Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Využití verzovacího systému GIT pro správu projektu a automatický deployment se systémem kontroly

Jakub Joukl

Bakalářská práce

2023

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2022/2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Jakub Joukl**
Osobní číslo:	**I20106**
Studijní program:	**B0688A140009 Informační technologie**
Téma práce:	**Využití verzovacího systému GIT pro správu projektu a automatický deployment se systémem kontroly**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování

Cílem práce je zpracovat podrobnou analýzu využití verzovacího systému GIT při tvorbě softwarového systému, možností využití systému GIT v teamových projektech a analýzu automatických deployment (hooky) s prozkoumáním možností spojení deploymentu s Unit testy. Tento kontrolní mechanismus by měl být schopen při neúspěšném merge nebo build informovat plnohodnotně uživatele o nastalé chybě. Součástí práce je i stručná analýza možností jiných nástrojů pro automatický deployment. V závěru práce autor provede srovnání těchto nástrojů s verzovacím systémem GIT. Výstupem praktické části je pak návrh a implementace kompletního řešení pro automatický deployment se systémem kontroly alespoň pro dva odlišné servery např. produkční a vývojový. V rámci praktické části by bylo vhodné postihnout i další problematiku – např. automatické spouštění migračních skriptů pro databázi (pokud jsou), případně jiných potřebných migračních skriptů na serveru s operačním systémem Linux.

Rozsah pracovní zprávy: **min. 30 s.,dop. rozsah 40 s.**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná**

Seznam doporučené literatury:

GitHub, https://github.com/progit/progit2/releases/download/2.1.360/progit.pdf
GitLab, https://about.gitlab.com/
Joost Evertse, Mastering GitLab 12: Implement DevOps culture and repository management solutions Paperback, Packt Publishing, 2019, ISBN 978-1789531282.
Adam O'Grady, *GitLab Quick Start Guide: Migrate to GitLab for all your repository management solutions*, Packt Publishing, 2018, ISBN 978-1789534344
SOMASUNDARAM, Ravishankar. *Git: Version Control for Everyone Beginner's Guide*. Packt Pub., 2013., http://kim.marcelandkim.com/git/Git_-_Version_Control_for_Everyone.pdf

Vedoucí bakalářské práce: **Ing. Monika Borkovcová, Ph.D.**
Katedra informačních technologií

Datum zadání bakalářské práce: **16. prosince 2022**
Termín odevzdání bakalářské práce: **12. května 2023**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

Ing. Jan Panuš, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 28. února 2023

Prohlašuji:

Práci s názvem „Využití verzovacího systému GIT pro správu projektu a automatický deployment se systémem kontroly“ jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 7/2019 Pravidla pro odevzdávání, zveřejňování a formální úpravu závěrečných prací, ve znění pozdějších dodatků, bude práce zveřejněna prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 26. 03. 2023

Jakub Joukl

PODĚKOVÁNÍ

Děkuji vedoucí mé bakalářské práce, Ing. Monice Borkovcové, Ph.D., za vstřícný přístup, veškerou poskytnutou pomoc, rady, připomínky a veškeré poskytnuté konzultace, které mi pomohly dokončit tuto bakalářskou práci.

ANOTACE

Cílem práce je zpracovat podrobnou analýzu využití verzovacího systému GIT při tvorbě softwarového systému, možností využití systému GIT v teamových projektech a analýzu automatických deployment (hooky) s prozkoumáním možností spojení deploymentu s Unit testy. Tento kontrolní mechanismus by měl být schopen při neúspěšném merge nebo build informovat plnohodnotně uživatele o nastalé chybě. Součástí práce je i stručná analýza možností jiných nástrojů pro automatický deployment. V závěru práce autor provede srovnání těchto nástrojů s verzovacím systémem GIT. Výstupem praktické části je pak návrh a implementace kompletního řešení pro automatický deployment se systémem kontroly alespoň pro dva odlišné servery např. produkční a vývojový. V praktické části je postihnuta i problematika spouštění automatických migračních skriptů pro databázi.

KLÍČOVÁ SLOVA

Verzovací systém, Git, GitLab, CI, CD, DevOps, vývoj software, hooks, databáze, automatické databázové migrační skripty

TITLE

Using GIT for project management and automatic deployment with control system

ANNOTATION

The goal of this thesis is to conduct a detailed analysis of the usage of the GIT version control system in software development, its usage in team projects and, in addition, analysis of automatic deployment (hooks) with system to conduct automatic deployment with tests. This control mechanism should fully inform the user of any errors that may have occured. This thesis also provides a brief analysis of other automatic deployment tools. At the end of the theoretical part, the author compares these tools with the version control system GIT. The practical part of this thesis aims to design and implement a solution for automatic deployment system with a mechanism to check for errors on at least two different servers, such as production and development. This part also incorporates a solution for creating and executing automatic database migration scripts.

KEYWORDS

Version control system, Git, GitLab, CI, CD, DevOps, software development, hooks, databases, automatic database migration scripts

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM ZKRATEK A ZNAČEK

CI – continuous integration

CD – continuous delivery/deployment (dle definice)

DevOps – Development Operations

RUP – Rational Unified Process

DSDM – Dynamic System Development Method

XP – extrémní programování

SDLC – Software Development Life Cycle

DVCS – distribuované systému

CVCS – centrálně řízené systémy

DB – databáze

GUI – grafické uživatelské rozhraní

ÚVOD

V oblasti profesionálního komerčního programování ve firmách vzrůstá během vývoje komplexnost vyvíjeného softwaru a nutnost spolupráce s ostatními vývojáři. Z důvodu narůstající složitosti vyvíjeného softwaru je nutné celý proces vývoje softwaru v profesionálním prostředí formálním způsobem organizovat a určit nutné kroky v jeho vývoji. Kromě formalizace a uspořádání celého procesu vývoje další nutnou podmínkou k úspěšnému a efektivnímu řízení softwarových projektů bývá použití vhodných nástrojů zajišťujících hlídání změn a jejich autorů s možností vrátit se zpět ke staré verzi. Pro co největší zefektivnění průběhu celého vývoje softwaru i s jeho nasazením se v praxi také hojně využívají nástroje z oblasti CI/CD zajišťující kontrolu provedených změn a jejich aplikování do existujících systémů.

Cílem této práce je popsat základní prostředky, principy, nástroje a postupy využívané při spolupráci v týmech při vedení rozsáhlých komerčních projektů a aplikovat získané znalosti při návrhu fiktivního projektu s využitím verzovacího systému Git.

Teoretická část této práce se nejprve zabývá prostředky nutnými k organizaci a formalizaci postupu vývoje softwaru, mezi které patří stanovení vhodných metodik vývoje softwaru, popisu fází jeho životního cyklu, testování softwaru a jeho části. Po objasnění těchto pojmů následuje základní popis verzovacích systémů a popis verzovacího systémů Git. Po této části práce pokračuje vysvětlením pojmů z oblasti CI/CD, využití nástrojů CI/CD, přehledem nejčastěji používaných nástrojů z této oblasti a jejich popisem a srovnáním. Po popisu nástrojů využívaných pro vedení projektu navazuje popis DevOps, tato oblast se věnuje sloučení použití nástrojů pro vedení projektu s teoretickými principy a postupy vedení projektu.

Praktická část se soustředí na vytvoření a naplnění scénáře popisujícího proces návrhu, vývoje testování a nasazení jednoduchého projektu v jazyce C# s využitím poznatků z teoretické části a s využitím nástroje GitLab. Součástí této části práce je i využití Git hooks a jejich popis a využití automatických databázových migračních skriptů.

1. VÝVOJ SOFTWARU

1.1 Metodiky vývoje

Metodiky vývoje softwaru se dělí na tradiční a agilní a jejich hlavní priorita spočívá v organizování, ucelení vedení projektu, popisu procesů a postupů a uskutečnění nutných kroků v návrhu a vývoji projektu. Hlavním cílem zavedení metodik vývoje softwaru je snaha zvýšit úspěšnost softwarových projektů a jejich dokončení.

1.1.1. Tradiční metodiky vývoje

Tradiční metodiky vývoje začaly vznikat v 60. letech minulého století kvůli potřebě vnést řád do postupu vedení, návrhu a vývoje softwarových projektů, do té doby se v projektech žádná ucelená metodika nepoužívala.

Tyto metodiky se nejprve zabývají naplánováním postupu vývoje celého softwaru bez ohledu na možné případné komplikace, které během vývoje softwaru mohou nastat. Velkou roli při jejich používání hraje sjednocení používání vývojových technologií, prostředí, nástrojů, procesů vývoje, pracovních postupů atd.

Strategií používanou v tradičních metodikách vývoje softwaru je, že tým odpovědný za vedení projektu cílí na realizaci veškerých požadavků zákazníka na začátku vývoje softwaru, dopředné naplánování veškerých významných mezních termínů, které se během vývoje softwaru vyskytují, tradiční metodiky také kladou velký důraz na rozsáhlou projektovou dokumentaci, která se vytváří jak před vývojem softwaru, tak i během jeho vývoje a po dokončení vývoje softwaru. Projektová dokumentace zachycuje veškeré požadavky zákazníka na vyvíjený software, obsahuje informace o veškerých naplánovaných termínech, zda byly dodrženy či zda se zpozdily, popisuje proces vývoje software, použité technologie při vývoji softwaru a použité technologie tvořící daný vyvinutý software.

Výsledkem výše uvedeného popisu tradičních metodik je jejich jednoduchá předvídatelnost, lze tedy snadno naplánovat termíny, fáze vývoje, rozpočet projektu, velikost projektového týmu a jeho strukturu, obtížnost projektu.

Mezi základní modely používanými při vývoji softwaru pomocí těchto metodik vývoje softwaru patří například vodopádový model, spirálový model, RUP (Rational Unified Process). [1]

1.1.2. Agilní metodiky vývoje

Agilní metodiky vývoje softwaru začaly vznikat v polovině devadesátých let minulého století jako protipól tradičních metodik vývoje softwaru.

Hlavními prioritami agilních metodik jsou [2]:

1. Reagování na změny – během vývoje softwarového projektu mohou vzniknout změny v požadavcích kladených na výsledek projektu, proto je zbytečné plánovat jednotlivé kroky v úvodu projektu do hloubky, plánování vyvíjených částí softwarového projektu probíhá až když to je potřeba.
2. Cílení na výslednou hodnotu – během vývoje je výsledek důležitější než zbytečná stavová dokumentace, za tímto účelem se více komunikuje se všemi účastníky vývoje (i se zákazníkem a uživateli systému) za účelem zefektivnění vývoje a zkvalitnění výsledného produktu.
3. Spolupráce – agilní metodiky vývoje softwaru preferují úzce spolupracující skupinu vývojářů na návrhu specifikací výsledného softwaru před návrhem těchto specifikací jedním expertem kvůli lepšímu povědomí projektového týmu o veškerých vlastnostech a procesech ve vyvíjeném softwarovém projektu.
4. Důraz na členy týmu – klade se důraz na disciplínu, vzájemnou důvěru a týmového ducha všech členů týmu.

Cíle agilních metodik byly shrnuty v Manifestu Agilního vývoje vytvořeného konsorciem DSDM (Dynamic System Development Method) v roce 2001 do následujících bodů [2]:

1. Jednotlivci a interakce před procesy a nástroji.
2. Funkční projekt před rozsáhlou dokumentací.
3. Spolupráce se zákazníkem před vyjednáváním o ceně.
4. Reakce na změny před dodržováním pevně stanoveného plánu.

