Automatizované testování software s důrazem na možné použití v rámci CI/CD pipeline

Martin Chuděj

Bakalářská práce
2024

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Martin Chuděj**
Osobní číslo:	**A21114**
Studijní program:	**B0613A140020 Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Automatizované testování software s důrazem na možné použití v rámci CI/CD pipeline**
Téma práce anglicky:	**Automated Software Testing with an Emphasis on Possible Use within the CI/CD Pipeline**

Zásady pro vypracování

1. Proveďte literární rešerši na téma automatizované testování software.
2. Navrhněte systém testování SW s důrazem na možné použití v rámci CI/CD pipeline.
3. Analyzujte vybrané nástroje pro testování SW.
4. Realizujte testování SW s důrazem na použití CI/CD pipeline na konkrétním projektu.
5. Vyhodnoťte použití vybraného nástroje v rámci CI/CD pipeline u konkrétního projektu.

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. ANASTATOV, Marko. *CI/CD Pipeline: A Gentle Introduction*. Online. Semaphoreci. 2022. Dostupné z: https://semaphoreci.com/blog/cicd-pipeline. [cit. 2023-11-08].
2. FOSCO, Molly. *What is a CI/CD pipeline?* Online. FOSCO, Molly. CIRCLECI. CircleCi. 2022. Dostupné z: https://circleci.com/blog/what-is-a-ci-cd-pipeline/. [cit. 2023-11-08].
3. HAMILTON, Thomas. *CI/CD Pipeline: Learn with Example*. Online. Guru99. 2023. Dostupné z: https://www.guru99.com/ci-cd-pipeline.html. [cit. 2023-11-08].
4. SOMMERVILLE, Ian, 2015. *Software Engineering*. Online. Tenth Edition. Pearson Education Limited. ISBN 9781292096131. Dostupné také z: https://iansommerville.com/software-engineering-book/.

Vedoucí bakalářské práce: **prof. Mgr. Roman Jašek, Ph.D., DBA**
Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce: **5. listopadu 2023**
Termín odevzdání bakalářské práce: **13. května 2024**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN
L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 5. ledna 2024

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.
- že při tvorbě této práce jsem použil nástroj generativního modelu AI DeepAI; https://deepai.org/chat za účelem zjednodušení složitého textu. Po použití tohoto nástroje jsem provedl kontrolu obsahu a přebírám za něj plnou zodpovědnost.

Ve Zlíně, dne

Martin Chuděj, v. r.
podpis studenta

ABSTRAKT

Bakalářská práce je zaměřená na testování softwaru. Teoretická část přináší přehled o testování softwaru a principů CI/CD pipeline. Jsou zde také vypsané jedny z nejpopulárnějších nástrojů pro automatizované testování softwaru s porovnáním mezi nimi. V praktické části je realizováno testování webové aplikace, které je automatizováno pomocí Selenium v jazyce Python a integrováno do CI/CD pipeline.

Klíčová slova:

Testování, software, aplikace, CI/CD pipeline, testy

ABSTRACT

The bachelor thesis focuses on software testing. The theoretical part provides an overview of software testing and principles of CI/CD pipeline. It also lists some of the most popular tools for automated software testing with a comparison between them. In the practical part,is implemented the testing of the web application, which is automated using Selenium in Python and integrated into the CI/CD pipeline

Keywords:

Testing, software, application, CI/CD pipeline, tests

Chtěl bych poděkoval vedoucímu mé bakalářské práce,

prof. Mgr. Roman Jašek, Ph.D., DBA

za jeho cenný čas a nezbytné rady.

Dále bych chtěl vyjádřit vděk společnosti, u které jsem práci realizoval,

PRINCIPIA SOLUTIONS s.r.o

včetně jejího jednatele,

Ing. Martina Burdíka

za umožnění realizace práce v reálném prostředí, díky čemuž jsem získal cenné praktické zkušenosti a pochopil význam tématu mé práce v praxi.

Prohlašuji, že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

V dnešní digitální éře, kde je internet a webové aplikace nepostradatelnou součástí každodenního života, je důležité zajistit, aby tyto aplikace byly spolehlivé, funkční a uživatelsky přívětivé. S narůstající složitostí a rozmanitostí webových aplikací se zvyšuje i potřeba automatizace testování, aby se zajistilo, že aplikace splňují stanovené požadavky a standarty.

Během zpracování této bakalářské práce jsem často použil anglická slova, která buď nemají český ekvivalent nebo se daleko častěji vyskytují anglicky.

V úvodní části bakalářské práce se zaměříme na testování softwaru jako klíčový prvek vývoje softwaru, který zajišťuje jeho kvalitu a spokojenost uživatelů. Budeme se stručně zabývat historií testování softwaru, vysvětlíme si, co to vlastně je manuální testování a automatizované testování. Jaké strategie pro testování existují nebo také typy testů.

V následující části práce se budeme věnovat architektuře CI/CD pipeline, která slouží k automatizaci procesů vývoje a nasazování softwaru. Projdeme stručně historii CI/CD, vysvětlíme si, co znamenají jednotlivé komponenty této pipeline. Dále se zaměříme na bezpečnost CI/CD, identifikaci bezpečnostních hrozeb a jak těmto hrozbám předejít. Podíváme se tedy i na bezpečnostní mechanismy pro ochranu softwaru.

Jako poslední kapitolu teoretické části práce si porovnáme nejčastěji používané nástroje pro automatické testování softwaru. Pro každý nástroj si shrneme jeho vlastnosti, výhody a nevýhody. Ke konci této kapitoly provedeme porovnání z hlediska aplikace použití, zda je nutné psaní skriptů, které jazyky podporují, jestli jsou zdarma nebo něco stojí a zda mají nějakou analýzu výsledků.

V praktické části se zaměříme na návrh a realizaci systému testování softwaru na konkrétním projektu registru digitalizace, který provozuje Národní knihovna. Při práci na tomto projektu jsem spolupracoval s firmou Principia. Implementace testovacího systému se opírá o konfigurační soubor ve formátu YAML (.yml). Kód pro testovací scénáře je napsaný v Pythonu s využitím Selenium frameworku.

Další důležitý proces je integrace do CI/CD. Díky tomuto automatizovanému spouštění testů v rámci CI/CD prostředí je zajištěna pravidelná kontrola kvality softwaru a minimalizace rizika při nasazování nových verzí aplikace.

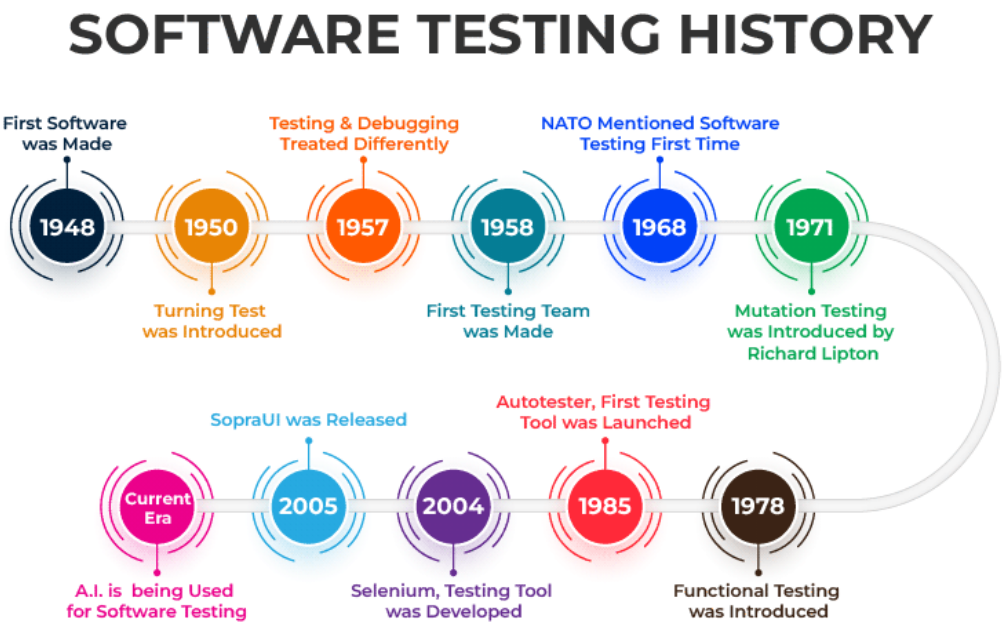
I. TEORETICKÁ ČÁST

1 LITERÁRNÍ REŠERŠE TESTOVÁNÍ SOFTWARU

1.1 Úvod do testovaní softwaru

Testování softwaru je důležitou součástí vývoje softwaru, jelikož zajišťuje kvalitu produktu a spokojenost uživatelů. V tomto úvodu se zaměříme na různé věci okolo testování softwaru, včetně historie nebo také důvodů pro zavedení do vývoje softwaru.

1.1.1 Historie a vývoj testování



Obrázek 1. Historie testování softwaru[1]

Druhá světová válka byla doba, kdy se ve velkém začaly zkoumat počítače a jejich využití. Okolo roku 1945 se některé termíny oficiálně zapsaly do historie testování. [1]

Jedním z těchto termínu je *Debugging*[2], který v této době byl hlavní metodou testování softwaru. Ovšem v roce 1957 Charles L. Baker odlišil *debbuging* od testování. Jeho myšlenkou bylo otestovat aplikaci pomocí vytvořeného scénáře, díky kterému najde chyby. Tato éra testování se nazývá *Demonstration-oriented era*[3] [2]

---

[1] *Software Testing History*. Online. In: GeeksforGeeks. 2022. Dostupné z: https://www.geeksforgeeks.org/history-of-software-testing/. [cit. 2024-05-03].
[2] Debugging = Ladění aplikace
[3] Demonstration-oriented era = Éra zaměřená na demonstraci chyby

Další věk testování nastal v roce 1979, kdy vnikl termín *breakage*[4] testování. Jedná se o testování, které útočí na kód takovým způsobem, že ho chce zničit. Jméno tohoto věku je *Destruction-oriented era*[5] [2]

Každým rokem je software více a více modernější a v 80. letech vznikly první pomůcky pro automatizované testy. Testeři, kteří do této doby dělali ty samé věci dokola mohli tyto kroky dělat automatizovaně. [2]

1.1.2 Definice manuálního testování

Manuální testování je takové testování, které nevyužívá žádné pomoci automatizovaných nástrojů a testovací případ je tedy celý proveden ručně testerem. Zabývá se tím, jestli všechny testovací případy probíhají tak, jak je uvedeno v příručce nebo dokumentaci. Testy jsou vytvářeny takovým způsobem, aby pokryly většinu funkcí testované aplikace. Hlášení výsledků testů je také vytvářeno ručně testerem. [3]

Manuální testy umožňují detailnější porozumění na problémy, které nastanou při testování aplikace. Jedním z těchto důvodů je lidský faktor, tudíž jde aplikace posoudit z pohledu uživatele. [4]

1.1.2.1 Strategie testování

Společnosti používají tři základní strategie testování. Každá z těchto strategií má vlastní využití a liší se úrovní přístupu testera.

