozilla. [Citace: 14. 03 2024.] https://developer.mozilla.org/en-US/docs/Web/CSS/:has#browser_compatibility.

34. Přístupnost webu. *Shaping Europe's digital future.* [Online] [Citace: 16. 03 2024.] https://digital-strategy.ec.europa.eu/cs/policies/web-accessibility.
35. Wikipedie, Přispěvatelé. Optimalizace pro vyhledávače. *Wikipedie: Otevřená encyklopedie.* [Online] Wikipedie: Otevřená encyklopedie. [Citace: 16. 03 2024.] https://cs.wikipedia.org/w/index.php?title=Optimalizace_pro_vyhled%C3%A1va%C4%8De&oldid=23533991.
36. Frog, Screaming. Screaming Frog Guide For The SEO Spider. *Screaming Frog SEO Spider Website Crawler.* [Online] [Citace: 08. 03 2024.] https://www.screamingfrog.co.uk/seo-spider/user-guide/.
37. Co je testování výkonu: hluboký ponoř do typů, postupů, nástrojů, výzev a dalších věcí. *Zaptest.com.* [Online] 2024. [Citace: 2024-03-18. 03 2024.] https://www.zaptest.com/cs/co-je-testovani-vykonu-hluboky-ponor-do-typu-postupu-nastroju-vyzev-a-dalsich.
38. Load Testing Guide. *TestSigma.* [Online] 2024. [Citace: 2024-03-18. 03 2024.] https://testsigma.com/guides/load-testing/.
39. contributors, Wikipedia. Stress testing. *Wikipedia, The Free Encyclopedia.* [Online] Wikipedia, The Free Encyclopedia. [Citace: 01. 03 2024.] https://en.wikipedia.org/w/index.php?title=Stress_testing&oldid=1177708952.
40. Testování škálovatelnosti. *Wikipedie: Svobodná encyklopedie.* [Online] 2023. [Citace: 2024-03-18. 03 2024.] https://cs.wikipedia.org/w/index.php?title=Integra%C4%8Dn%C3%AD_testov%C3%A1n%C3%AD&oldid=23135583.
41. contributors, Wikipedia. Software performance testing. *Wikipedia.* [Online] Wikipedia, The Free Encyclopedia. [Citace: 18. 03 2024.] https://en.wikipedia.org/w/index.php?title=Software_performance_testing&oldid=1202649549 .
42. Testování výdrže. *ArtOfTesting.* [Online] [Citace: 2024-03-18. 03 2024.] https://artoftesting.com/endurance-testing.
43. Google. About PageSpeed Insights. *Developers Google.* [Online] Google. [Citace: 13. 03 2024.] https://developers.google.com/speed/docs/insights/v5/about.
44. Hamilton, Thomas. API Testing Tutorial: What is API Test Automation? *GURU 99.* [Online] [Citace: 12. 03 2024.] https://www.guru99.com/api-testing.html.

45. 6 důvodů, proč si vybrat Postman pro testování API. *Tesena.* [Online] [Citace:] https://www.tesena.com/novinky/6-duvodu-proc-si-pro-testovani-api-vybrat-postman.
46. Wikipedie. Selenium. *Wikipedie: Otevřená encyklopedie.* [Online] c2022. [Citace: 2024-03-18. 03 2024.] https://cs.wikipedia.org/w/index.php?title=Selenium_(software)&oldid=21364773.
47. Oracle. JDK Installation Guide. *Oracle.* [Online] [Citace: 20. 03 2024.] https://docs.oracle.com/en/java/javase/22/install/overview-jdk-installation.html.
48. Chromedriver. *Chromedriver.* [Online] [Citace: 24. 03 2023.] https://chromedriver.chromium.org/.
49. npm Docs. *npmjs.* [Online] [Citace: 20. 03 2024.] https://docs.npmjs.com/.
50. Cypress. Your First Test with Cypress. *Cypress.* [Online] Cypress. [Citace: 20. 03 2024.] https://docs.cypress.io/guides/end-to-end-testing/writing-your-first-end-to-end-test.
51. Key Differences. *Cypres Docs.* [Online] Cypress. [Citace: 28. 03 2024.] https://docs.cypress.io/guides/overview/key-differences.
52. Introduction. *Cypress Documentation.* [Online] Cypress. [Citace: 29. 03 2024.] https://docs.cypress.io/guides/continuous-integration/introduction.
53. Cypress Cloud. *Cypress docs.* [Online] Cypress. [Citace: 20. 03 2024.] https://docs.cypress.io/guides/cloud/introduction.
54. Andryashin, Alexander. Selenoid. *Selenoid.* [Online] [Citace: 24. 03 2024.] https://aerokube.com/selenoid/latest/.
55. Getting started. *Selenium IDE.* [Online] Selenium. [Citace: 30. 03 2024.] https://www.selenium.dev/selenium-ide/docs/en/introduction/getting-started.
56. Cypress vs. Selenium. *Testim.* [Online] 2022. [Citace: 2024-03-18. 03 2024.] https://www.testim.io/blog/cypress-vs-selenium/.
57. Selenium vs. Cypress: Selecting the Right Test Automation Tool. Medium. *Craven, R.* [Online] 2023. [Citace: 2024-03-18. 03 2024.] https://medium.com/testopia/selenium-vs-cypress-selecting-the-right-test-automation-tool-940c58e8f322.
58. Oloja, T. Selenium vs. Cypress: A Comprehensive Comparison. *Medium.* [Online] 2023. [Citace: 2024-03-18. 03 2024.] https://medium.com/@timilehinoloja10/selenium-vs-cypress-a-comprehensive-comparison-3d9242ed2fb3.

59. Cruz, Marco A. Cypress vs Selenium: Collision of Champions . *automateNow.* [Online] 19. 12 2023. [Citace: 20. 03 2024.] https://automatenow.io/cypress-vs-selenium.
60. How to Avoid Flaky Tests in Selenium. *Semaphore.* [Online] 27. 03 2024. [Citace: 28. 03 2024.] https://semaphoreci.com/blog/flaky-tests-selenium.
61. Automatizované testování. *Testování softwaru.* [Online] 2011. [Citace: 2022-08-29. 08 2022.] http://testovanisoftwaru.cz/automatizovane-testovani/.

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání bakalářské práce

Autor:	**Jan Theodor**
Studium:	I2000427
Studijní program:	B1802 Aplikovaná informatika
Studijní obor:	Aplikovaná informatika
Název bakalářské práce:	**Principy testování webových aplikací**
Název bakalářské práce AJ:	Principles of web application testing

Cíl, metody, literatura, předpoklady:

Cíl: Cílem práce je seznámit čtenáře s principy vývoje a testování webových aplikací a porovnat jednotlivé technologie.

Osnova:

1. Úvod
2. Webové aplikace
3. Principy a nástroje testování
4. Výsledky
5. Závěr

Zadávající pracoviště:	Katedra informatiky a kvantitativních metod, Fakulta informatiky a managementu
Vedoucí práce:	doc. Mgr. Tomáš Kozel, Ph.D.
Datum zadání závěrečné práce:	10.10.2023