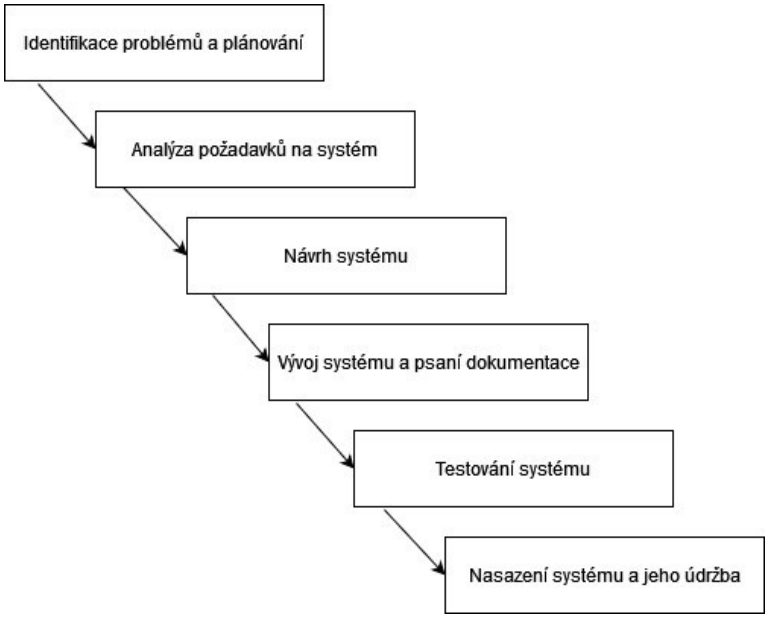
Mezi základní modely používanými při vývoji softwaru pomocí agilních metodik vývoje softwaru patří například XP (Extrémní programování), Scrum, SAFe (Scaled Agile Framework). [2]

1.2. SDLC – Životní cyklus informačního systému

Životní cyklus vývoje informačního systému definuje fáze vývoje softwarového projektu. Jedná se o popis životního cyklu od samého začátku, fáze návrhu, až do ukončení používání vyvinutého softwarového produktu.

Životního cyklus informačního systému a jeho rozfázování se používá jako pomůcka pro plánování celého vývoje softwarového produktu, jednotlivé etapy na sebe navazují, etapy je možné opakovat. Veškeré jeho části by měly být pokryty v rámci zvolené metodiky vývoje softwaru.

Existuje více dělení životního cyklu vývoje informačního systému. V této práci bylo zvoleno dělení na 6 fází dle Ing. Pavla Jiravy, Ph.D. [3]. Jedná se o fáze identifikace problémů a plánování, analýza požadavků na systém, návrh systému, vývoj systému a psaní dokumentace, testování systému, nasazení systému a jeho údržba.



Obrázek 1 - dělení SDLC na 6 fází, přepracováno dle [3]

1.2.1. Identifikace problémů a plánování

Identifikace problémů a plánování představuje první fázi v softwarových projektech. Tento vstup je velmi důležitý pro úspěch celého projektu, její zanedbání může vést k nezdárnému konci projektu. V této fázi probíhá sběr požadavků na software od klienta, analýzou podobných projektů, ze zkušeností a identifikovaných problémů. Na základě sesbíraných požadavků se provede sestavení plánu projektu.

1.2.2. Analýza požadavků na systém

V další fázi se analyzují požadavky na systém získané v etapě identifikace problémů a plánování. Na základě sesbíraných dat se provede analýza požadavků a z této analýzy se provede návrh informačního systému či jeho podsystému. Zanalyzuje se také stávající informační systém klienta, existuje-li nějaký, a podobné informační systémy.

1.2.3. Návrh systému

V této etapě se provede návrh informačního systému na základě analýzy provedené ve fázi analýzy požadavků na systém. Z analyzovaných požadavků se navrhne uživatelské rozhraní, databázový systém, struktura souborů, vhodné technologie, ve kterých bude systém realizovaný, postupy a procesy používané v systému, návrh autorizace a autentifikace uživatelů systému. Je také nutné navrhnout postupy a procedury v případě selhání systému.

1.2.4. Vývoj systému a psaní dokumentace

Čtvrtá fáze vývoje informačního systému zahrnuje na základě provedené analýzy v kroku návrhu systému provedení naprogramování částí systému nebo celého systému. Veškeré vyvinuté části se vhodně zdokumentují, součástí dokumentace by měla být část jak programátorská, tedy popis implementace a vnitřní struktury systému, tak i část uživatelská, což znamená, jakým způsobem dané naprogramované části systému bude uživatel systému využívat.

1.2.5. Testování systému

V pátém kroku vývoje se dosud vytvořený systém testuje, a to jak manuálně s pomocí programátorů, zákazníka, analytiků, tak i za pomocí automatických testů. Systém se testuje nejprve s použitím sady testovacích dat a poté i s použitím sady produkčních dat.

1.2.6. Nasazení systému a jeho údržba

Zpravidla poslední fáze životního cyklu softwaru, je nejdelší a nejnákladnější fází vývoje, dle některých odhadů vyjde až na 80 % celkových nákladů projektu. Během této etapy se musí uživatelé systému seznámit se systémem a naučit se ho používat.

Dle Ing. Pavla Jiravy, Ph.D. existuje více způsobů nasazení systému [3]:

1. Přímá záměna – Starý systém se zamění za nový během pár dní. Rizikem tohoto způsobu implementace je fakt, že během testování se nemusely odhalit veškeré problémy vyvíjeného informačního systému nebo softwaru.
2. Paralelní změna – Starý a nový systém se používá současně, uživatelé systému postupně přecházejí na nový systém. Rizikem tohoto způsobu přechodu je nebezpečí, že uživatelé budou pokračovat v používání starého systému namísto nového.
3. Pilotní testování – Systém se nasadí omezeně, například do jedné pobočky či oddělení, po kladném pilotním testování je systém připraven na plošné nasazení.

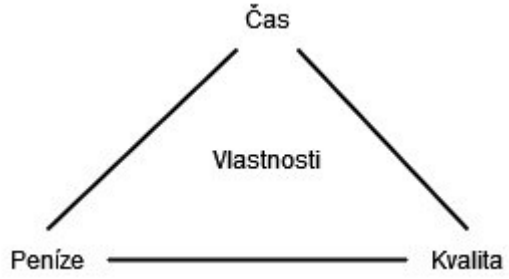
[3]

1.3. Testování

Testování je nedílnou součástí cyklu návrhu a vývoje softwaru a informačního systému, přičemž jeho nedostatečné či neúplné testování může mít za následek jeho selhání. Pokud je software chybně otestovaný, tak to může mít za následek finanční škodu a ztrátu reputace pro zákazníka, poškození životního prostředí, poranění či smrt lidí a zvířat, na straně návrháře a vývojáře daného informačního systému pak může dojít ke ztrátě reputace a může mu vzniknout finanční škoda. Cílem testování je odhalit co největší množství problémů a chyb softwaru, zdokumentovat dané chyby a jejich dopady, zdokumentovat průběh testů, dané chyby je poté nutné nahlásit k opravě vývojářům, samotná oprava chyb se v rámci testování neprovádí.

1.3.1. Rizika spojená s testováním

Testování softwaru by tedy mělo být dostatečně kvalitní a dlouhé k odhalení co největšího množství chyb v softwaru a informačním systému vedoucí k jejich odstranění. V praxi se však velice často stává, že testováním není možné postihnout veškeré možné scénáře, které mohou při používání informačního systému nastat z důvodu velké časové, rozsahové a finanční náročnosti testů, pokud měly pokrýt veškeré možné případy, které by se v rámci používání systému mohly vyskytnout, a proto je nutné testovat informační systém či software a navrhnout testy takovým způsobem, aby bylo pokryto co největší množství nejčastějších a nejzávažnějších scénářů, které se v průběhu používaní informačního systému mohou nastat. Obecně také platí, že čím více vlastností vyvíjený informační systém obsahuje, tím je náročnost testování vyšší a je těžší navrhnout správné testy.



Obrázek 2 - trojúhelník problematiky testování IS a softwaru, přepracováno dle [4]

Vzhledem k tomu, že v testování není možné pokrýt veškeré možné případy, které v rámci používání softwaru nebo informačního systému mohou nastat, je nutné vybrat dostatečné množství testů dle možných důsledků, které mohou nastat v případě selhání či poruchy vyvinutého softwaru či informačního systému. To lze vidět na porovnání důsledků selhání

softwaru řídícího systému metra a selhání softwaru u výdejního automatu na nápoje. V případě selhání softwaru řídícího systému metra může dojít k vysokým finančním škodám, zraněním a ztrátám na životech, v případě selhání softwaru u výdejního automatu na nápoje může dojít pouze k finanční škodě.

Aby bylo možné určit, jaké množství testů je dostatečné, je nutné zvolit vhodná kritéria, při jejíchž splnění považujeme počet testů za dostatečný a testy za dostatečně kvalitní. V rámci těchto kritérií se vyberou sledované stavy a procesy, které mohou v rámci běhu softwaru nebo v rámci používání informačního systému nastat, sledované objekty a entity, jaké chyby jsou v rámci testů závažné a jaké chyby mohou být díky své nízké závažnosti opraveny až během nasazení systému, množství finančních prostředků a času, které na testy chceme vynaložit, milník, do kterých chceme dokončit testování.

1.3.2. Etapy testování

Testování softwaru se dělí na fáze podle toho, kdo testy provádí, jakým způsobem se testy provádějí, jaký je celkový účel testů v dané fázi, a kdy se dané testy provádějí. Rozdělení testování do fází je důležité pro zajištění co nejvyšší kvality vyvinutého softwaru a obvykle se dělí do těchto fází [4]:

1. Jednotkové testování
2. Integrační testování
3. Systémové testování
4. Akceptační testování

1.3.2.1. Jednotkové testování

Předpokladem k efektivnímu jednotkovému testování softwaru je jeho rozdělení na jednotlivé části, pokud vyvíjený software není efektivně a logicky rozčleněn na jednotlivé části, výsledky jednotkových testů mohou být nepříznivě ovlivněny jinými částmi, které jsou v rámci našeho testování pro nás nezajímavé. Jednotlivými částmi, nad kterými se provádí jednotkové testování, jsou program, třída, metoda. Jednotkové testování je zpravidla první fází testování v testování softwaru, zaměřuje se na testování a debugging jednotlivých části vyvíjeného softwaru, v rámci debuggingu se mohou používat i různé debuggovací nástroje.

Jednotkové testování je prováděno vývojářem softwaru, obvykle tím, který danou část softwaru naprogramoval, a vyžaduje znalost kódu testované části softwaru. Cílem jednotkového testování je zajistit, aby výstupem vývoje jednotlivých části softwaru byl spustitelný kód a odhalení a korekce chyb v testovaných částech softwaru. V této fázi testování se obvykle

nevytváří žádná dokumentace obsahující informace o nalezených a případně i opravených problémech v testovaných částech softwaru a ani se nevytváří dokumentace popisující průběh jednotlivých testů.

1.3.2.2. Integrační testování

Integrační testování je etapa testování softwaru, při které se jednotlivé vyvinuté části softwaru spojí dohromady a testují se jako celek. Pro lepší návrh a průběh integračního testování je možné sestavit integrační diagram vizuálně zachycující komunikaci a interakci mezi jednotlivými částmi softwaru. Cílem integračního testování je odhalit chyby vzniklé v softwaru či v nedodržování průběhu a pořadí interakcí při komunikaci a interakcemi mezi jednotlivými částmi softwaru. V rámci integračního testování se obvykle vytváří dokumentace popisující průběh testů a nalezené chyby.

Integrační testování sebou nese rizika spojená s nekompatibilitou systémů jednotlivých částí, meziplatformní komunikací, problémy spojené se změnou funkční logiky jednotlivých částí a problém s dopadem této změny na ostatní části softwaru. V rámci integračního testování se také mohou zbytečně testovat případy, které během reálného provozu softwaru nemohou nastat a mohou být opomenuty nebo upozaděny případy, které se v reálném provozu budou vyskytovat často.