1.1.2.1.1 *White-box*[6] testování

Také známe jako testování průhledné skříňky. Při testování touto metodou mohou testeři volně nahlédnout do všech zdrojových kódů. Testuje tedy vnitřní kódování a infrastrukturu. Zaměřuje se hlavně na kontrolu předem definovaných vstupů oproti očekávaným a požadovaným výstupům. Tato metoda testování je nejlepší pro ranné fáze vývoje softwaru, kde vývojáři mohou porovnat svůj kód s testovacími případy a snadno najít oblast, ve které se vyskytují problémy. Název tohoto typu testování se odvíjí od toho, co vlastně testuje, tudíž

[4] Breakage = ničení
[5] Destruction-oriented era = Éra zaměřená na zničení kódu
[6] White-box = Bílá skříňka

white box, *clear box* nebo *transparent box* podle toho, že jde nahlédnout za vnější slupku do jeho vnitřního běhu. [3]

1.1.2.1.2 *Black-box*[7] testování

Jedná se o techniku softwaru, při které tester zkoumá funkčnost systému, aniž by znal jeho interní kód nebo strukturu. Testeři poskytují vstupy do systému a pozorují výstupy se zaměřením na očekávané a neočekáváné uživatelské akce, ale také výkon tohoto systému, jeho spolehlivost i použitelnost. *Black-box* testování nevyžaduje žádné technické znalosti a může být prováděno externími testery, protože tento typ testování hodnotí systém od začátku do konce a simuluje uživatelské chování. [5]

1.1.2.1.3 *Gray-box*[8] testování

Tato metoda testování softwaru kombinuje prvky *white-box* testování a *black-box* testování. Testeři mají pouze částečně znalosti o zdrojovém kódu, takže narozdíl od *white-box* testování testeři nevidí zdrojový kód celý, ale jelikož vidí nějakou část kódu tak se nejedná ani o *black-box* testování. Cílem této metody je hledat chyby způsobené nevhodnou strukturou kódu nebo nevhodným použitím softwaru. [6]

1.1.3 Definice automatizovaného testování

Automatizované testování je takové testování, kdy vývojáři nebo testeři programují testovací skripty. Tyto skripty vytvářejí pomocí testovacích nástrojů a *frameworků*[9]. Vytvořené skripty se poté automaticky spouštějí na softwaru bez lidského zásahu. Výsledky automatizovaného testování, ať už obsahuje software chyby nebo ne, jsou zobrazeny automaticky. První procesy vytváření automatizovaných testu jsou náročné, protože je nutné ručně vytvořit testovací skript, poté už je porovnávání skutečných výsledků s očekávanými výsledky prováděno automaticky. Automatizované testy umožňují rychlé a opakované spouštění testů, což vede k efektivní identifikaci problémů a jsou vhodné pro opakované testovací scénáře. [7]

---

[7] Black-box = Černá skříňka
[8] Gray-box = Šedá skříňka
[9] Framework = Kostra, soustava, struktura

1.1.4 Důvody pro zavedení do vývoje softwaru

Zavést testování do vývoje softwaru je klíčové, protože pomáhá vytvořit kvalitnější produkt, šetří čas a náklady, zvyšuje spokojenost zákazníků a zlepšuje vývojový proces.

1.1.4.1 Pro manuální testování

V dnešní době jsou testovací softwary na velmi vyspělé úrovni, ale může se stát, že automatizované testování neodhalí v aplikaci problém, protože ho prostě a jednoduše nevidí. V tomto případě je lidský faktor důležitou součástí testování. Tester může také poskytnout zpětnou vazbu ohledně testované aplikace z pohledu uživatele, zda se v prostředí dobře vyzná nebo jestli je něco nepřirozené. [8]

1.1.4.2 Pro automatizované testování

Zavedení automatizovaného testování do vývoje softwaru přináší efektivitu, časovou úsporu a lepší pokrytí funkcí a scénářů. Automatizace zajišťuje konzistenci v testování, umožňuje lepší detekci chyb a snadnou rozšiřitelnost testovacího pokrytí. Celkově zvyšuje kvalitu softwaru a přispívá k větší spokojenosti uživatelů.

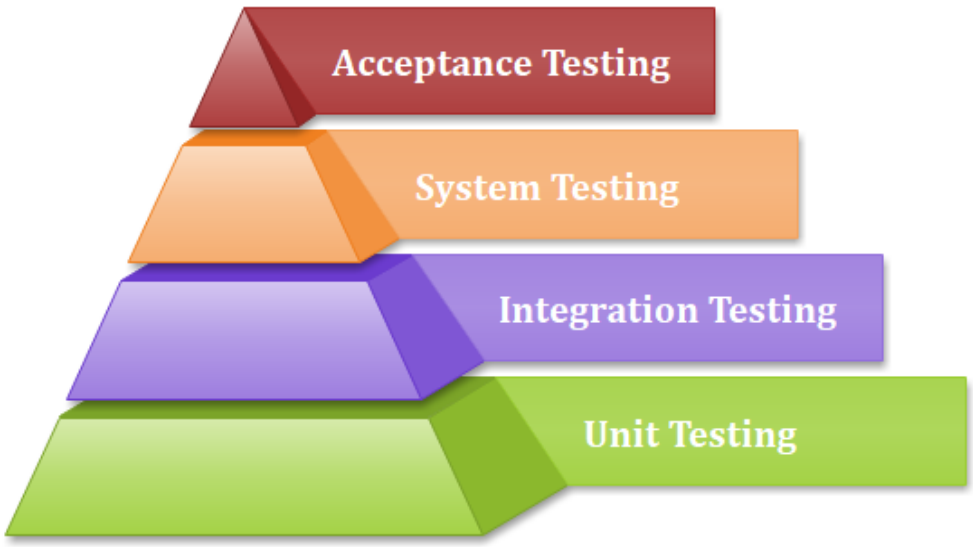
1.2 Rozdíly mezi manuálním a automatizovaným testováním

Rozdíly mezi manuálním a automatizovaným testováním jsou klíčové pro pochopení, jak se tyto dvě metody od sebe odlišují. V tabulce můžeme vidět, že automatizované testování je obvykle rychlejší a opakovatelnější, ale může být náročnější na údržbu nebo na vytvoření. Naopak manuální testování je jednodušší na začátku, ale může být pomalejší a méně spolehlivé dlouhodobě. Každá metoda má své výhody a nevýhody, které je důležité zvážit při rozhodování, jak testovat software.

Tabulka 1. Tabulka rozdílů mezi typy testování (Vlastní)

Kritérium	Manuální testování	Automatizované testování
Tester	Lidský tester	Skripty nebo automatizační nástroje
Rychlost	Závisí na rychlosti testera	Rychlé provedení skriptů
Opakovatelnost testů	Závisí na konzistenci testera	Opakované a konzistentní testy
Náklady na údržbu	Nižší náklady na údržbu testů	Vyšší náklady na údržbu skriptů
Flexibilita a interaktivita	Vyšší	Omezená
Vhodnost pro regresní testování	Nevhodné	Vhodné
Chybovost	Vyšší	Nižší
Náklady na počáteční implementaci	Nižší náklady na implementaci	Vyšší náklady na implementaci
Investice	Nízká návratovost investice	Vysoká návratovost investice

1.3 Typy testů automatizovaného testování



Obrázek 2. Typy testů[10]

[10] *Testing*. Online. In: Medium. 2019. Dostupné z: https://medium.com/@jonathansychan/some-code-testing-concepts-85fbc8fc27e2. [cit. 2024-05-03].

1.3.1 Unit testy

Tento typ testů se zaměřuje na detailní kontrolu jednotlivých komponentů aplikace. Vývojáři tyto testy píšou během vývoje, aby zajistili spolehlivost a funkčnost každé části softwaru. V unit testech jsou jednotkou testování různé entity, jako jsou funkce, procedury, metody nebo objekty. Jejich cílem je ověřit, že každá část softwaru pracuje správně a splňuje stanovené požadavky. Díky tomu se snižuje pravděpodobnost chyb při uvedení do provozu. Postup unit testování zahrnuje plánování, psaní testovacích scénářů, provádění těchto scénářů pomocí nějakého testovacího *frameworku* a analýza výsledků. [9]

Výhody této metody testování:

- Brzké odhalení problémů
- Snížené náklady na opravu
- Podporuje *test-driven development*[11]
- Usnadňuje refaktorování kódu a detekci změn

[9]

Nevýhody této metody testování:

- Psaní testovacích scénářů je časově náročné
- Nepokryje všechny možné chyby v softwaru
- Potřebuje větší údržbu při časté změně zdrojového kódu

[10]

1.3.2 Integrační testy

Tento typ testování je klíčový pro vývoj softwaru. Různé jednotky, moduly nebo komponenty softwaru se testují jako jeden. Jeho hlavním cílem je ověřit, že tyto komponenty správně spolupracují a pracují mezi sebou, což je zásadní pro zajištění kvality a spolehlivosti výsledné aplikace. Tento proces může být také označován jako *string testing*[12] nebo *thread testing*[13]. Proces testování spočívá v tom, že tyto různé komponenty se integrují a následně

[11] Test-driven development = Vývoj řízený testy. Testy jsou napsány před vlastním kódem.
[12] String testing = Testování řetězců
[13] Thread testing = Testování vláken

se testuje jejich chování jako jednotného celku, ověřuje se tedy jestli jednotlivé moduly komunikují správně i po jejich integraci. [11]

Integrační testování obvykle probíhá stejně jako vývoj softwaru, což někdy může být problém, protože všechny moduly nemusí být dostupné. Hlavní rozdíl mezi integračním a unit testováním je úroveň detailu, kterou testujeme. Unit testování se zaměřuje na testování jednotlivých modulů, zatímco integrační testování se soustředí na ověřování, jak spolu tyto moduly komunikují. [11]

Výhody integračního testování:

- Jsou vhodné do menších systémů
- Jednoduchý a přímočarý přístup
- Testovací scénáře lze rychle dokončit

[11]

Nevýhody integračního testování :

- Časově náročné, protože se čeká na integraci všech modulů
- Není vhodné pro velké projekty
- Kritické moduly nejsou izolovány a prioritně testovány, protože se testují jako celek
- Vysoké riziko těžko identifikovatelných chyb.