Pro zahájení integračního testování softwaru je nutné si nejprve zvolit integrační strategii, podle které budeme během integračního testování postupovat. V rámci této práce jsou zde uvedeny 3 časté strategie testování [4]:

1. Integrace Big-Bang
2. Integrace odshora dolů
3. Integrace zespoda nahoru

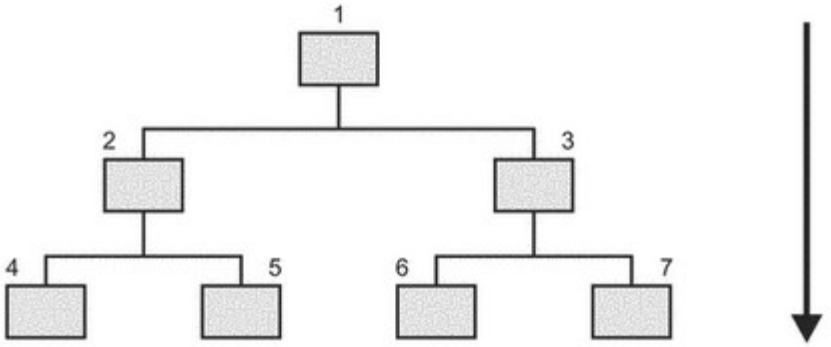
1.3.2.2.1. Integrace Big-Bang

Při použití Big-Bang integrace se veškeré části systému integrují všechny naráz a tím okamžitě vznikne celý systém. Nevýhodou této strategie je, že na otestování některých částí systému se může zapomenout a z toho důvodu mohou být problémy odhaleny pozdě a tím pádem bude jejich oprava složitější a nákladnější.

1.3.2.2.2. Integrace odshora dolů

V rámci této integrační strategie se systém spojuje po fázích, začíná se spojovat od částí, které jsou v hierarchii nahoře, to jsou ty části, které volají jiné podřízené části. V rámci jednotlivých

fází se komponenty v hierarchii níže, které nebyly ještě do testovaného systému integrovány, nahradí provizorními částmi, pahýly, které simulují logiku komponent pod nimi. Nevýhodou této strategie je psaní velkého množství pahýlů.



Obrázek 3 - integrace odshora dolů [4]

1.3.2.2.3. Integrace odspoda nahoru

Při použití strategie odspoda nahoru se systém spojuje po částech, začíná se spojovat od částí, které jsou v hierarchii dole, to jsou ty části, které jsou volány jinými částmi. V rámci jednotlivých fází se komponenty v hierarchii výše, které nebyly ještě do testovaného systému integrovány, nahradí speciálně napsanými komponentami, ovladači, které řídí už integrované komponenty v hierarchii pod nimi. Nevýhodou této strategie je psaní poměrně velkého množství složitých ovladačů. [4]

1.3.2.3. Systémové testování

Systémové testování je fáze testování softwaru, kde se prověřuje správnost fungování celého softwaru jako celku z pohledu zákazníka a jeho scénářů používání vyvinutého softwaru. Tato etapa testování je jednou z nejdůležitějších etap testování softwaru na straně jeho vývojářů, vývojářům slouží jako výstupní kontrola správnosti fungování softwaru před předáním softwaru zákazníkovi. Rozsah tohoto kroku testování je variabilní dle množství testování provedeného v předchozích krocích testování. V rámci systémového testování se testují reálné případy, které mohou nastat během provozu softwaru. Výsledky testování a průběh testů se obvykle dokumentují.

Toto testování se obvykle provádí ve více fázích a obvykle ho provádí jiný tým než ten tým, který daný software vyvíjel. Během systémového testování se testují funkční a nefunkční požadavky na systém. V rámci testování funkčních požadavků se testuje, zda software má veškeré požadované funkcionality, které má mít a zda fungují správně. V průběhu testování

nefunkčních požadavků se testují obecné požadavky na software, které nemusejí souviset s funkcemi, které daný software nabízí. [4] [5]

Mezi nefunkční požadavky patří [4]:

1. Instalace – procedury na zavedení softwaru
2. Udržovatelnost – možnost provádět změny v softwaru
3. Výkon – požadavek na správné chování systému
4. Zvládání zátěže – chování systému s rostoucí zátěží
5. Zvládání hraniční zátěže – chování systému s hraničními hodnotami zátěže
6. Přenositelnost – použitelnost na jiných platformách
7. Zotavení z chyb – soubor procedur na obnovení softwaru do správného stavu v případě nastalé chyby
8. Spolehlivost – schopnost softwaru fungovat s narůstajícím provozním časem
9. Použitelnost – jednoduchost používání softwaru uživatelem

1.3.2.4. Akceptační testování

Tato fáze testování je prováděna zákazníkem. Akceptační testování je prováděno nezávisle na předchozích provedených fázích testování, je prováděno podle sestavených scénářů sestavených s použitím softwarové dokumentace. Nalezené chyby a případné nesrovnalosti mezi specifikací softwaru a jeho implementací v rámci této fáze testování jsou hlášeny vývojářům pro co nejrychlejší opravu, případná pomalá oprava může mít za následek zpoždění termínu nasazení softwaru do provozu. [5]

1.4. Nasazení

Během této fáze životního cyklu informačního systému probíhají 3 hlavní aktivity, které jsou na sebe pevně navázané, jedná se o doručení softwaru, podporu a zpětnou vazbu. V rámci podpory je zákazníkovi poskytnuta dokumentace a pomoc s využíváním softwaru. V rámci zpětné vazby se zpracují podněty získané od zákazníka a použijí se během dalšího budoucího vývoje aplikace.

Ve fázi nasazení je vhodné se řídit následujícími principy, definovanými v knize Software engineering: a practitioner's approach, 7nd edition [7]:

1. Musíme se starat o očekávání zákazníka – zákazníkovi nesmíme předkládat konfliktní tvrzení o stavu vývoje softwaru a nesmíme mu slibovat nerealistické termíny jeho vývoje.

2. Dodaný balíček se softwarem a dokumentací by měl být sestavený a otestovaný – veškeré materiály potřebné k instalaci a užívání softwaru by měly být pořádně otestované na veškerých prostředích, kde se budou používat.
3. Před dodáním softwaru musí být zajištěna dostatečná podpora – musí být předem sestavený tým, který zajistí podporu zákazníkovi v případě toho, že by nastaly problémy během používání softwaru, veškeré problémy se musí zaznamenávat do dokumentace pro budoucí použití v rámci oprav problémů, uživatelům systému musí být poskytnuty veškeré manuály a návody pro práci se systémem.
4. Koncovým uživatelům musí být zajištěny veškeré potřebné materiály – Uživatelům systému musí být poskytnuty veškeré materiály, co se v systému od poslední verze systému změnilo a tyto funckionality jím musí být ukázány.
5. Před nasazením softwaru se musí opravit veškeré chyby – Uživatelům softwaru by se neměl dodávat software obsahující chyby s příslibem, že se opraví později – tento způsob dodávání softwaru snižuje důvěru zákazníka ve výslednou kvalitu vyvíjeného produktu. Veškeré odhalené chyby v softwaru se musí i za cenu pozdějšího nasazení před nasazením opravit.

Dodržování těchto principů vede k lepšímu průběhu nasazování systému, k vyšší spokojenosti zákazníka, k vyšší kvalitě vyvíjeného softwaru a včasnějšímu odhalení a opravě chyb, které se během této fáze odhalí. [7]

2. VERZOVACÍ SYSTÉMY

2.1. Základní principy verzování

Verzovací systémy umožňují vývojářům efektivně sdílet své změny s ostatními vývojáři, nasdílet softwarový projekt a jeho strukturu nově příchozím vývojářům, také jim umožňují vracet se k verzím souborů, které byly vytvořeny v minulosti, a dohledat autory těchto změn, napsat komentáře k provedeným změnám. Vzhledem k tomu, že během vývoje softwaru se kód a soubory neustále mění a v souvislosti s těmito změnami přibývají a ubývají nové soubory, tak bez použití systému na správu a hlídání těchto změn by se organizace a vedení projektu staly velkou překážkou pro efektivitu, a rychlost vývoje a také by se v důsledku nutnosti přidání další manuální činnosti do činností vedoucích k vývoji softwaru přidal prostor pro vzniku chyb v důsledku možného omylného smazání souboru vývojářem bez možnosti obnovení, manuálního slučování souborů a složek, hlídání změn v souborech a organizační struktuře složek a projektu zvětšila chybovost softwarového projektu. Systém pro správu a verzování také registruje, kdo a kdy provedl jaké změny a v jakých souborech a složkách. V případě, že by se vývojářský tým rozhodl, že se chce vrátit ke starší verzi projektu nebo jen určitých souborů a při vedení softwarového projektu by se nepoužíval žádný systém, který by prováděl organizaci a správu změn v softwarovém projektu, tak by bylo nutné buď přejít k poslední existující záloze projektu nebo by bylo nutné pro určení poslední platné verze souborů zkontrolovat čas změny souborů. [8]

„*Základní myšlenkou verzování je oddělení souborů, na kterých pracují programátoři, zvaných pracovní kopie, od hlavních kopií stejných souborů uložených v repozitáři. Soubory v repozitáři programátoři přímým způsobem nikdy neupravují*“ [1]. [6]

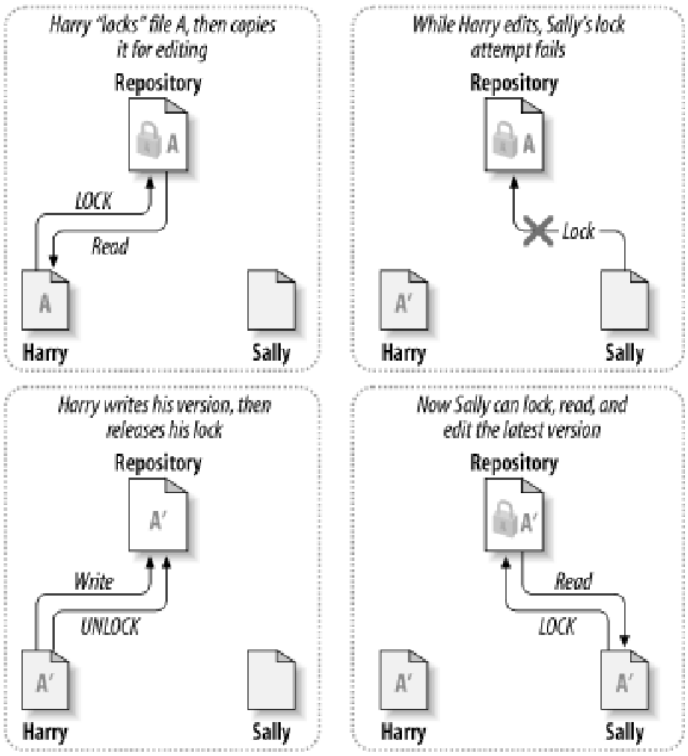
Verzování systémů probíhá takovým způsobem, že při poslání souboru do vzdáleného repozitáře, se v repozitáři vytvoří nová verze poslaného souboru. Při posílání dalších verzí určitého souboru se do repozitáře ukládají pouze změny mezi po sobě jdoucími soubory, díky tomuto mechanismu nedochází ke zbytečnému plýtvání místa na disku v repozitáři. [6]

Základním problémem, který při používání systémů pro správu verzí může nastat, je problém současného přístupu ke stejné verzi téhož souboru 2 a více programátory. Problém nastává

[1] The basic VC idea is to separate the files that programmers work on, called working copies, from the master copies of the same files, which are stored in a repository. The programmers never work on the repository.

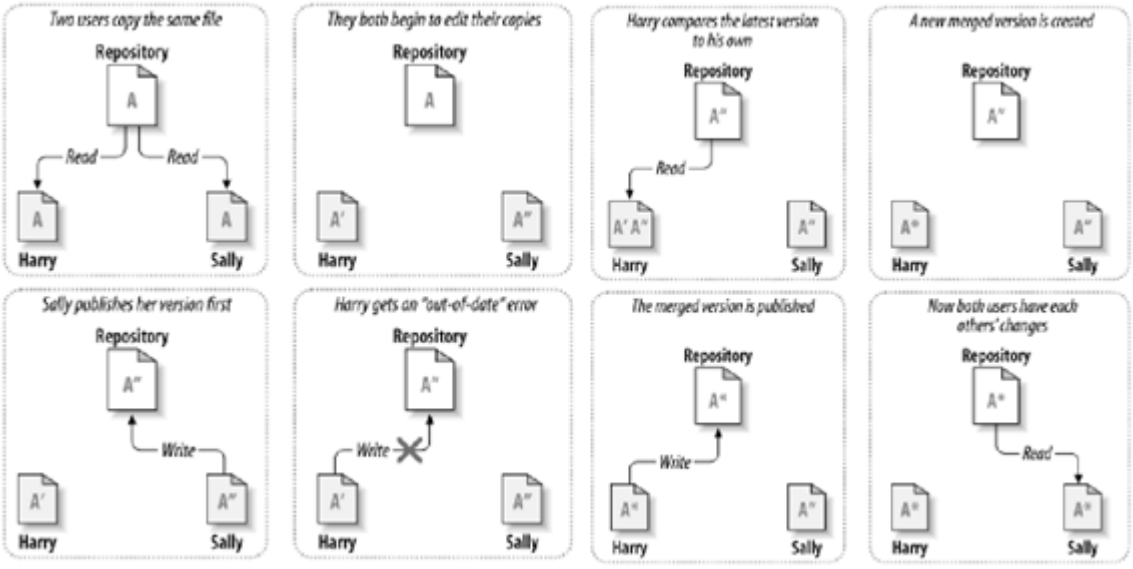
v případě, že by více programátorů chtělo měnit obsah souboru současně, tento problém lze řešit dvěma způsoby [9]:

1. Zamknutím souboru, modifikací a následným odemčením – vývojář si vyžádá na upravovaný soubor zámek, po dokončení úprav zámek odstraní a soubor odemkne, během doby trvání zámku lze soubor ostatními pouze prohlížet. Nevýhodou tohoto přístupu je fakt, že soubor nemohou upravovat dva vývojáři současně, výhodou je, že nemůžou nastat konflikty ve změnách souboru vzniklé současnou úpravou souboru více vývojáři.



Obrázek 4 - zamknutí souboru pro úpravy [9]

2. Vytvoření kopie souboru, změna, sloučení – upravovaný soubor se nijak nezamyká, vývojáři si své lokální kopie souborů mohou upravovat dle libosti, při pokusu o nahrání souboru na vzdálený repozitář si systém řízení kontroly zkontroluje, zda již nedošlo k úpravě souboru na serveru, pokud ano, systém řízení kontroly na to vývojáře, který se pokouší nahrát svoji verzi, upozorní, a daný vývojář musí vyřešit sloučení svých úprav s úpravami v repozitáři.



Obrázek 5 - průběh změn při modifikaci stejného souboru [9]

Existují dva různé způsoby implementace systému na správu verzí, které se odlišují způsobem své implementace: distribuované systému (DVCS) a centrálně řízené systémy (CVCS). [8]

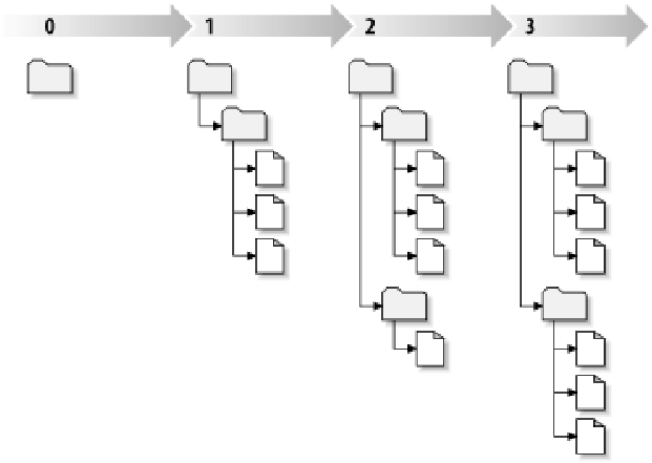
2.2. Centralizované systémy

Projekt a veškeré jeho soubory se nachází na jednom místě – v jednom centrálním repozitáři, který se nachází na centrálním serveru. Veškeré informace o verzování souborů, historii verzovacího stromu jsou pouze na jednom místě – centrálním repozitáři. Pro veškeré operace se soubory a strukturou projektu musí být vývojář připojen k centrálnímu repozitáři. Pokud chce vývojář změnit soubor, tak si k sobě z centrálního repozitáře nahraje jeho lokální kopii, v které provádí změny, po dokončení úprav svoji lokální kopii sloučí se soubory v centrálním repozitáři. [8]

2.2.1. Apache Subversion

Subversion je jedním z centralizovaných verzovacích systémů, který umožňuje pracovat se soubory v repozitáři přes síť. Tento verzovací systém nepodporuje SCM, to tedy znamená, že nativně nerozumí zdrojovému kódu ani nepodporuje operace přímo spojené se zdrojovým kódem. Díky tomu, že Subversion využívá systém „vytvoření kopie souboru, změna, sloučení“, tak se veškeré provedené změny nahrají na centrální repozitář až dokončení úprav. Při každém nasdílení změn do centrálního repozitáře platí pro zajištění integrity ve stromu změn a v samotném verzování v tomto verzovacím systému pravidlo, že veškeré změny se buď propíšou naráz nebo se nepropíše žádná ze změn. Sdílení změn do repozitáře je atomickou operací. Subversion umožňuje dohledávání změn pomocí čísla revizí – jedná se o celé číslo,

které označuje dané nasdílené změny, číslování začíná od 0 (prázdný adresář). Každé změně, poté, co je nasdílena do centrálního repozitáře, je přiděleno unikátní číslo revize, podle které jde dohledat v rámci celého stromu revizí. Tato přidělená čísla nejsou unikátní pro jednotlivé soubory, ty žádné osobní číslo změny nemají, ale označují celou revizi. [9]



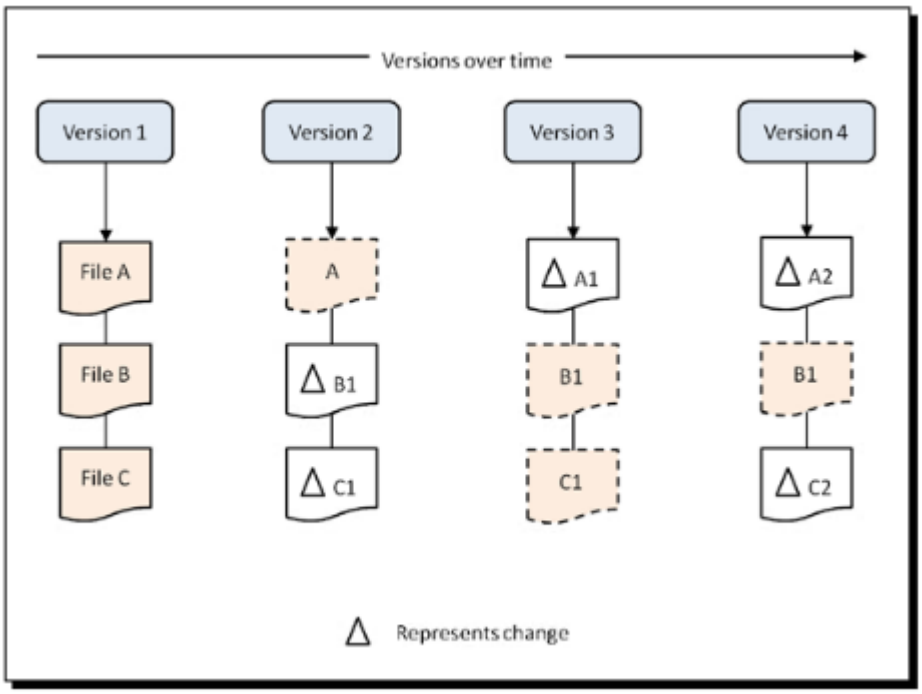
Obrázek 6 - strom změn Subversion [9]

2.3. Distribuované systémy

Projekt, historie změn a veškeré soubory se nachází jak na vzdáleném serveru, tak i na zařízení každého vývojáře. Díky této vlastnosti mohou vývojáři změny sdílet a synchronizovat mezi sebou bez nutnosti komunikace a synchronizace s centrálním repozitářem a v případě poruchy serveru, kde se nachází repozitář, nehrozí ztráta souborů projektu, vývojáři mohou pomocí svých lokálních historií změn a souborů ztracená data obnovit do původního stavu. [8]

2.2.3. Git

Git je jedním z distribuovaných verzovacích systémů, byl navržen Linusem Torvaldsem. Hlavní motivací pro vznik Gitu byla potřeba navrhnout verzovací systém, který by kombinoval tyto 3 vlastnosti v jednom systému: atomicitu, výkon, bezpečnost. Atomicita v pojetí Gitu znamená, že pro zamezení ztráty dat při nasdílení změn ostatním vývojářům nebo do vzdáleného repozitáře se změny nasdílí buď všechny nebo se žádná ze změn v případě chyby sdílení neaplikuje. Díky svému specifickému způsobu verzování souborů a organizace verzovací historie a výkon představuje jednu z největších výhod Gitu oproti konkurenčním softwarům. Git historii souboru neeviduje tím způsobem, že by pro každý soubor vytvářel jeho individuální historii změn, ale historizuje pomocí snímků stavu všech souborů. Vždy se vytvoří snímek změn všech souborů a stavu souborů, snímek stavu souborů se vytvoří i pro ty nezměněné, ale u nezměněných souborů se místo zapsání změn do historie zapíše reference na předchozí snímek.



Obrázek 7 - verzování v Gitu [10]

Tento přístup verzování v Gitu přináší další výhodu – informace o verzích zabírají málo místa na úložišti. Bezpečnost představuje pro Git také jednu z priorit. Aby se zabránilo riziku, že někdo, ať se jedná o útočníka, záměr vývojáře nebo jen jeho neopatrnost, záměrně upraví historii změn v Gitu, tak z veškerých změny v historii verzí souborů, které se v Gitu vytváří, se vytvoří kontrolní součet a ten se poté používá k odkazování na konkrétní změny v souboru v historii změn. [10]

2.4. Srovnání centralizovaných a distribuovaných VCS

Srovnání DVCS a CVCS se nachází v následující tabulce.

Tabulka 1 - srovnání CVCS a DVCS [8]

Systém na správu verzí	CVCS	DVCS
Repozitář	Existuje pouze jeden repozitář – server	Každý uživatel má svůj lokální repozitář na svém počítači
Přístup k repozitáři	Každý uživatel, který chce přistupovat do repozitáře musí být připojen k síti	Uživatelé mohou vyvíjet i offline, pro sdílení svých změn s ostatními uživateli se musí připojit k síti

Příklad nástrojů daného systému na správu verzí	Subversion, Perforce Revision Control System	Git, Mercurial, Bazaar, BitKeeper
Příklad vhodných charakteristik vyvíjeného softwarového projektu	1. Projekty, které umožňují přispívat k vývoji softwaru jen omezenému množství uživatelů 2. Vývojářský tým se nachází na jedné pobočce	1. Díky tomu, že repozitář s projektem je distribuovaný mezi více vývojářů, tak výhody DVCS jsou vítaným vylepšením pro vývoj projektu 2. Díky usnadnění sdílení práce je vhodný jak pro malé, tak i rozsáhlé projekty 3. Tým, který se nachází na více pobočkách nebo zemích a různých časových pásmech

[8]

3. CI/CD

3.1. Popis CI/CD

Zkratka CI/CD slučuje metody z CI (Continuous integration) a CD (dle definice buď Continuous delivery nebo Continuous deployment). CI/CD slouží k zautomatizování kroků, které nastávají během vývoje ve fázích kompilace kódu, testování softwaru a slučování kódu s kódem v repozitáři a nasazováním kódu na zvolené prostředí, v rámci CI/CD nástrojů je také často vidět, v jaké fázi CI/CD cyklu nám cyklus selhal.