[12]

1.3.3 Systémové testy

Tento typ testování je zaměřen na zkoumání interakcí mezi různými komponentami aplikace v plně integrovaném systému nebo aplikaci jako celku. Cílem je ověřit, zda aplikace provádí to co má dělat podle stanovených požadavků a očekávání. Využívá metodu *black-box* testování, což znamená, že se nezabývá vnitřním kódem nebo mechanismem, ale spíše jeho vnější funkčností. Tento typ testování může být také označování jako *system-level testing*[14] nebo *system integration testing*[15]. Systémové testování se provádí po integračním testováním a před akceptačním testováním. [13]

---

[14] System-level testing = Testování na úrovni systému
[15] System integration testing = Testování systémové integrace

Výhody systémového testování:

- Ověřuje celkovou funkčnost systému
- Brzké odhalení problémů na úrovni systému
- Zajištuje, aby software splňoval potřeby uživatele
- Zvyšuje spolehlivost a kvalitu softwaru

[13]

Nevýhody systémového testování:

- Časově náročnější než jiné typy testování, jelikož testuje celý software
- Dražší než jiné typy testování, jelikož testuje celý software
- Náročný na implementaci u velkého softwaru
- Jedná se o *black-box* testování, takže limitovaný náhled do kódu

[14]

1.3.4 Akceptační testy

Hlavní cíl tohoto typu testování je zajištění toho, že software splňuje obchodní požadavky a potřeby uživatelů. Tato fáze je klíčování pro minimalizaci rizika nespokojenosti uživatelů a vytlačení jakýchkoliv drahých odvolání. Průzkum provedený společností IBM ukazuje, že akceptační testování může odhalit až 90 % chyb v softwaru. Pokud se najdou chyby po uvedení do provozu, tak náklady na jejich opravu mohou být až stokrát vyšší než během akceptačního testování. [15]

Během akceptačního testování jsou zapojeni samotní uživatelé nebo představitelé skupiny uživatelů, což zajišťuje, že systém je testování z perspektivy lidí, kteří s ním budou denně pracovat. To pomáhá zajistit, že systém splňuje skutečné potřeby a očekávání uživatelů.

Existuje několik typů akceptačního testování, včetně testování uživatelského přijetí, alfa testování, beta testování, operačního testování akceptace a mnoho dalších. Každý typ testování má svůj specifický účel a přínos pro ověření různým aspektů softwaru [15]

Výhody akceptačního testování:

- Přímá zpětná vazba od uživatelů
- Automatizované provádění testů
- Vylepšení uživatelské spokojenosti

Nevýhody akceptačního testování:

- Požadovaná základní znalost softwaru
- Časově náročné pro uživatele
- Zpětná vazba a analýza výsledků od spousty uživatelů může být náročná
- Vývojářský tým se aktivně nezapojuje, což může vést k nedorozuměním

[15]

2 ARCHITEKTURA CI/CD PIPELINE

2.1 Úvod do CI/CD pipeline

2.1.1 Historie

CI/CD nebo jinými slovy *Continuous Integration*[16] a *Continuous Delivery*[17] (nebo také *continous deployment*[18]), zásadně změnilo způsob vývoje a nasazování softwaru. Vniklo už v 90. letech, ale do širšího povědomí se dostalo až na začátku 2010. let. [16]

První generace nástrojů pro CI přišla se softwarem zvaném CruiseControl v roce 2001 a později Hudson, který byl po nějakém čase přejmenován na Jenkins. Tyto nástroje vyžadovaly nasazení na fyzické servery, a hlavně integrovaly s SVN [16], což je systém správy verzí, který se používá k udržování a sledování změn v souborech a adresářích. [17]

Druhá generace CI/CD přišla s rozvojem cloudu okolo roku 2010, kdy se služby jako TravisCI a CircleCI staly populárními díky své integraci s GitHubem a snadné konfiguraci pomocí YAML souborů. Poté velcí poskytovatelé cloudu jako Amazon, Google a Microsoft následně přestavili své vlastní služby pro CI/CD. Jedná se například o CodeBuild, Cloud Build nebo Azure DevOps. [16]

Třetí generace CI/CD přinesla integraci s hostingovými službami jako GitLab a GitHub. Gitlab v roce 2015 integroval CI/CD přímo do svého hostingového produktu, následovaný GitHubem s vydáním Actions v roce 2018. Tato integrace zjednodušila pracovní postupy vývojářů a vedla k vytvoření ekosystémů opětovně použitelných komponent. To způsobilo to, že druhá generace CI/CD nebo také CI-jako-služba v roce 2023 efektivně zastarala. [16]

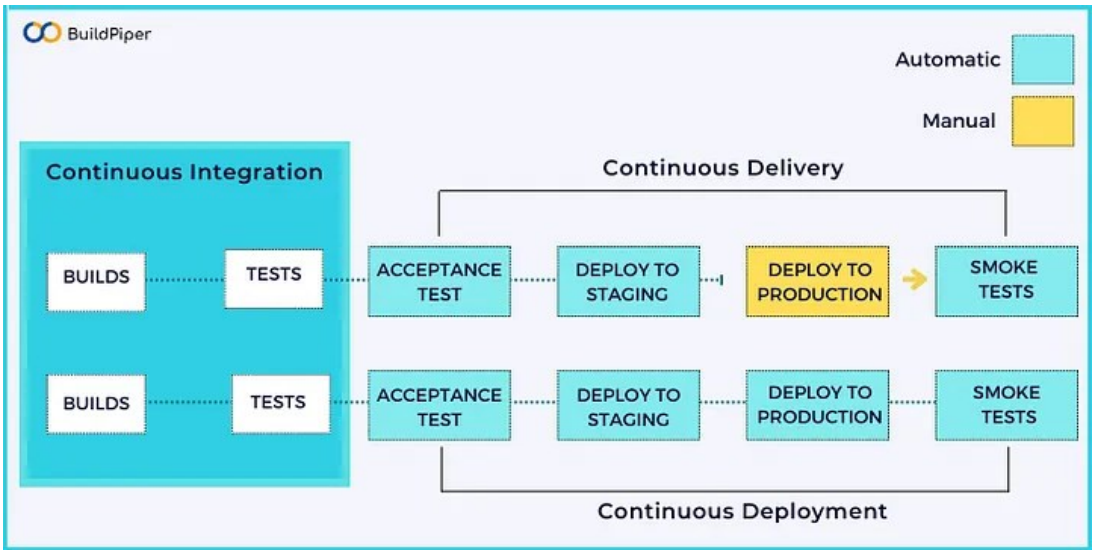
2.1.2 Definice

CI/CD pipeline je automatizovaný proces, který spojuje praktiky kontinuální (průběžné) integrace a kontinuálního (průběžného) nasazování. Tento proces automatizuje kroky potřebné k přenesení nového kódu z vývoje do produkce. To zahrnuje sestavení kódu, jeho testování a nasazení do prostředí, kde je dostupný uživatelům. CI/CD pipeline umožňuje vývojářským

[16] Continuous Integration = Průběžná integrace
[17] Continuous Delivery = Průběžné doručování
[18] Continuous Deployment = Průběžné nasazování

týmům provádět změny v kódu, které jsou následně automaticky testovány a nasazeny, což minimalizuje výpadky a urychluje vydávání nových verzí softwaru. [18]

2.2 Základní komponenty



Obrázek 3. Rozdíly mezi komponenty CI/CD[19]

2.2.1 Continuous Integration (CI):

Continuous integration je automatizovaný proces začleňování změn v kódu změněného vývojáři jednoho softwarového projektu. Tyto změny kódu jsou sloučeny do hlavní větve až několikrát denně, což způsobí spuštění automatizované sekvence sestavení a testování kódu. Pokud nějaký test selže, tak je kód zablokován a neprochází do dalších fází. Je tedy odeslán zpět vývojářům k opravě. [19]

2.2.2 Continuous Delivery (CD)

Continuous delivery rozšiřuje už již známé principy *continuous integration*. Jakmile všechny změny v kódu úspěšně projdou testovací fází v CI, jsou nasazeny do testovacího prostředí. Kromě automatizovaných testů zahrnuje CD také automatizovaný proces vydání nebo anglický, více používaný výraz, *release*[20]. Rozdíl mezi *continuous delivery* a *continuous*

[19] PRABHAKAR, Komal J. *Elements Of CI/CD Pipeline*. Online. In: Medium. 2022. Dostupné z: https://medium.com/buildpiper/elements-of-ci-cd-pipeline-2a4f31161075. [cit. 2024-05-03].

[20] Release = Vydání uživatelům

deployment je ten, že u *continuous delivery* se vydává verze přes kliknutí na tlačítko, což znamená, že je potřeba lidský zásah pro vydání nové verze. [19]

2.2.3 Continuous Deployment (CD)

Continuous deployment představuje nejvyšší úroveň automatizace v softwarovém vývoji. Pro tento přístup není po úspěšném absolvování všech fází vyžadován lidský zásah. Každá změna kódu, která projde fázemi CI a CD, je automaticky doručena uživatelům bez jakéhokoliv zpoždění. Díky tomu, že je automatizovaný *release*, je zrychlena zpětná vazba uživatelů, jelikož nemusí čekat na den vydání. [19]

2.3 Základní prvky pro úspěšnou implementaci CI/CD

V CI/CD existuje několik klíčových faktorů, které zajišťují optimální průběh vývojového cyklu a nasazení softwaru. Tyto prvky pokrývají různé fáze procesu a jsou zásadní pro úspěšnou implementaci CI/CD. [18]

2.3.1 Jednotný repositář

Repositář, který bude obsahovat všechny potřebné soubory pro sestavení. Mezi tyto soubory se zahrnuje zdrojový kód, knihovny, struktura databáze, konfigurační soubory a nějaká správa verzí. Může také obsahovat testovací skripty a skripty pro sestavení aplikace. [20]

2.3.2 Časté kontroly hlavní větve

Časté integrování kódu do hlavní větve je klíčový pro kontrolu kódu. Doporučuje se používání malých úseků kódu a častějšího slučování s hlavní větví. Práce na vedlejších větvích se nedoporučuje. [18]

2.3.3 Automatizované sestavení

Vaše sestavení by mělo být plně automatizované a zahrnovat veškeré kroky potřebné k vytvoření funkční aplikace. Automatizování procesů kompilace je tedy velice chtěná, aby výsledná aplikace byla připravena k použití. [18]

2.3.4 Automatické testování

Kontinuální testování je zásadní součástí CI/CD. Automatické testy by měly ověřit integritu, kvalitu a bezpečnost kódu. Selhání testu by mělo vést k neúspěchu sestavení a upozornit na potenciální problémy. [18]

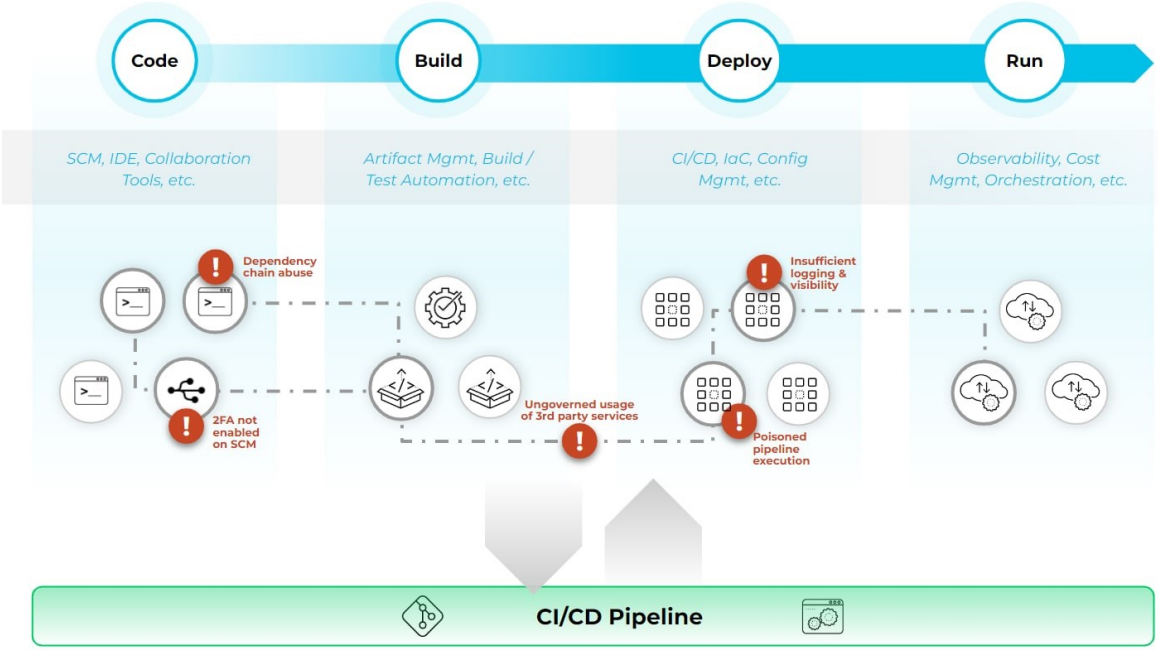
2.3.5 Časté iterace

Práce s menšími a častějšími iteracemi je klíčová a umožní rychlejší nápravu chyb. Kdyby se udělala velká část kódu a teprve poté by se dal *commit*[21] do repositáře, oprava jakékoliv chyby by mohlo být náročná. [18]

2.4 Bezpečnost v CI/CD

CI/CD security distribuuje bezpečnostní opatření po celém CI a CD. Je to také klíčová součást DevSecOps, což je strategie, která integruje bezpečnost do každé fáze vývoje softwaru, od počátku až po nasazení [20]. Tato komponenta odmítá myšlenku, že by bezpečnost byla samostatnou fází, a namísto toho se snaží zajisti, aby byla nedílnou součástí dodávání aplikací. Cílem CI/CD security je identifikovat a minimalizovat rizika. [21]

2.4.1 Bezpečnostní hrozby



Obrázek 4. Bezpečnosti hrozby pro CI/CD[22]

[21] Commit = Odevzdání kódu do větve

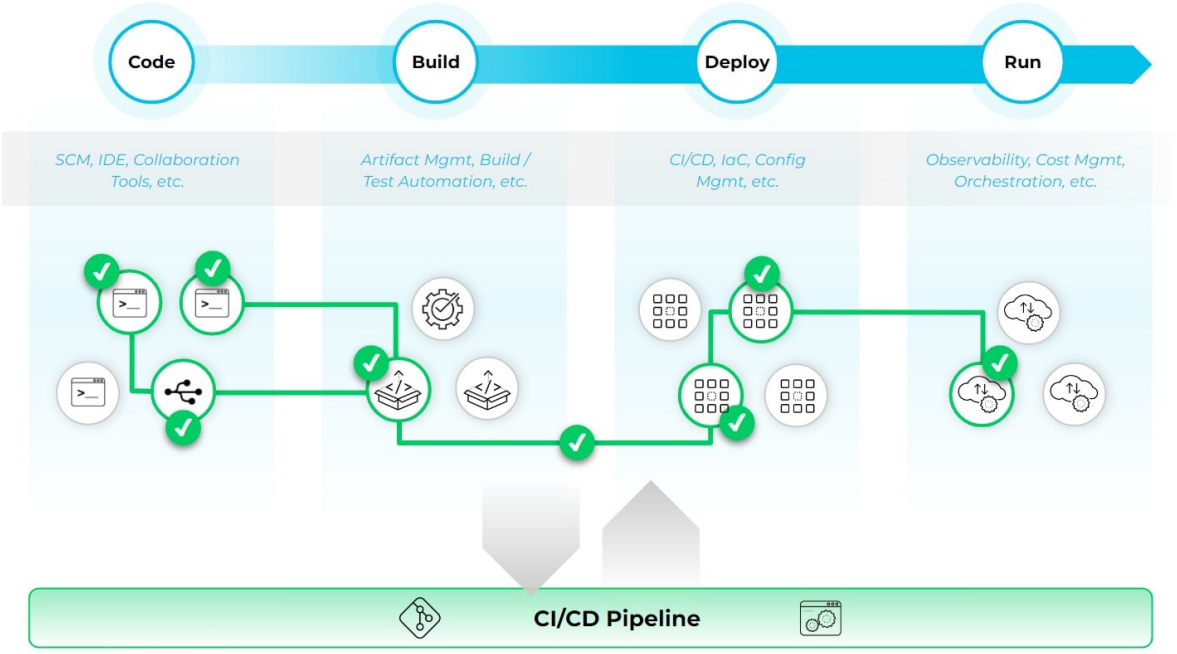
[22] *Software supply chains are complex with a range of tools connected to highly sensitive source code, yet most organizations have little visibility*. Online. In: Paloalto networks. C2024. Dostupné z: https://www.paloaltonetworks.com/cyberpedia/what-is-ci-cd-security. [cit. 2024-05-03].

V oblasti CI/CD se skrývají různé bezpečnostní hrozby, které mohou využít zranitelnosti od nedostatečných mechanismů řízení toku až po nesprávnou integritu prvků. Tyto hrozby mohou vést k neoprávněnému přístupu, útokům na integritu softwaru či úniku citlivých informací. Jednou z klíčových strategií pro jejich zvládnutí je příprava na nejvýznamnější bezpečnostní rizika související s CI/CD podle OWASP Top 10. [21]

- Nedostatečné mechanismy řízení toku (CICD-SEC-1)
    - Mohou umožnit útočníkům manipulovat s tokem změn kódu
- Nedostatečné řízení identity a přístupu (CICD-SEC-2)
    - Může vést k neoprávněnému přístupu k důležitým zdrojům
- Zneužití závislostního řetězce (CICD-SEC-3)
    - Umožňuje útočníkům vkládat zákeřný kód z veřejných repositářů
- Otrávené spouštění pipeline – PPE (CICD-SEC-4)
    - Umožňuje útočníkům vložit zákeřný kód do procesu sestavení nebo nasazení
- Nedostatečné kontrolní mechanismy založené na pipeline (CICD-SEC-5)
    - Mohou umožnit neoprávněný přístup k důležitým komponentám
- Nedostatečná správa přístupových údajů (CICD-SEC-6)
    - Může poskytnout útočníkům prostředky k ohrožení bezpečnosti pipeline
- Nezabezpečení konfigurace systému (CICD-SEC-7)
    - Útočníci získají neoprávněný přístup nebo mohou provádět zákeřné aktivity
- Neověřené používání služeb třetích stran (CICD-SEC-8)
    - Může vystavit pipeline rizikům
- Nesprávná validace integrity artefaktů (CICD-SEC-9)
    - Může vést k nasazení škodlivému softwaru
- Nedostatečné protokolování a viditelnost (CICD-SEC-10)
    - Znemožňuje detekci a reakci na bezpečnostní incidenty

[21]

2.4.2 Bezpečnostní mechanismy v CI/CD pipeline



Obrázek 5. Zabezpečení CI/CD[23]

Tyto mechanismy pomáhají minimalizovat rizika a zajišťují, že bezpečnost je integrována do každé fáze CI/CD [21]

- Statické testování bezpečnosti aplikací (SAST)
  - Skenuje zdrojový kód na známé rizika již v rané fázi vývoje
- Dynamické testování bezpečnosti aplikací (DAST)
  - Provádí testy na běžících aplikacích ke konci CI/CD pipeline
- Analýza složení softwaru (SCA)
  - Kontroluje rizikovost komponentů s otevřeným zdrojovým kódem (open-source)
- Správa tajemství
  - Zajišťuje bezpečné zacházení s citlivými daty. Například API klíče, přihlašovací údaje, certifikáty, …
- Bezpečnost kontejnerů
  - Prověřuje konfigurace, závislosti a chování kontejnerů

[23] *Securing the CI/CD pipeline requires security throughout the CI/CD pipeline*. Online. In: Paloalto networks. C2024. Dostupné z: https://www.paloaltonetworks.com/cyberpedia/what-is-ci-cd-security. [cit. 2024-05-03].

- Bezpečnost infrastruktury jako kódu (IaC)
    - Analyzuje skripty infrastruktury na konfigurační chyby nebo porušení bezpečnostních pravidel

[21]

3 NÁSTROJE PRO AUTOMATICKÉ TESTOVÁNÍ SOFTWARU

3.1 Výběr nástroje pro testování SW

S rostoucím rozsahem a složitostí softwarových projektů se stává stále důležitějším zvolit správné nástroje pro testování, které umožní efektivní a kvalitní ověření, zda aplikace funguje tak jak má. Správný výběr nástroje pro testování je proto klíčovým rozhodnutím, které může mít dopad na celý vývojový proces a kvalitu aplikace.

3.1.1 Důležité kritéria výběru nástroje pro testování softwaru

- Náklady
    - Jeden z nejdůležitějších faktorů pro výběr testovacího nástroje je rozpočet
    - Záleží na velikosti týmu nebo výběru použití
    - Finanční možnosti nebo potřeby uživatelů jsou rozhodující
- Kompatibilita
    - Efektivní spouštění a provádění testovacích případů
    - Ušetří čas při spouštění testovacích případů na různých platformách
- Snadnost použití
    - Je důležité si vybrat nástroj, který je uživatelsky přívětivý
    - Pokud ne, testovací proces může být náročný
    - Seznámit se s nástrojem může být náročné
- Technická podpora
    - Prodejce testovacích nástrojů by měl poskytnout technickou podporu
    - Velkou výhodou je obsah online chatu nebo jiné rychlé podpory
- Komunitní podpora
    - Užitečné pro nezávislé testery

[22]

3.2 Popis nejčastějších nástrojů pro automatické testování SW

3.2.1 Selenium

Obrázek 6. Selenium logo[24]

Selenium je skvělý nástroj na automatické testování prohlížečů. Usnadňuje opakující se úkoly a umožňuje efektivní testování na různých prohlížečích a operačních systémech. Selenium je *open-source* a má spoustu nástrojů a knihoven a je proto oblíbený u testerů nebo i u vývojářů. [23]

Výhody:

- Selenium je *open-source* a zdarma
- Testování je možné na různých prohlížečích a operačních systémech
- Selenium podporuje většinu populárních programovacích jazyků
- Jedná se o jeden z nejpopulárnějších testovacích nástrojů
    - Technická podpora je na vysoké úrovni
    - Vždy se najde komunitní pomoc

Nevýhody:

- Jedná se o nástroj pro testování prohlížečů
    - Testovací scénáře mohou vyžadovat pravidelné aktualizace
    - Není to vhodný nástroj pro testování počítačových nebo mobilních aplikací
- V některých situacích, pomalá rychlost testování
- Nemá vestavěné hlášení výsledků
    - Musí se docílit přidaným nástrojem.