Obrázek 8 - CI/CD fáze a jejich spojení [11]

CI nám pomáhá řešit problémy spojené s vývojem kódu na větvích a slučování větví kódu. V případě rozsáhlých projektů se často nezávisle na sobě vyvíjí a upravuje mnoho různých vlastností na různých nezávislých větvích kódu, ale po sloučení změn z těchto různých větví se může stát, že vyvíjený software nebude správně fungovat nebo dokonce nepůjde ani spustit. Tuto problematiku slučování větví CI řeší tím, že po sloučení změn se u těchto změn se pomocí automatického spuštění testů ověří, že tyto změny nenarušily současné fungování softwaru, pokud změny neprojdou testy, tak tato chyba skutečnost je vývojářům oznámena.

„*CD může znamenat buď Continuous delivery nebo Continuous deployment*.“[2] [11].

CD ve významu Continuous delivery usnadňuje práci vývojářům tím, že kód úspěšně validovaný z fáze CI se automaticky, bez nutnosti dalšího manuálního zásahu, sloučí s kódem, který se nachází v repozitáři.

CD ve významu Continuous deployment vykoná to samé, co Continuous delivery, ale software se na zvoleném prostředí spustí s kódem, který byl v tomto kroku nahrán do repozitáře. Slabinou tohoto přístupu je, že je nutné napsat rozsáhlé a kvalitní testy, jinak se nám může stát, že na prostředí se dostane i nefunkční kód. [11]

[2] The "CD" in CI/CD refers to continuous delivery and/or continuous deployment

3.2. Nástroje CI/CD

Vzhledem k tomu, že principy CI/CD zefektivňují, zkvalitňují a automatizují proces vývoje softwaru, tak stále více vývojářských společností začíná s procesem integrace nástrojů, které CI/CD pravidla realizují, do svých projektů a procesů vývoje. Díky rostoucí popularitě nástrojů pro CI/CD v komerční sféře v době psaní této bakalářské práce již existuje velké množství takovýchto nástrojů, v této bakalářské práci jsou popsány nejčastěji používané. [12]

3.2.1. Jenkins

Jenkins je v době psaní bakalářské práce jedním z nejrozšířenějších a nejpoužívanějších open-source integračních nástrojů poskytující veškeré výhody, které CI/CD principy nabízejí. Z důvodu, že je open-source, tak jeho používání je bezplatné. [12] [14]

Vnitřní struktura programu Jenkins je rozčleněná na několik různých částí, tyto části spolu spolupracují a jsou navzájem provázané. Těmito částmi jsou [13]:

1. Jenkins Controller – organizuje ostatní Jenkins uzly (Jenkins Node) s nainstalovaným Jenkins agentem a řídí jejich práci, obsahuje a spravuje veškeré globální informace, jako jsou například pluginy.
2. Jenkins Agent – Jenkins agent je uzel, který spouští tzv. sestavení v rámci projektů.
3. Jenkins Node – Buď Controller nebo Node.
4. Jenkins Project – Soubor více sestavení definovaný uživatelem.
5. Jenkins Plugins – komunitně vyvíjené vlastnosti pro Jenkins, které Jenkins server v základní konfiguraci neobsahuje.
6. Jenkins Pipeline – soubor kroků pro opravdovou podporu CI/CD, za pomocí pluginů lze dosáhnout [13]:
    - 6.1. Automatických sestavení
    - 6.2. Vícekrokového testování
    - 6.3. Procedur nasazení softwaru
    - 6.4. Bezpečnostní kontroly kódu

3.2.1.1. Výhody a nevýhody Jenkins

Mezi hlavní výhody Jenkins patří [13][14]:

1. Veliká rozšiřitelnost funkcí pomocí pluginů.
2. Škálovatelnost serveru zajištěná pomocí distribuovaných Jenkins Node.
3. Multiplatformnost – Windows, Linux a macOS.

4. Z důvodu rozšíření existuje mnoho výukových materiálů a rozsáhlá dokumentace.
5. Je zdarma a open source.
6. Po nainstalování pomocí instalátoru lze hned spustit.
7. Podpora autentizace a rolí.

Hlavními nedostatky Jenkins jsou [13]:

1. Architektura jednoho serveru – z důvodu, že Jenkins nepodporuje přímou spolupráci dvou více serverů jako jeden, tak je nutné mít dostatečně výkonné zařízení pro hostování serveru s dostatečným množstvím operační paměti.
2. Rozlehlost serverů – díky omezení na architekturu jednoho serveru je často nutné pro pokrytí veškerých potřeb organizace a týmů vytvářet více na sobě nezávislých serverů.
3. Závislost na starších technologiích Javy – z důvodu, že v době, kdy vznikal Jenkins, neexistovaly moderní frameworky jako je Spring a V GraalVM, tak je Jenkins silně závislý na technologii Maven a Java Servlet.
4. Bez nativní podpory pro kontejnery.
5. Vzhledem k tomu, že se pro nasazování píšou skripty, tak vznikají obtíže s psaním tohoto kódu a jeho debuggingem.
6. Vzhledem k množství pluginů může být pro nezkušené uživatele vybrat si vhodné a prověřené pluginy.

3.2.2. Gitlab

Gitlab je dalším z často používaných nástrojů v z důvodu potřeby zefektivnění práce pomocí metodik obsažených v CI/CD přístupu. Kromě samotného rozhraní pro podporu CI/CD operací Gitlab obsahuje i vlastní rozhraní pro plánování management vývojářských týmů, přidělování práce jednotlivým vývojářům, zapojení zákazníka do procesu vývoje za pomocí systému s rolemi přístupu na vytváření zjištěných defektů ve vyvíjeném softwaru a přehled provedených změn v kódu, které, jak je patrné z názvu, jsou spravovány pomocí distribuovaného verzovacího systému Git. [15]

3.2.2.1. Výhody nástroje Gitlab

Výhody nástroje Gitlab se dají shrnout do následujících bodů [12][14]:

1. Celý projekt a jeho zdrojové kódy je možné přehledně spravovat pomocí přehledného webového rozhraní s funkcemi podporujícími spolupráci zákazníků a vývojářských týmů.

2. Nativní integrace verzovacího systému Git a CI/CD řešení umožňuje vývojářům zrychlit nasazování softwaru, zrychlit vydávání nových verzí softwaru a včasné odhalování chyb v softwaru.
3. Poskytuje zvýšenou bezpečnost a integritu pro zdrojové kódy softwaru.
4. GitLab realizuje API, pomocí kterých mohou vývojáři volat jeho funkce a mohou tak lépe integrovat GitLab a jeho prvky do svých produktů využívajících GitLab.
5. Multiplatformnost – mezi podporované platformy se řadí Windows, Linux a macOS.
6. Optimalizace procesu CI/CD – jednotlivá softwarová sestavení je možné sestavovat paralelně na více zařízeních.
7. Jednotlivá sestavení se mohou sestavovat efektivně paralelně – díky optimalizaci cachingu jednotlivá paralelně běžící sestavení mohou přistupovat ke stejným informacím a nemusí plýtvat čas zbytečným duplicitním čtením informace.
8. Podpora autentizace a rolí.

3.2.3. TeamCity

TeamCity je dalším z mnoha existujících CI/CD nástrojů, je vyvíjen společností JetBrain, jeho jádro je naprogramováno v jazyce Java, součástí vlastností zajišťující efektivní funkci TeamCity jako CI/CD nástroje je přehledné webové rozhraní s přehledem životního cyklu jednotlivých fází životního cyklu CI/CD. Veškeré vlastnosti nabízené touto aplikací lze rozšiřovat pomocí volně dostupných pluginů. [16] [17]

3.2.3.1. Výhody TeamCity

Jako hlavní výhody využívání TeamCity lze uvést [16][12][14]:

1. Vysoká škálovatelnost pomocí pluginů.
2. Udržuje historii nasazení a obsahuje funkce podporující detekci změn, které mohou způsobovat problémy s úspěšným sestavením.
3. Nativní podpora pro Docker, Maven a další podobné programy.
4. Podpora autentizace a rolí.
5. Obsahuje zabudované funkce, pomocí kterých lze sledovat průběh sestavení a vidět případné problémy se sestavením.
6. Multiplatformnost – podpora Linux a Windows.
7. Podpora API funkcí a z ní vycházející možnost zakomponování jednotlivých funkcionalit do vlastní aplikace.
8. Podpora Cloudových funkcí.

3.2.4. Bamboo

Bamboo je dalším z v praxi hojně využívaných nástrojů s vlastním grafickým webovým rozhraním pro CI/CD, byl vyvinut společností Atlassian, který pro účely CI/CD využívá architekturu centrálního serveru, hostovaným na serverech Apache Tomcat, s agenty pro jednotlivé dílčí činnosti jako je například spuštění sestavení. V jeho webovém rozhraní lze nalézt informace o průběhu sestavení a o případných nastalých chybách během sestavení. [18][19]

3.2.4.1. Výhody Bamboo

Mezi hlavní důvody, proč používat Bamboo, patří [12][14][19]:

1. Možnost paralelního spouštění více různých sestavení a možnost provádění testů paralelně.
2. Podpora rolí a práv umožňující nastavit práva k omezení prostředí, kde vývojáři mohou spouštět testy a sestavení.
3. Multiplatformnost – hlavní server může být hostován na operačních systémech Linux a Windows, agenti mohou být spuštěni na operačních systémech Linux, Windows a macOS.
4. Rozšiřitelnost pluginy.

3.2.5. Circle CI

Circle CI se řadí mezi hojně využívané služby v rámci uplatnění metodiky CI/CD. Tento nástroj nabízí jak placené, tak i neplacené tarify, které se liší různými parametry, jako je například počet souběžně běžících jobů, poskytovanou podporou. Circle CI v sobě v rámci běhu testů obsahuje zabudovanou validaci, zda testy a sestavení proběhly správně, na vzniklé chyby je schopen upozornit jak pomocí informace v GUI své webové aplikace, tak i prostřednictvím notifikací a emailů.[20]

3.2.5.1. Výhody Circle CI

Hlavními výhodami, kterými Circle CI oplývá, jsou [12][14]:

1. Multiplatformnost – podpora platforem Windows, Linux, a macOS.
2. Integrovaná podpora pro analýzu kódu většiny programovacích jazyků.
3. Testy lze spustit samostatně – z toho vyplývá možná vysoká rychlost spouštění testů.
4. Podpora pro debugging díky integrované podpoře analýzy kódu.
5. Podpora pro paralelismus při spouštění sestavení a testů.
6. Nativní podpora pro verzovací systém Git.

3.3. Srovnání nástrojů CI/CD

V navazující tabulce jsou srovnány vybrané vlastnosti nástrojů CI/CD uvedené v této kapitole:

Tabulka 2 - srovnání nástrojů CI/CD [13][28]

	Jenkins	TeamCity	CircleCI	Bamboo	GitLab
Open Source	Ano	Ne	Ne	Ne	Ne
Zabudované vlastnosti	3/5	4/5	4/5	4/5	4/5
Integrace	5/5	3/5	4/5	3/5	4/5
Hostování	Vlastní prostory a Cloud	Vlastní prostory a Cloud	Vlastní prostory	Vlastní prostory, Bitbucket v Cloudu	Vlastní prostory a Cloud
Verze zdarma	Ano	Ano	Ano	Ano	Ano
Cena agenta pro sestavení	Zdarma	Od $59 měsíčně	Od $15 měsíčně	Od $1200 ročně	Od $19 za měsíčně za uživatele
Podporované Operační systémy	Windows, Linux, macOS	Linux, macOS	Windows, Linux, macOS, Solaris, FreeBSD a další	Windows, Linux, macOS, Solaris	Následující Linuxové distribuce: Ubuntu, Debian, CentOS, Oracle Linux

[13]

4. DEVOPS

4.1. Charakteristika problémů vedoucích k DevOps

S narůstající rozsáhlostí a náročností softwarových projektů a rostoucí velikostí projektových týmů se začínají projevovat problémy koordinace mezi jednotlivými týmy. Díky pokroku v oblasti vedení a plánování softwarových projektů a vývoji legislativy a strategií týkající se informační bezpečnosti a nakládání s informacemi začalo být zjevné, že dosavadní přístup ke komunikaci a koordinaci spolupráce mezi týmy, vedení projektů a způsob sestavování a nasazování projektů u rozsáhlejších projektů a týmů přestává být dostatečně efektivní. Z tohoto důvodu bylo nutné se touto problematikou začít zabývat, jako řešení, které má dané problémy vyřešit, je zavedení principů, pravidel a filozofie, jakým způsobem organizovat vývoj a spolupráci v projektech, jaká pravidla vývoje projektu používat, jaké postupy během vývoje používat, jaké procesy by měly v řízení a vývoji projektu probíhat – DevOps (Development Operations).