[23]

---

[24] *Selenium logo*. Online. In: Selenium. C2024. Dostupné z: https://www.selenium.dev/. [cit. 2024-05-03].

3.2.2 Katalon

Obrázek 7. Katalon logo[25]

Nástroj pro automatické testovaní, který byl poprvé uveden v roce 2015. Je primárně určen pro automatizaci testování uživatelského rozhraní bez nutnosti psaní kódu. Podporuje testování webových rozhraní, počítačových aplikací pro Windows i mobilních aplikací. [24]

Výhody:

- Podporuje různé typy testování
  - Testování přes klíčová slova, na které není potřeba testovací skript
  - Testování přes data, které zahrnuje čtení testovacích skriptů
- Lehce pochopitelný analytický přehled a reporty
- Uživatelsky přívětivé rozhraní
- Vhodný pro lidi bez programátorských dovedností

Nevýhody:

- Pouze jeden skriptovací jazyk – Groovy
  - Pro platformu Java, má jednoduchou syntaxi i integraci [25]
- Menší komunita ve srovnání se Selenium
- Není open-source jako Selenium, takže nelze použít komunitní přídavné nástroje
  - Katalon má spoustu komponentů a další k zakoupení
- Vyšší nároky na hardware a paměť
- Některé pokročilejší funkce jsou pouze za platbu

---

[25] *Katalon logo*. Online. In: Katalon. C2024. Dostupné z: https://katalon.com/. [cit. 2024-05-03].

[24]

3.2.3 Cypress

Obrázek 8. Cypress logo[26]

Nástroj pro testování webových aplikací, který je založen na JavaScriptu. Primárně slouží k automatizaci end-to-end testů. Tento nástroj funguje přímo v prohlížeči pomocí techniky manipulace s DOM [26], což je platformově a jazykově neutrální rozhraní, které umožňuje skriptům dynamicky přistupovat k obsahu nebo struktuře a aktualizovat je. [27]

Výhody:

- Univerzální, jelikož je napsán v JavaScriptu a při spuštění využívá Node.js
- Jednoduchá instalace na rozdíl od Selenium.
  - Obsahuje přednastavený prohlížeč, není potřeba složitá konfigurace prostředí
- Podporuje testování na různých prohlížečích
- Detailní analýza chyb s obrázky selhání testů
  - Poskytuje návrhy na opravu chyb při jejich zjištění
- Nástroj je známý tím, že je velice rychlý
  - Má zabudované automatické čekání ve *frameworku*, to znamená že *framework* automaticky čeká na DOM, animace a dalších prvky

[26]

Nevýhody:

- Pouze pro testování webových aplikací

---

[26] *Cypress logo*. Online. In: Cypress. Dostupné z: https://www.cypress.io/blog. [cit. 2024-05-03].

- Cypress nemůže testovat 2 různé prohlížeče najednou
- Pouze jeden skriptovací jazyk – JavaScript
- Nepodporuje více panelů najednou

[28]

3.2.4 Ranorex

Obrázek 9. Ranorex logo[27]

Nástroj určený pro testování desktopových, webových aplikací na platformě Windows. Nevyužívá specifický skriptovací jazyk, ale je postaven na platformě .NET od Microsoftu. [29]

Výhody:

- Testování webových, mobilních i desktopových aplikací
- Uživatelsky přívětivé prostředí s mnoha funkcemi
  - Není potřeba programovat testy
  - Funkce jako například nahrávání nebo přehrávání akcí
- Poskytuje nástroj pro porovnávání obrazových prvků
  - Dobré pro přesné ověření uživatelského rozhraní aplikace
- Rychlá a kvalitní technická podpora
- Automatické generování reportů

Nevýhody:

- Placená licence

---

[27] *Ranorex Logo*. Online. In: Ranorex. C2024. Dostupné z: https://www.ranorex.com/features/. [cit. 2024-05-03].

- Podporuje pouze .NET jazyky, jako například C# a VB.NET
- Nepodporuje macOS
- Menší uživatelská komunita než i jiných nástrojů

[29]

3.3 Porovnání vybraných testovacích nástrojů

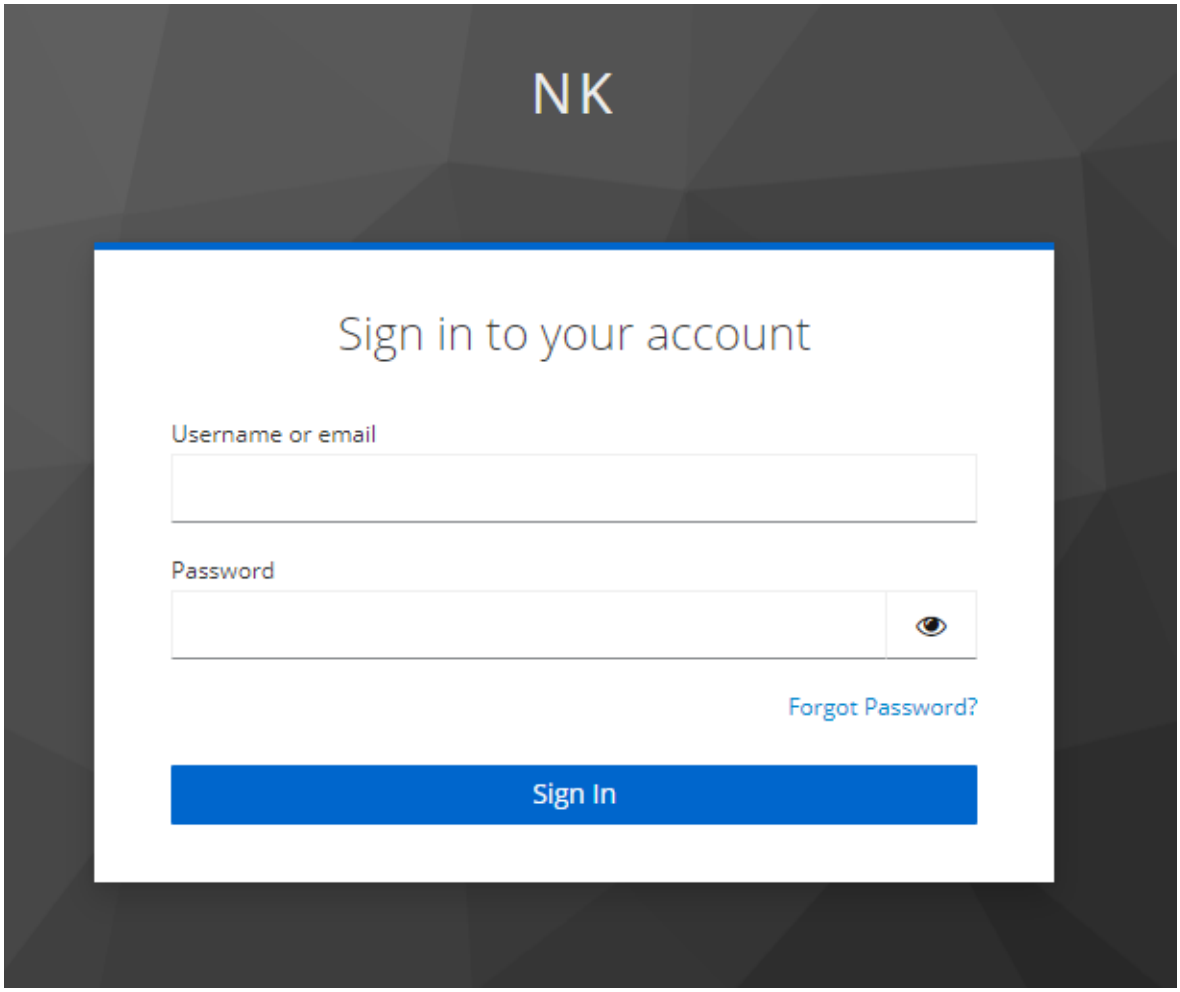
Tabulka 2. Tabulka rozdílů mezi nástroji testování (Vlastní)

Nástroj	Aplikace	Skripty	Jazyky	Analýza výsledků	Cena
Selenium	Web	Psané	Python, Java, …	Ne	Zdarma
Katalon	Web, Mobil, Počítač	Volitelné	Groovy	Ano	Volitelné
Cypress	Web	Psané	JavaScript	Ano	Volitelné
Ranorex	Web, Počítač	Psané	C#, VB.NET	Ano	Licencované

II. PRAKTICKÁ ČÁST

4 NÁVRH A REALIZACE SYSTÉMU TESTOVÁNÍ SW

Praktická část je realizována na projektu registru digitalizace, který provozuje Národní knihovna. Jedná se o národní registr digitalizovaných dokumentů, jejímž základním posláním je zamezit nechtěné duplicitní digitalizaci a také přispět k efektivnímu sdílení výsledků různorodých digitalizačních aktivit. Testování probíhá v jazyce Python pomocí *frameworku* Selenium a je zaměřeno na hlavní stránku, tudíž na přihlašování do systému. Při této práci jsem spolupracoval s firmou Principia, která disponuje vlastním rozšířením pro Selenium zvaném *„Princster“*, které běží v Pythonu. Toto rozšíření přispívá k efektivitě testovacího procesu. Jelikož se jedná o firemní rozšíření, nemůžu zveřejnit jeho kód.



Obrázek 10. Uvítací stránka registru digitalizace [28]

[28] *Registr Digitalizace*. Online. Dostupné z: https://edtest.registrdigitalizace.cz/. [cit. 2024-05-06].

4.1 Konfigurační soubor .yml

Pro konfiguraci testovacích souborů jsem použil formát YAML (.yml), který má přehledný způsob ukládání dat. Jeho přehledná struktura umožňuje snadněji organizovat a upravovat nastavení testovacích scénářů, což je skvělé pro orientaci a správu testovacích případů. Na *obrázku č.11* jde hezky vidět struktura, o které jsem psal a kterou jsem níže rozebral dopodrobna.

```
---
test_registr_digitalizace:
    url: https://edtest.registrdigitalizace.cz/
    login_title: Sign in to nk
    input_usr: input[name='username']
    confirm_usr: False
    input_pwd: input[name='password']
    submit: input[class='pf-c-button pf-m-primary pf-m-block btn-lg']

    login_ok:
        username:
        password:
        validate_ele: div[class='sc-hKgILt cSPLMw']
        validate_txt: name

    login_not_ok:
        username: aa
        password: aa

    login_logout:
        logout_button: button[class='sc-fubCfw bvcEpK']

    forgot_password:
        forgot_button:
/html[1]/body[1]/div[1]/div[2]/div[1]/div[1]/div[1]/div[1]/form[1]/div[3]
/div[2]/span[1]/a[1]
        validate_ele: h1[id='kc-page-title']
        validate_txt: Forgot Your Password?

...
```

Obrázek 11. Struktura konfiguračního souboru (Vlastní)

Část kódu pod *„test_registr_digitalizace“* definuje parametry a prvky, které jsou použity při testování funkcionality přihlašování na webu registru digitalizace. Nejprve je určena adresa URL testované stránky, která se nachází pod proměnnou *„url“*. Dále je specifikován text *„login_title“*, díky kterému lze jednoduše zjistit, že se nacházíme na titulní stránce webu. Poté jsou identifikovány vstupní pole pro uživatelské jméno a heslo mezi nimiž je proměnná *„confirm_usr“*, která je nastavena na *False*. Značí to, jestli je potřeba potvrdit uživatelské jméno předtím, než se dostaneme k poli s heslem. V tomto případě to tak není, proto *False*. Nakonec je určeno tlačítko pro přihlášení do systému s názvem *„submit“*.

```
test_registr_digitalizace:
    url: https://edtest.registrdigitalizace.cz/
    login_title: Sign in to nk
    input_usr: input[name='username']
    confirm_usr: False
    input_pwd: input[name='password']
    submit: input[class='pf-c-button pf-m-primary pf-m-block btn-lg']
```

Obrázek 12. Konfigurace pro testy (Vlastní)

Další část kódu definuje scénář *„login_ok“*, který popisuje úspěšné přihlášení do systému. Jsou zde specifikovány testovací údaje pro uživatelské jméno a heslo (*„username“* a *„password“*). Dále je zde uveden element *„validate_ele“*, který po přihlášení bude ověřovat, zda proběhlo úspěšně. Toho docílí tak, že bude svoje informace bude porovnávat s textem *„validate_txt“*. Test proběhne úspěšně pokud text je stejný jako text, který se má v elementu nacházet. Na *obrázku č.11* a *obrázku č.13* nejde vidět *„username“* a *„password“*, je to z toho důvodu, že se jedná o údaje pro přihlášení do systému, tudíž je nemůžu zveřejnit.

```
login_ok:
    username:
    password:
    validate_ele: div[class='sc-hKgILt cSPLMw']
    validate_txt: name
```

Obrázek 13. Testovací scénář pro úspěšné přihlášení (Vlastní)

Další testovací scénář *„login_not_ok“* obsahuje jen uživatelské jméno a heslo. Konkrétní jméno a heslo je náhodně vybrané tak aby neexistovalo nebo nemohlo existovat.

```
login_not_ok:
    username: aa
    password: aa
```

Obrázek 14. Testovací scénář pro neúspěšné přihlášení (Vlastní)

Další konfigurace s názvem *„login_logout“* obsahuje pouze jeden element, který identifikuje tlačítko. Žádný jiné element není potřeba jelikož využijeme již hotový test.

```
login_logout:
    logout_button: button[class='sc-fubCfw bvcEpK']
```

Obrázek 15. Testovací scénář odhlášení ze systému (Vlastní)

Testovací scénář *„forgot_password“* obsahuje tlačítko *„forgot_button“*, které je identifikované pomocí absolutní XPath, který začíná od kořene dokumentu a určuje úplnou cestu k prvku. Není ovšem tak flexibilní a při nějaké změně webu se může rozbít. [30] Poté, stejně jako u testování úspěšného přihlášení, je specifikován element *„validate_ele“* a text, který by měl element obsahovat *„validate_txt“*.

```
forgot_password:
    forgot_button:
/html[1]/body[1]/div[1]/div[2]/div[1]/div[1]/div[1]/div[1]/form[1]/div[3]
/div[2]/span[1]/a[1]
    validate_ele: h1[id='kc-page-title']
    validate_txt: Forgot Your Password?
```

Obrázek 16. Testovací scénář pro zapomenutí hesla (Vlastní)

4.2 Implementace pomocí Selenium v Pythonu

Tato část kódu obsahuje nastavení a definici testovacích scénářů pro webovou aplikaci. Na začátku je importování knihoven. První řádek importuje modul *„os“*, který poskytuje funkce pro práci s operačním systémem jako například manipulace se soubory. Druhý řádek importuje module *„pytest“*, což je testovací *framework* pro Python. Třetí řádek importuje třídu *„PrincsterSelenium“* z modulu *„selenium_tests“*. Jako další jsou definovány vlastní nastavení, jako je prohlížeč (Chrome) a název konfiguračního souboru (*„registr_digitalizace“*).

Slovo *„self“*, které se často v kódu vyskytuje, označuje instanci třídy a umožňuje přístup k atributům a metodám této instance. [31]

```
import os
import pytest
from selenium_tests import PrincsterSelenium


# CUSTOM SETTINGS
BROWSER = 'chrome'
PROJECT = 'registr_digitalizace'



class MenuTests(PrincsterSelenium):

    def open_webpage(self):
        self.cfg = self.get_project_config(PROJECT, os.path.base-
name(__file__).split('.')[0])
        self.open(self.cfg['url'])
        self.assert_title(self.cfg['login_title'])

    def test_login_ok(self):
        self.open_webpage()
        self.type(self.cfg['input_usr'], self.cfg['login_ok']['userna-
me'])
        if self.cfg['confirm_usr']:
            self.click(self.cfg['submit'])
        self.type(self.cfg['input_pwd'], self.cfg['lo-
gin_ok']['password'])
        self.click(self.cfg['submit'])
        self.assert_text(self.cfg['login_ok']['validate_txt']),
self.cfg['login_ok']['validate_ele']

    def test_login_not_ok(self):
        self.open_webpage()
        self.type(self.cfg['input_usr'],
```

```
self.cfg['login_not_ok']['username'])
        if self.cfg['confirm_usr']:
            self.click(self.cfg['submit'])
        self.type(self.cfg['input_pwd'], self.cfg['lo-
gin_not_ok']['password'])
        self.click(self.cfg['submit'])
        self.assert_title(self.cfg['login_title'])

    def test_login_logout(self):
        self.test_login_ok()
        self.click(self.cfg['login_logout']['logout_button'])
        self.assert_title(self.cfg['login_title'])

    def test_forgot_password(self):
        self.open_webpage()
        self.click(self.cfg['forgot_password']['forgot_button'])
        self.assert_text(self.cfg['forgot_password']['validate_txt']),
self.cfg['forgot_password']['validate_ele']
```

Obrázek 17. Kód s funkcemi (metodami) testovacích scénářů (Vlastní)

Třída *„MenuTests“* dědí z třídy *„PrincsterSelenium“*. První funkce (metoda) je *„open_webpage()“*, která slouží k otevření webové stránky a ověření správnosti načtení. Uvnitř této funkce se nejprve načítají konfigurační údaje projektu pomocí metody *„get_project_config()“*. Poté se otevře webová stránka pomocí *„open()“*, kde *„self“* označuje instanci třídy, a nakonec se pomocí *„assert_title()“* ověří, zda je stránka správně načtená.

```
class MenuTests(PrincsterSelenium):

    def open_webpage(self):
        self.cfg = self.get_project_config(PROJECT, os.path.base-
name(__file__).split('.')[0])
        self.open(self.cfg['url'])
        self.assert_title(self.cfg['login_title'])
```

Obrázek 18. Kód pro otevření stránky (Vlastní)

Metoda *„test_login_ok()“* spouští test pro ověření úspěšného přihlášení. Nejprve se otevře stránka pomocí metody *„open_webpage()“*, poté se do příslušných polí zadají hodnoty, které jsou načteny z konfiguračního souboru pod testem *„login_ok“*. Jako další krok je kliknutí na tlačítko pro přihlášení, které je také v konfiguračním souboru specifikováno a jako poslední krok je ověření zda je text stejný jako text konkrétního elementu.

```
def test_login_ok(self):
    self.open_webpage()
    self.type(self.cfg['input_usr'], self.cfg['login_ok']['username'])
    if self.cfg['confirm_usr']:
        self.click(self.cfg['submit'])
    self.type(self.cfg['input_pwd'], self.cfg['login_ok']['password'])
    self.click(self.cfg['submit'])
    self.assert_text(self.cfg['login_ok']['validate_txt']), self.cfg['lo-
gin_ok']['validate_ele']
```

Obrázek 19. Kód pro otestování úspěšného přihlášení (Vlastní)

Tato metoda je stejná jako *„test_login_ok“*, ale bere konkrétní údaje z jiného testu v konfiguračním souboru.

```
def test_login_not_ok(self):
    self.open_webpage()
    self.type(self.cfg['input_usr'], self.cfg['login_not_ok']['userna-
me'])
    if self.cfg['confirm_usr']:
        self.click(self.cfg['submit'])
    self.type(self.cfg['input_pwd'], self.cfg['lo-
gin_not_ok']['password'])
    self.click(self.cfg['submit'])
    self.assert_title(self.cfg['login_title'])
```

Obrázek 20. Kód pro otestování neúspěšného přihlášení (Vlastní)

Metoda *„test_login_logout“* spouští test, který odhlásí uživatele po jeho přihlášení. Využívá tedy metodu *„test_login_ok“*, pro úspěšné přihlášení. Po provedení této metody je kliknutí na tlačítko pro odhlášení, které je specifikované v konfiguračním souboru pod příslušným testem a ověření zda se nacházíme zpět na hlavní stránce (přihlašovací stránka).

```
def test_login_logout(self):
    self.test_login_ok()
    self.click(self.cfg['login_logout']['logout_button'])
    self.assert_title(self.cfg['login_title'])
```

Obrázek 21. Kód pro otestování odhlášení (Vlastní)

Metoda „test_forgot_password“ spouští test pro ověření funkčnosti tlačítka *„Forgot password?“* na přihlašovací stránce, jak lze vidět na *obrázku č. 10*. Po kliknutí na příslušné tlačítko, které je specifikováno v konfiguračním souboru pod testem *„forgot_password“*, se ověří, zda jsme na správné lokaci pomocí elementu a textu, který se v elementu očekává.

```
def test_forgot_password(self):
    self.open_webpage()
    self.click(self.cfg['forgot_password']['forgot_button'])
    self.assert_text(self.cfg['forgot_password']['validate_txt']),
self.cfg['forgot_password']['validate_ele']
```

Obrázek 22. Kód pro otestování tlačítka *„Forgot password?“* (Vlastní)

4.3 Spouštění v CI/CD

Tyto 4 základní testy jsou vytvořené způsobem, díky kterému se budou lehce spouštět v CI/CD a díky kterému se mohou lehce aktualizovat v případě aktualizace testované aplikace. Testy budou spouštěny přes GitLab CI/CD s využitím Dockeru. Tento proces umožňuje spouštění testů v kontejnerech místo běhu testů na dedikovaném serveru CI/CD. Klíčovým prvek je registrace GitLab běžce, který používá Docker executor. [32]