DevOps lze také definovat jako firemní kulturu vývoje – dodržování těchto pravidel pro vývoj může vylepšit vztahy na pracovišti a důvěru mezi jednotlivými účastníky vývoje. [22]

„Termín DevOps se stal předmětem výzkumu poté, co ho Patrick Debois představil na konferenci s názvem „Agile Infrastructure and Operations“, která se konala v roce 2008.“[3] [22]

4.2. Hlavní cíle DevOps

DevOps se pokouší vyřešit řadu problémů, které se vyskytují v rámci organizace a vedení rozsáhlých softwarových projektů.

V oblasti komunikace a spolupráce se jedná zejména o problémy plynoucí z rozdílných cílů jednotlivých týmů účastnících se vývoje projektu, a to i v případě, že se tyto rozdílné týmy účastní vývoje stejné části softwaru. To je například patrné z příkladu týmu manažerů a týmu vývojářů – tým vývojářů, pracující na určité nové vlastnosti softwaru chce mít vlastnosti co nejlépe otestované a prověřené, hlavní prioritou týmu business je co nejvíce uspokojit zákazníka a vnutit mu co nejvíce požadavků za účelem vygenerování co největšího možného zisku pro společnost, toho se pokouší docílit návrhem co největšího množství požadavků a vlastností softwaru. V této oblasti DevOps cílí na to, aby tým vývojářů chápal a respektoval

[3] The term DevOps started to be researched after Patrick Debois introduced it at a conference entitled “Agile Infrastructure and Operations” in 2008.

cíle business týmu a aby tým business rozuměl a respektoval problémy a přání týmů vývojářů. Výsledkem tohoto aspektu DevOps vedení vývoje je lepší spolupráce mezi týmy a se zákazníkem, a vyšší kvalita, vyšší stabilita a rychlejší vývoj výsledného produktu. Důraz na spolupráci mezi týmy a členy týmu také zvyšuje motivaci a odhodlání všech členů vývoje projekt zdárně dokončit a prohlubuje vzájemnou důvěru mezi jednotlivými týmy a jejich členy.

Další výhoda, kterou implementace poznatků DevOps přináší, plyne z nutnosti v rámci implementace této filozofie a metod vývoje implementace poznatků z oblasti CI/CD. Využití poznatků z oblasti CI/CD v praxi napomáhá k návrhu automatických testů a jejich provádění, tím se dosáhne včasného odhalení chyb v softwaru a to vede k jejich rychlejšímu a efektivnějšímu odstranění, což snižuje náklady spojené s návrhem testů a jejich prováděním a ve výsledku je tímto způsobem zajištěno zrychlení vývoje, zvýšení stability softwaru a snížení jeho ceny jak pro nás, tak pro zákazníka, tím se v konečném důsledku zvýší i spokojenost zákazníka, poznatky CI/CD se využívají také v oblasti CD, v závislostí na uplatněné definici CD se buď provede automatické nasazení nebo jen automatické sloučení kódu do zvoleného repozitáře, poznatky z CD vedou k rychlejšímu nasazení změn. Další nespornou výhodou, vycházející z včasného odhalení chyb, je vyšší kvalita kódu. Nutnost návrhu a psaní testů také ve vedlejším důsledku vývojáře lépe seznamuje s vyvíjeným kódem a jednotlivými komponentami aplikace.

Dalším benefitem, který využívání principů DevOps přináší, zejména díky zvýšené spoluprácí mezi všemi členy procesu, která je popsána výše, je lepší vymezení plánu a cílů pro všechny vyvíjející a aktivní členy vývojového procesu a realističtější vytyčení milníků a termínů vývoje. [22]

Z principů této kultury také vyplývá jasná hierarchie a nutnost jejího respektování – nadřízení by se měli respektovat a úkoly od nadřízených by se měly plnit a jejich rady by se měly využít při řešení nastalých situací běžných během celého procesu vývoje [21]

Z důvodu standardizace postupů, principů a pravidel vývoje využívání DevOps snižuje celkový počet chyb ve výsledném produktu. [22]

Neméně důležitou výhodou, kterou tento způsob vedení projektu přináší, se nachází v oblasti řízení zdrojových kódu projektu a sdílení práce mezi jednotlivými účastníky vývoje, kteří potřebují přístup ke kódu. Tuto problematiku řeší zavedení verzovacích systémů do procesu vývoje, často se jedná o systém Git a další distribuované verzovací systémy. [21]

4.3. Životní cyklus DevOps

Životní cyklus vývoje DevOps se skládá s cyklicky opakujících se fází, každá z etap se soustředí na řešení odlišných aspektů vývoje, etapy na sebe navazují. [21]

Mezi jednotlivé fáze životního cyklu se většinou uvádí těchto 7 fází [21]:

1. Plánování – v této fázi se definují požadavky a časový plán vývoje
2. Continuous Delivery – Pomocí aplikace vhodných postupů se dosáhne automatických sloučení změn do hlavního repozitáře projektu a tím se urychlí proces vývoje, tato fáze je více popsána v CI/CD části této bakalářské práce.
3. Continuous Deployment – Za pomocí sady automatických testů, které zajišťují a ověřují správnost kódu, se docílí automatických nasazení kódu do produkce, tato fáze je více popsána v CI/CD části této bakalářské práce.
4. Continuous Development – vývojáři začnou spolupracovat na vývoji vlastností a modulů naplánovaných ve fázi plánování.
5. Continuous Testing – Vyvinuté změny nebo celá aplikace je manuálně a automaticky otestována na případné chyby.
6. Continuous Integration – v této fázi se využijí výsledky z fází nasazení, testování a poté se výsledky těchto fází využijí pro odhalení vzniklých problémů a chyb a návrh další iterace těchto fází.
7. Continuous monitoring – Tato část životního cyklu se zabývá pozorováním chování softwaru a na základě poznatků získaných z pozorování se vyhodnotí, zda se software chová korektně a jaké části softwaru je nutné opravit, dovyvinout nebo změnit, pro správný monitoring je nezbytně porovnávat chování a vlastnosti systému s naplánovanými a realizovanými požadavky.

5. NÁVRH A IMPLEMENTACE PROJEKTU S VYUŽITÍM VERZOVACÍHO SYSTÉMU GIT

5.1. Hardwarové a softwarové předpoklady

Pro vytvoření konfigurace dle postupu uvedeného v této práci byl využit počítač s operačním systémem Windows 10 Pro, verze 21H2, build 19044.2604, operační paměť 16 GB. Předpokládá se dostatečná volná disková kapacita pro programy potřebné pro provoz v práci dále uvedeného projektu, a na tomto operačním systému byla provedena instalace Oracle VM VirtualBox, verze 7.0.6 r155176 (Qt5.15.2), Docker Desktop 4.16.3 (96739) a aktivní internetové připojení.

5.2. Git hooks

Distribuovaný verzovací systém Git umožňuje řízení procesu sdílení úprav s ostatními vývojáři využitím systému Git hooks, které se spouští automaticky po vykonání určitých, v názvu hooku specifikovaných, akcích, jedná se tedy o skripty, nativními jazyky těchto skriptů jsou shell a Python, změny je možné těmito skripty i odmítnout. Podle strany, na které jsou dané hooks určeny k vykonávání, se dělí na dvě kategorie: hooks na straně klienta a hooks na straně serveru. [25]

5.2.1. Git Hooks na straně klienta

Hooks na straně klienta se spouští z lokální kopie repozitáře Git ze složky .git/hooks, která se nachází v kořenovém adresáři verzovaného projektu, skripty obsažené v těchto hooks není možné v základním nastavení nijak automaticky sdílet s ostatními vývojáři a je možné je libovolně měnit, na tyto skripty se z těchto důvodů není vhodné spoléhat jako na prostředky striktně zajišťující konzistenci nahraných změn, typu nahraných souborů a formát nahraných zpráv vztahujících se ke změnám. Ve výše zmíněném adresáři se nachází příklady možných hooks na straně klienta, které lze modifikovat a odstraněním koncovky .sample i využít pro řízení kontroly prováděných akcí při zpracovávání projektu nebo je možné vytvořit nové soubory, které mají název kromě koncovky .sample stejný jako typ zamýšleného hook. Pomocí nastavení v souboru .git/config je možné lokálně změnit výchozí složku, z které se hooks spouští. [25]

Nejdůležitějšími hooks na straně klienta a jejich nejčastějším použitím jsou [26]:

1. Pre-commit – Použití tohoto hook umožňuje provést základní automatickou kontrolu vytvořených změn a spustit skripty před pokusem o nasdílení vytvořených změn a v případě, že změny nevyhovují standardům, je možné je odmítnout.
2. Prepare-commit-msg – Tento hook umožňuje vygenerovat základní šablonu zprávy na základě sdílených změn nebo určitých pravidel.
3. Commit-msg – Užitím Commit-msg hook lze zkontrolovat připravenou zprávu a v případě nevyhovující zprávy je možné změny odmítnout.
4. Post-commit – Tento hook slouží zejména k účelům notifikace, vykoná se po potvrzení a zkontrolování zprávy sdílených změn.
5. Post-checkout – Při přepnutí se z jedné branche na druhou je možné pomocí tohoto hook vymazat vygenerované soubory.
6. Pre-rebase – Tímto hook je možné kontrolovat, zda povolit sloučení více sdílených změn v historii sdílení a případně odmítnout takové změny.

5.2.2. Git Hooks na straně serveru

Hooks na straně serveru umožňují efektivně řídit, jaké změny a v jakém stavu chceme do repozitáře přijmout a jaké ne, případně nimi lze informovat uživatele nebo případně i jiné osoby ohledně různých chyb nastalých při interakci uživatele se vzdáleným repozitářem.

Mezi nejdůležitější hooks na straně serveru se řadí [26]:

1. Pre-receive – Po provedení příkazu git push vývojářem a před přijmutím změn vzdáleným repozitářem se zkontrolují příchozí změny a v případě, že nevyhovují nastaveným pravidlům, je změny možné odmítnout. Kontrola povolených akcí v rámci sdílení změn také bývá jednou z často využívaných technik v případě použití tohoto hooku. Pre-receive hook se vykoná na veškeré sdílené změny zároveň a odmítá veškeré změny, a to i ty, které patří na jinou větev a prošly automatickou kontrolou.
2. Update – Používá se podobně jako Pre-receive, ale vykonává se pro veškeré sdílené změny zvlášť.
3. Post-receive – Jelikož se vykoná až po úspěšném sdílení změn, tak slouží zejména k zaslání notifikací.