Tento kus kódu je konfigurace pro GitLab CI/CD a má 2 sekce. První sekce je *„stages“*, která definuje fáze, které budou prováděny v rámci procesu CI/CD. Druhá sekce s názvem *„princster“* je úloha v rámci CI/CD procesu. Každá úloha provádí určité operace, v mojí práci je pouze jedna úloha, která provádí tyto operace:

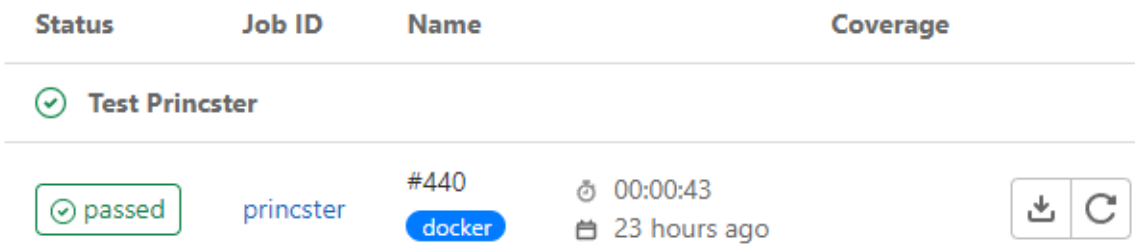
- *„Image“*
    - Specifikuje docker image, která se používá pro spouštění úloh
- *„Tags“*
    - Zde jsou definovány tagy pro úlohu
- *„Stage“*
    - Určuje, ve které fázi CI/CD má být úloha spuštěna
- *„Only“*
    - Podmínky, za kterých se má tato úloha spouštět
    - *„Refs“* znamená větev, v tomto případě větev *„devops“*
- *„Script“*
    - Obsahuje seznam příkazů, které mají být provedeny v rámci úlohy
    - Například spuštění testů, vytvoření adresáře, …
- *„Cache“*
    - Tady je možné nastavit cachování, což může urychlit proces spouštění
- *„Artifacts“*
    - Tato sekce definuje artefakty, které mají být zachovány po dokončení úlohy

```
stages:
  - test-princster


princster:
  image: principiank.azurecr.io/princster:${PRINSTER_VERSION}
  tags:
    - docker
  stage: test-princster
  only:
    refs:
      - devops
  script:
    - python3 /princster/selenium_tests/run_tests.py
${CI_PROJECT_DIR}/tests/prinster/config.yml
    - mkdir ${CI_PROJECT_DIR}/tests/reports
    - cp -r /princster/selenium_tests/reports/*
${CI_PROJECT_DIR}/tests/reports/
    - ls -la ${CI_PROJECT_DIR}/tests/reports
  cache:

  artifacts:
    untracked: true
    paths:
      - tests/reports/
```

Obrázek 23. Konfigurace CI/CD (Vlastní)

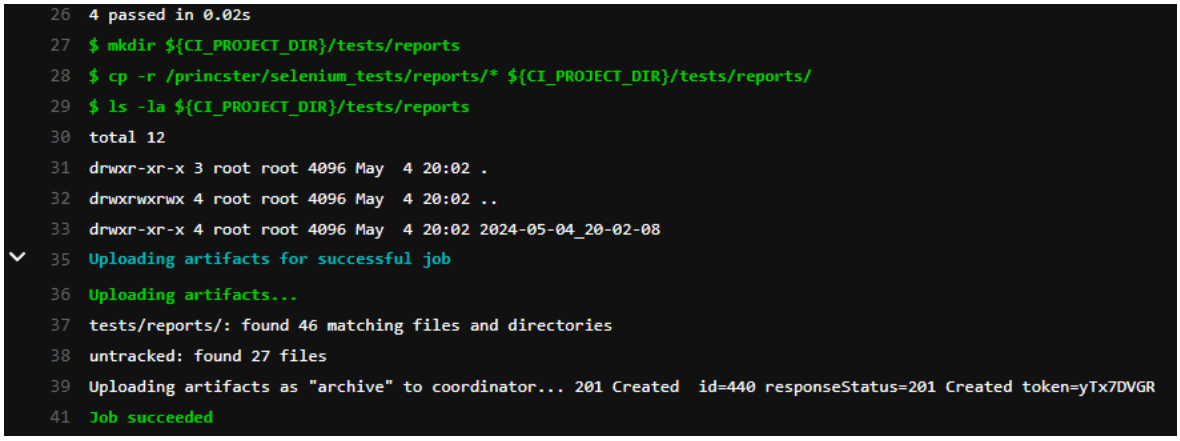
5 VYHODNOCENÍ POUŽITÉHO NÁSTROJE

Selenium je velice rozšiřitelné a poskytuje tudíž široké možnosti pro automatizaci testování webových aplikací. Testy ovšem mohou být náchylné na pomalou odezvu, i v tomto případě na *obrázku č. 24* jde vidět, že pipeline trvala 43 sekund a běžely pouze 4 jednoduché testy. V moment, co běží desítky až stovky testů, může být Selenium velice zdlouhavé a může prodloužit nasazení aplikace. Selenium je lehce integrováno do CI/CD.



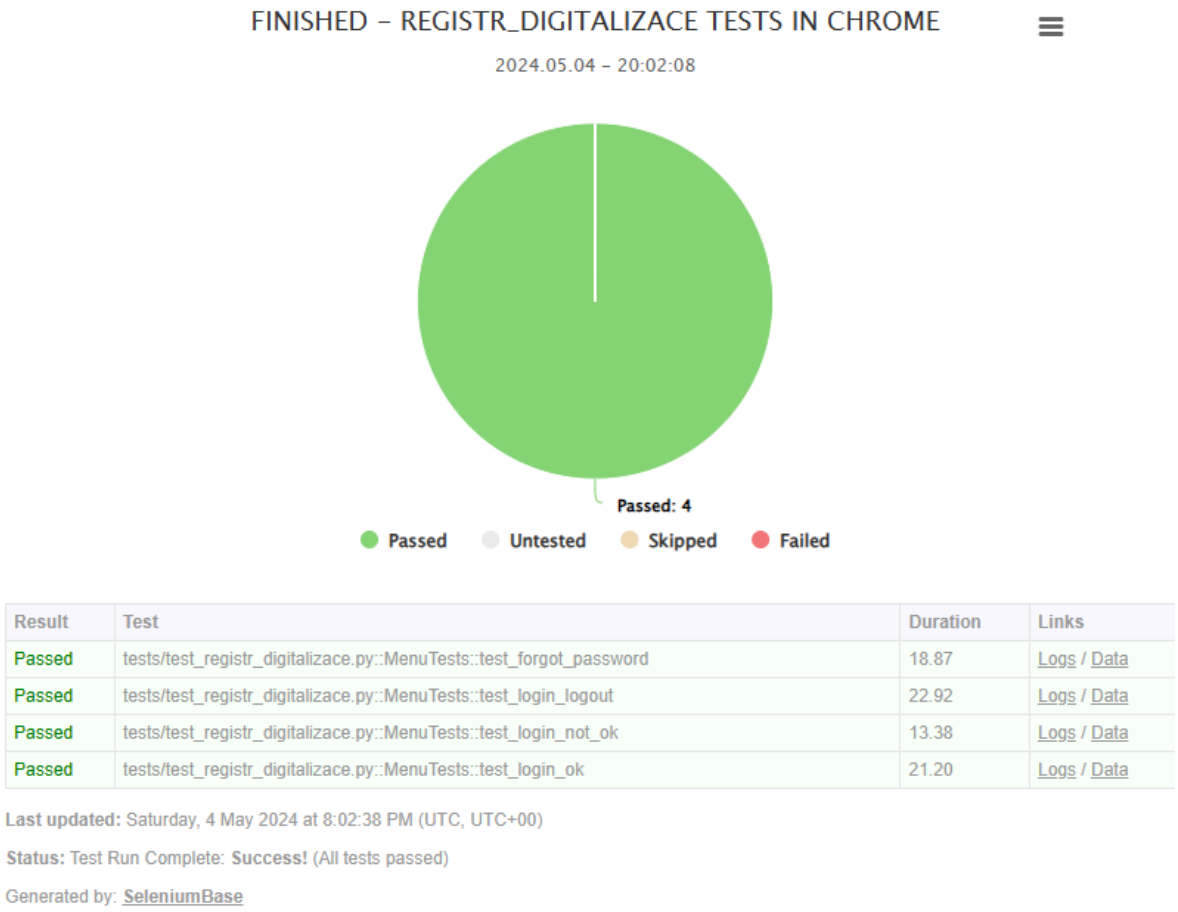
Obrázek 24. Status CI/CD pipeline (Vlastní)

GitLab umožňuje velice snadnou konfiguraci pomocí souboru *„.gitlab-ci.yml“*, který lze vidět na *obrázku č.23*. GitLab obsahuje velice podrobnou a přehlednou dokumentaci pro CI/CD. V případě výsledků CI/CD je zde víceméně přehledná konzole, jejíž část je na *obrázku č. 25*. Jde vidět, že 4 testy prošly a výsledky (artefakty) se zapsali do jednotlivých složek.



Obrázek 25. Konzole CI/CD pipeline (Vlastní)

Jeden ze zmíněných artefaktů je na *obrázku č. 26*. Jedná o jednoduché hlášení výsledku testů.



Result	Test	Duration	Links
Passed	tests/test_registr_digitalizace.py::MenuTests::test_forgot_password	18.87	Logs / Data
Passed	tests/test_registr_digitalizace.py::MenuTests::test_login_logout	22.92	Logs / Data
Passed	tests/test_registr_digitalizace.py::MenuTests::test_login_not_ok	13.38	Logs / Data
Passed	tests/test_registr_digitalizace.py::MenuTests::test_login_ok	21.20	Logs / Data

Last updated: Saturday, 4 May 2024 at 8:02:38 PM (UTC, UTC+00)

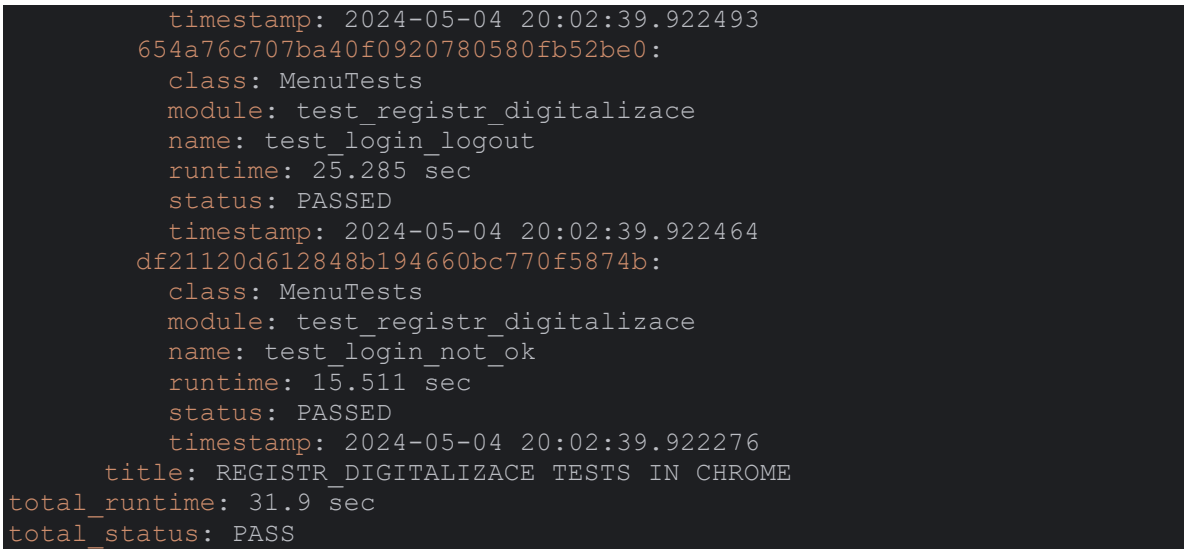
Status: Test Run Complete: **Success!** (All tests passed)