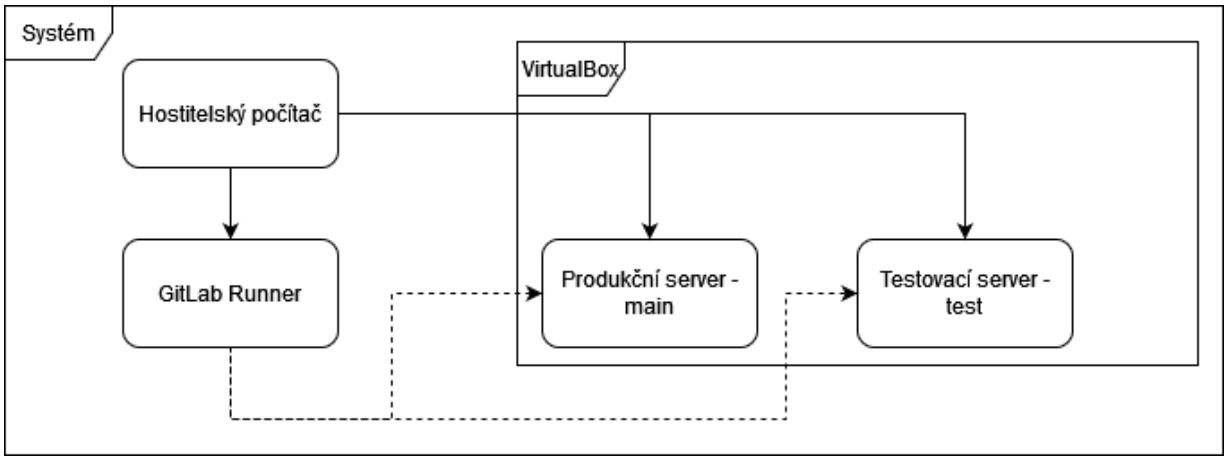
5.2.3. Konfigurace Git Hooks v projektu

V této bakalářské práci jsou použité hooks, které se nacházejí ve složce hooks, jsou stejné pro všechny přiložené verze projektu, je vhodné podotknout, že použití daných hooks není

v současné době podporováno všemi vývojovými prostředími, a proto případné sdílení změn přes tato vývojová prostředí nemusí fungovat.

5.3. Průvodní slovo k popisovanému projektu a jeho instalaci

Fiktivního zákazníka zajímá v současné době velice aktuální téma – umělou inteligenci, dotazy na ni a získání odpovědí na zadané dotazy, zvolenou umělou inteligencí je umělá inteligence volaná skrz API od OpenAI. Vzhledem k tomu, že klient chce multiplatformní aplikaci dostupnou co možná největšímu množství lidí různých věkových kategorií a s co možná nejmenšími nutnými obtížemi s uvedením aplikace do provozu, tak jedním z jeho požadavků je, aby aplikaci bylo možné spustit ve webovém prohlížeči a aby pro testování aplikace ze strany zákazníka existovalo testovací prostředí. V následujících odstavcích je uveden postup vývoje, testování a nasazení projektu s využitím agilní metodiky vývoje softwaru a s využitím verzovacího systému se servery hostovanými lokálně ve VirtualBoxu v místní síti s využitím GitLab Runneru počítači hostujícím VirtualBox. Plné čáry a šipky na následujícím obrázku představují vztahy mezi počítačem a servery, čárkované čáry a šipky mezi GitLab Runnerem a servery zobrazují směr primární interakce mezi těmito částmi.

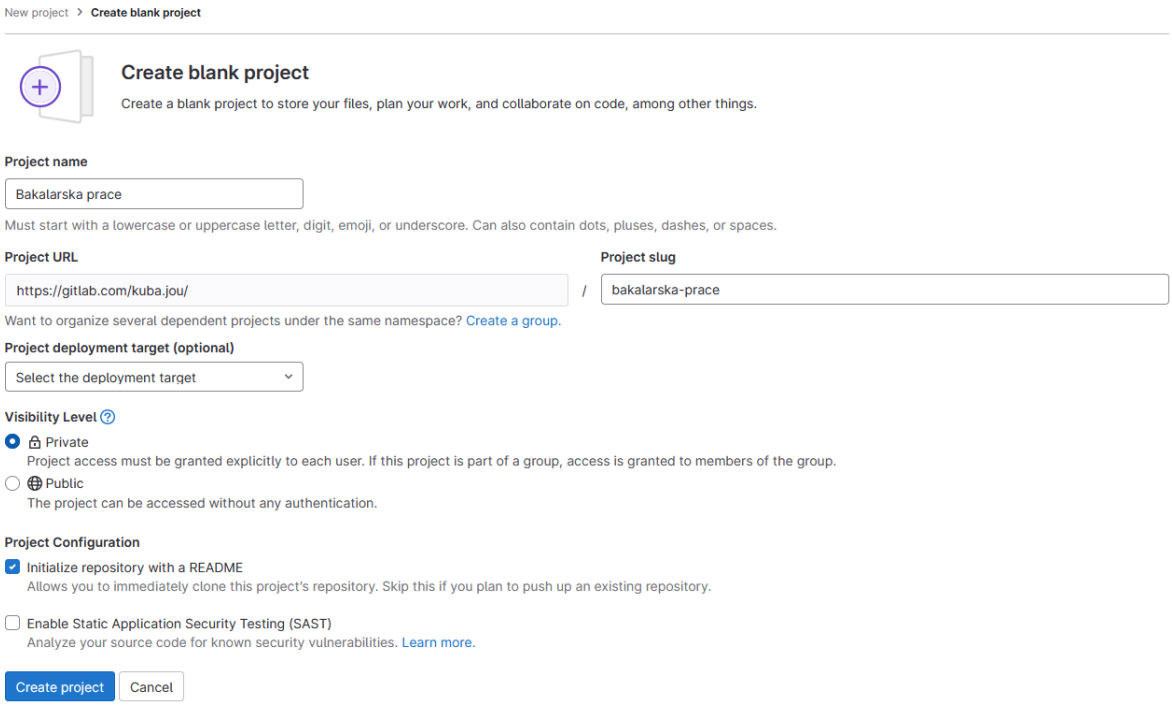


Obrázek 9 - architektura (zdroj vlastní)

5.4. Založení účtu GitLab, založení projektu a zřízení přístupu zákazníka

Po zpracování emailu zákazníka s žádostí o vývoj webového projektu s voláním API od Open AI bude komunikace se zákazníkem pokračovat skrze GitLab. Po dokončení registrace a na oficiálních stránkách GitLab https://gitlab.com/ a následném přihlášení se pokračuje založením projektu pro organizaci vývoje softwaru, CI/CD procesů a komunikace se zákazníkem. Projekt se založí pomocí modrého tlačítka „New Project“, případně z adresy https://gitlab.com/projects/new volbou Create blank project (odkaz https://gitlab.com/projects/new#blank_project). Po vyplnění pole „Project name“ textem

„Bakalarska prace“ následuje vytvoření projektu tlačítkem „Create project“. Obrazovka před zmáčknutím tlačítka „Create project“ vypadá následovně:



Obrázek 10 - nastavení projektu (zdroj vlastní)

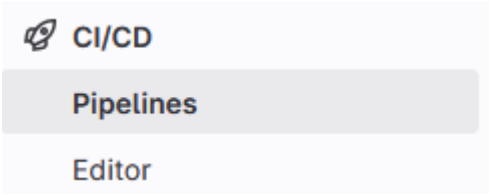
Po založení projektu se pokračuje přidáním zákazníka do projektu ze záložky **project information** > **members,** stisknutím tlačítka „Invite members“, po vyplnění emailu zákazníka a vybrání role „Reporter“ je po kliknutí na tlačítko „Invite“ zákazník přizván k projektu pro potřeby komunikace.

5.5. Systém kontroly stavu změn

Základní kontrolu, zda byly změny úspěšně nasdíleny, poskytuje verzovací systém Git, který je schopen pomocí zobrazené zprávy v konzoli nebo zvoleném vývojovém prostředí informovat o stavu nasdílení. Kromě této kontroly je také možné využitím další funkcionality tohoto verzovacího systému, Git hooks na straně serveru, kterými je možné například zaslat email informující uživatele, zda bylo či nebylo přijato nasdílení změn a proč se tak stalo, vykonávané akce vedoucí k informování by neměly být příliš komplexní a zdlouhavé. [27][26]

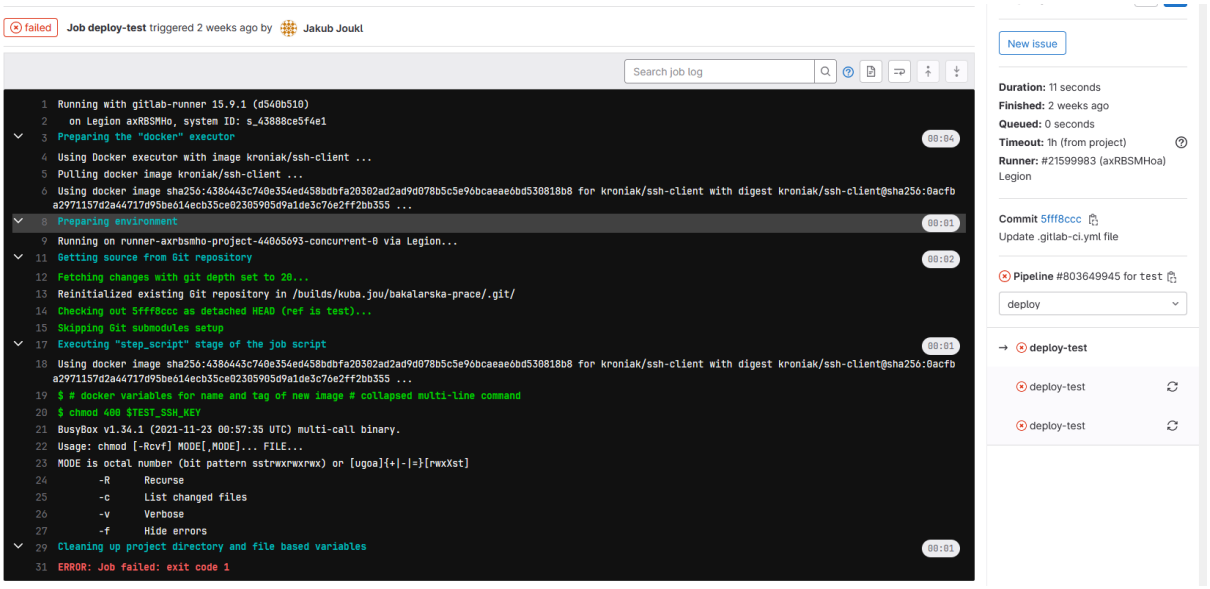
Dalším mechanismem, který slouží k informování uživatelů o stavu změn, jsou emailové notifikace, které posílá GitLab. Z oblasti notifikací ohledně stavu projektu GitLab vývojáře automaticky informuje zasláním emailu a ve svém systému v případě neúspěšného provedení CI/CD pipeline, v případě opravy dříve rozbité CI/CD pipeline, v případě požadavku na sloučení změn, jejich přijmutí nebo odmítnutí. Stav pipeline a požadavků na sloučení změn je

také možné sledovat na zabudovaných obrazovkách. Obrazovka obsahující informace ohledně provedených a prováděných pipeline se nachází u příslušného projektu v levém menu v záložce **CI/CD > pipelines**. [27]



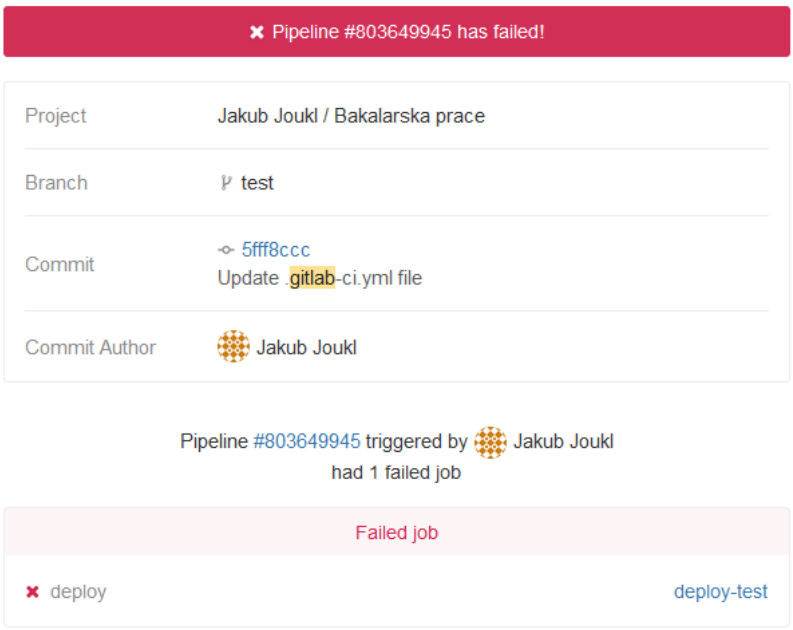
Obrázek 11 - CI/CD pipeline záložka (zdroj vlastní)