Generated by: SeleniumBase

Obrázek 26. Výsledky testování (Vlastní)

Jako další z artefaktů je *obrázek č. 27*, což je podrobný popis testů. Soubor je uložený jako YAML a jde s ním dále pracovat na dalších projektech nebo výsledcích.

```yaml
config: /builds/nk/nk/tests/prinster/config.yml
princster_id: 2024-05-04_20-02-08
projects:
  registr_digitalizace:
    chrome:
      browser_runtime: 31.37 sec
      browser_status: PASSED
      failed: 0
      passed: 4
      tests:
        0336be5e0af34e30a5fcd10dc2e65d79:
          class: MenuTests
          module: test_registr_digitalizace
          name: test_login_ok
          runtime: 23.486 sec
          status: PASSED
          timestamp: 2024-05-04 20:02:39.922516
        071ad71eadcd424bb785b622dae77875:
          class: MenuTests
          module: test_registr_digitalizace
          name: test_forgot_password
          runtime: 20.310 sec
          status: PASSED
```



Obrázek 27. Výsledky uložené pro další zpracování (Vlastní)

ZÁVĚR

V bakalářské práci jsem analyzoval a porovnal klíčové aspekty testování softwaru a CI/CD pipeline. V teoretické části jsem se zaměřil na historii testování softwaru, strategie, a především typy testů. Dále jsem vysvětlil proces CI/CD a jeho hrozby. Poté jsem provedl porovnání nejčastěji používaných nástrojů pro automatizované testování softwaru.

V praktické části jsem vytvořil pár základních testů pro konkrétní projekt při spolupráci se společnosti Principia využívající Selenium a Python. Realizovaný systém umožňuje pravidelné testování funkcionality přihlašování do systému. Integrace do CI/CD pipeline dále zvyšuje efektivitu a kontrolu kvality softwaru.

Výsledkem této práce je nyní v praxi používaný systém, který přispívá ke kvalitě a spolehlivosti.

SEZNAM POUŽITÉ LITERATURY

[1] PRYTULENETS, Alesya. *A Brief History of Software Testing*. Online. In: DogQ Blog. C2024. Dostupné z: https://dogq.io/blog/a-brief-history-of-software-testing/.

[2] TERRAZAS, Armando. *History of Software Testing*. Online. In: Medium. [cca 2020]. Dostupné z: https://medium.com/@armandotrsg/the-evolution-of-software-testing-b379672877ae. [cit. 2024-05-03].

[3] *Manual Testing*. Online. JavaTpoint. C2011-2021. Dostupné z: https://www.javatpoint.com/manual-testing. [cit. 2024-05-03].

[4] *PROČ JE MANUÁLNÍ TESTOVÁNÍ DŮLEŽITÉ I V DNEŠNÍ DOBĚ?* Online. In: Praha Coding School. C2024. Dostupné z: https://prahacoding.cz/proc-je-dulezite-manualni-testovani/. [cit. 2024-05-03].

[5] *Black Box Testing*. Online. In: Imperva. C2024. Dostupné z: https://www.imperva.com/learn/application-security/black-box-testing/.

[6] HAMILTON, Thomas. *What is Grey Box Testing? Techniques, Example*. Online. In: Guru99. 2024. Dostupné z: https://www.guru99.com/grey-box-testing.html. [cit. 2024-05-03].

[7] *Automation Testing – Software Testing*. Online. In: GeeksforGeeks. Last Updated: 26 Mar, 2024. Dostupné z: https://www.geeksforgeeks.org/automation-testing-software-testing/. [cit. 2024-05-03].

[8] *Proč je manuální testování důležité v dnešní době?* Online. In: PrahaCodingSchool. C2024. Dostupné z: https://prahacoding.cz/proc-je-dulezite-manualni-testovani/. [cit. 2024-05-03].

[9] BAKHAREV, Nickolay. *Proč je manuální testování důležité v dnešní době?* Online. In: Bright. 2023. Dostupné z: https://brightsec.com/blog/unit-testing/. [cit. 2024-05-03].

[10] *Unit Testing – Software Testing*. Online. In: GeeksforGeeks. Last Updated: 23 Apr, 2024. Dostupné z: https://www.geeksforgeeks.org/unit-testing-software-testing/. [cit. 2024-05-03].

[11] AWATI, Rahul. *Integration testing or integration and testing (I&T)*. Online. In: Techtarget. C2006-2024. Dostupné z: https://www.techtarget.com/searchsoftwarequality/definition/integration-testing. [cit. 2024-05-03].

[12] *Integration Testing – Software Engineering*. Online. In: GeeksforGeeks. Last Updated: 23 Apr, 2024. Dostupné z: https://www.geeksforgeeks.org/software-engineering-integration-testing/?ref=lbp. [cit. 2024-05-03].

[13] https://www.techtarget.com/searchsoftwarequality/definition/system-testing

[14] *System Testing – Software Engineering*. Online. In: GeeksforGeeks. Last Updated: 23 Apr, 2024. Dostupné z: https://www.geeksforgeeks.org/system-testing/. [cit. 2024-05-03].

[15] *Acceptance Testing | What, Why, Types & How to Do?* Online. In: Testsigma. Dostupné z: https://testsigma.com/guides/acceptance-testing/#Advantages_and_Disadvantages_of_Acceptance_Testing. [cit. 2024-05-03].

[16] *A Brief History of CI/CD Tooling*. Online. In: Medium. 2023. Dostupné z: https://medium.com/@DiggerHQ/a-brief-history-of-ci-cd-tooling-5a67c2638f3a. [cit. 2024-05-03].

[17] *What Is Subversion? SVN Explained*. Online. In: Perforce. 2023. Dostupné z: https://www.perforce.com/blog/vcs/what-svn. [cit. 2024-05-03].

[18] *What is CI/CD?* Online. In: GitLab. c2024. Dostupné z: https://about.gitlab.com/topics/ci-cd/#what-is-continuous-integration-ci. [cit. 2024-05-03].

[19] PRABHAKAR, Komal J. *Elements Of CI/CD Pipeline*. Online. In: Medium. 2022. Dostupné z: https://medium.com/buildpiper/elements-of-ci-cd-pipeline-2a4f31161075. [cit. 2024-05-03].

[20] *What is DevSecOps?* Online. In: Paloalto networks. C2024. Dostupné z: https://www.paloaltonetworks.com/cyberpedia/what-is-devsecops. [cit. 2024-05-03].

[21] *What Is CI/CD Security?* Online. In: Paloalto networks. C2024. Dostupné z: https://www.paloaltonetworks.com/cyberpedia/what-is-ci-cd-security. [cit. 2024-05-03].

[22] COLLINS, Tom. *What are different Software Testing Tools?* Online. In: BrowserStack. 2023. Dostupné z: https://www.browserstack.com/guide/what-are-testing-tools. [cit. 2024-05-03].

[23] CSER, Tamas. *Automation Testing Tools – A Deep Dive*. Online. In: Functionize. Updated March 2024. Dostupné z: https://www.functionize.com/automated-testing/automation-testing-tools-deep-dive. [cit. 2024-05-03].

[24] *The Good and the Bad of Katalon Studio Automation Testing Tool*. Online. In: Altexsoft. Last updated: 15 čvn, 2021. Dostupné z: https://www.altexsoft.com/blog/the-good-and-the-bad-of-katalon-studio-automation-testing-tool/. [cit. 2024-05-03].

[25] *Apache Groovy*. Online. C2003-2024. Dostupné z: https://groovy-lang.org/. [cit. 2024-05-03].

[26] KINSBRUNER, Eran. *What is Cypress Testing? What It Is and How to Get Started*. Online. In: Perfecto. 2021. Dostupné z: https://www.perfecto.io/blog/cypress-testing. [cit. 2024-05-03].

[27] *Javascript DOM Manipulation*. Online. In: StudySmarter. C2024. Dostupné z: https://www.studysmarter.co.uk/explanations/computer-science/computer-programming/javascript-dom-manipulation/. [cit. 2024-05-03].

[28] UNADKAT, Jash. *Javascript DOM Manipulation*. Online. In: BrowserStack. 2023. Dostupné z: https://www.browserstack.com/guide/cypress-vs-selenium. [cit. 2024-05-03].

[29] *The Good and the Bad of Ranorex GUI Test Automation Tool*. Online. In: Altexsoft. 2018. Dostupné z: https://www.altexsoft.com/blog/the-good-and-the-bad-of-ranorex-gui-test-automation-tool/. [cit. 2024-05-03].

[30] DAITYARI, Shaumik. *How to use XPath in Selenium? (using Text, Attributes, Logical Operators)*. Online. In: BrowserStack. 2023. Dostupné z: https://www.browserstack.com/guide/xpath-in-selenium. [cit. 2024-05-04].

[31] VADAPALLI, Pavan. *Self in Python*. Online. In: . 04/03/2024 Last Updated. Dostupné z: https://www.upgrad.com/tutorials/software-engineering/python-tutorial/self-in-python/. [cit. 2024-05-04].

[32] *GitLab Docs*. Online. GitLab. Dostupné z: https://docs.gitlab.com/ee/ci/docker/using_docker_images.html. [cit. 2024-05-05].

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

IBM	International Business Machines Corporation
CI/CD	Continuous integration/Continuous delivery (deployment)
CI	Continuous integration
YAML	Yet another markup language
SVN	Apache Subversion
CD	Continuous delivery (deployment)
DevSecOps	Development, Security a Operations
OWASP	Open Web Application Security Project
CICD-SEC	CI/CD Security
PPE	Poisoned Pipeline Execution
SAST	Static application security testing
DAST	Dynamic application security testing
SCA	Software composition analysis
API	Application programming interface
IaC	Infrastructure as code
SW	Software (aplikace)
DOM	The Document Object Model
macOS	Systémová platforma od firmy Apple
URL	Uniform Resource Locator

SEZNAM OBRÁZKŮ

SEZNAM TABULEK