Na této obrazovce se nachází přehled všech vykonaných a probíhajících pipeline s informacemi, v jakém stavu se pipeline nachází nebo v jakém skončila, v případě selhání pipeline je označen krok, v kterém selhala, a informace o daném kroku, případně i informace ohledně jeho selhání, je možné si podrobněji prohlédnout po jeho rozkliknutí, selhaný krok, u kterého se nachází informace ohledně jeho trvání, nasdílení, datumu dokončení, větev a jakým Runnerem byl prováděn, vypadá následujícím způsobem:



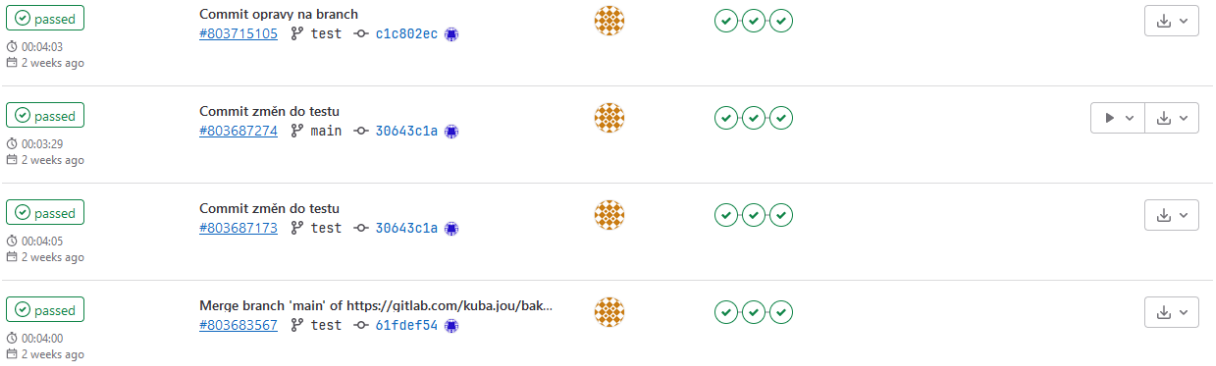
Obrázek 12 - selhaný krok (zdroj vlastní)

Emailová notifikace obsahuje, odeslaná v případě selhání či opravy pipeline, obsahuje informace ohledně nasdílení změn, které bylo odesíláno, větev, na které byla pipeline prováděna, projekt, informace o autorovi nasdílení změn a v případě selhané pipeline i krok, v kterém došlo k chybě.



Obrázek 13 - selhaná pipeline (zdroj vlastní)

Obrazovka s přehledem všech CI/CD pipeline je realizována tímto způsobem:



Obrázek 14 - obrazovka CI/CD pipeline (zdroj vlastní)

V případě odmítnutého nebo přijatého sloučení změn je informace zaslána emailem jako v případě vytvoření klasického issue a příslušná notifikace vypadá stejným způsobem jako u klasického issue, notifikace obdržené emailem obsahují informace o stavu požadavku na sloučení, případné komentáře a zda byl schválen/zamítnut.

V projektu jsou uskutečněny notifikace pomocí emailů prvky GitLabu na zasílání emailů a prvky z funkcí na sledování stavu pipeline z GitLabu.

5.6. Systém automatického nasazení

V této práci je řešen systém automatického nasazení řešen pomocí nastavení uvedeného v speciálním GitLab souboru .gitlab-ci.yml, potřebná kontrola vzniklá během sestavování,

nasazení a hlášení chyb, které mohou během nasazení nastat, je řešena v předcházející sekci. Tento soubor obsahuje nastavení pro spuštění automatických testů, sestavení a nasazení projektu v případě nasdílení změn na větev main nebo větev test.

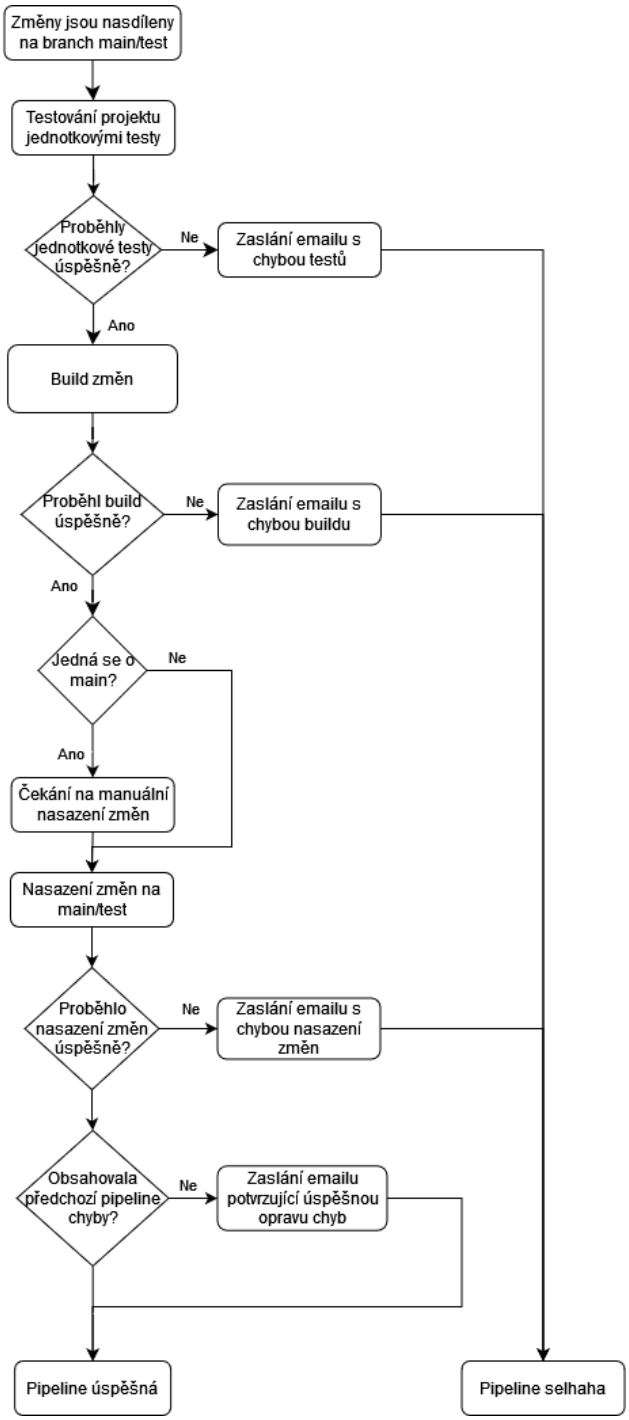
Obsah souboru .gitlab-ci.yml je uveden v příloze 1.

Nastavení .yml souboru obsahuje konfiguraci pro nastavení globálních proměnných zajišťujících nastavení jména a označení image před spuštěním samotného nasazovacího skriptu a pro 3 hlavní části pipeline: testy, sestavení a nasazení. První část, část zabývající se spuštěním testů, se spustí hned po nasdílení změn na větev main nebo test a provede se automatické spuštění a vyhodnocení jednotkových testů, pokud se jednotkové testy provedou úspěšně, pokračuje se následujícím krokem, krokem sestavení. Fáze sestavení se připojí k repozitáři Docker Image GitLabu a zde vytvoří a nasdílí novou Docker Image odpovídající aktuálnímu stavu repozitáře, pokud se vše zdaří, pokračuje se poslední fází, dle větve buď fází deploy pro main nebo fází deploy-test pro test. Tyto poslední kroky jsou obsahově téměř totožné, během těchto kroků probíhá přes protokol SSH nasazení kontejnerů s Docker Image na servery, tyto fáze se jen liší tím, že na větvi main je nutné tento krok manuálně potvrdit, pro test se provede automaticky. Po úspěšném provedení těchto kroků se změny nasadí na příslušný server.

Tři výše uvedené etapy průběhu nasazení lze shrnout do těchto bodů:

1. Testování změn jednotkovými testy. Pokud testy proběhnou v pořádku, pokračuje se sestavením změn, pokud selžou, zašle se email s informacemi o chybě během vykonávání testů.
2. Sestavení změn. Pokud sestavení změn uspěje, pokračuje se nasazením změn na prostředí, v opačném případě je zaslán email s popisem nastalého problému.
3. Nasazení změn. V případě, že nasazení neproběhne v pořádku, informace o chybě je zaslána emailem, pokud nasazení proběhne úspěšně a minulé nasazení selhalo, tak proběhne oznámení této skutečnosti vývojářům.

Graficky je průběh nasazení je zobrazen na následujícím obrázku:



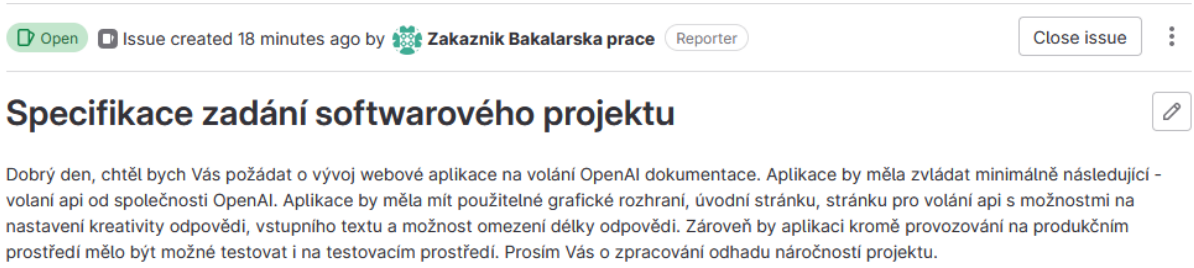
Obrázek 15 - průběh pipeline (zdroj vlastní)

Dalším důležitým souborem, který nezajišťuje průběh samotné pipeline, ale zajišťuje sestavení C# projektu do kontejneru, je Dockerfile, který obsahuje konfiguraci potřebnou k sestavení Docker kontejneru. Tento soubor je k dispozici v přiložených materiálech.

5.7. Zpracování projektu

5.7.1. Vstupní požadavky zákazníka

Úvodní požadavky klienta kladené na výsledný produkt jsou následující:

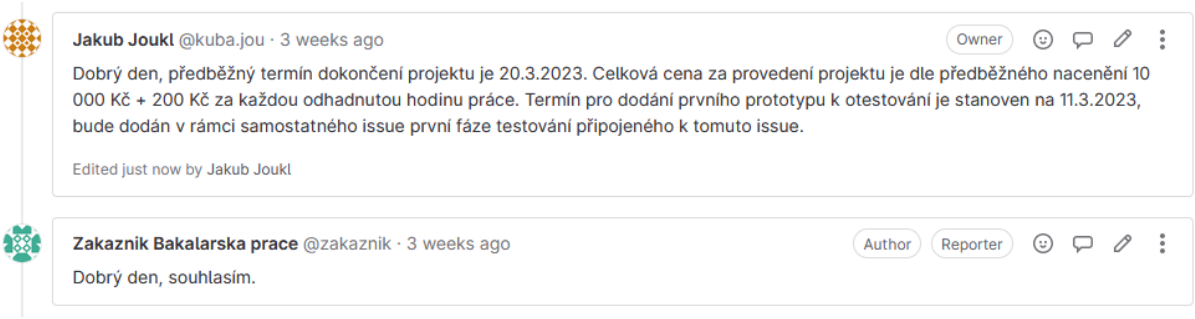


Obrázek 16 - úvodní zadání (zdroj vlastní)

Po založení issue se zadáním byl jako řešitel nastaven vývojář Jakub Joukl.

Uvedené požadavky je nejprve nutné naplánovat, nacenit a určit předběžné termíny celkového dokončení projektu, první fáze projektu. Jednotlivé další návazné dílčí části dle agilní metodiky vývoje softwaru není nutné plánovat dopředu.

Zapojení zákazníka do průběhu testování a komunikaci lze vidět na následujícím obrázku:



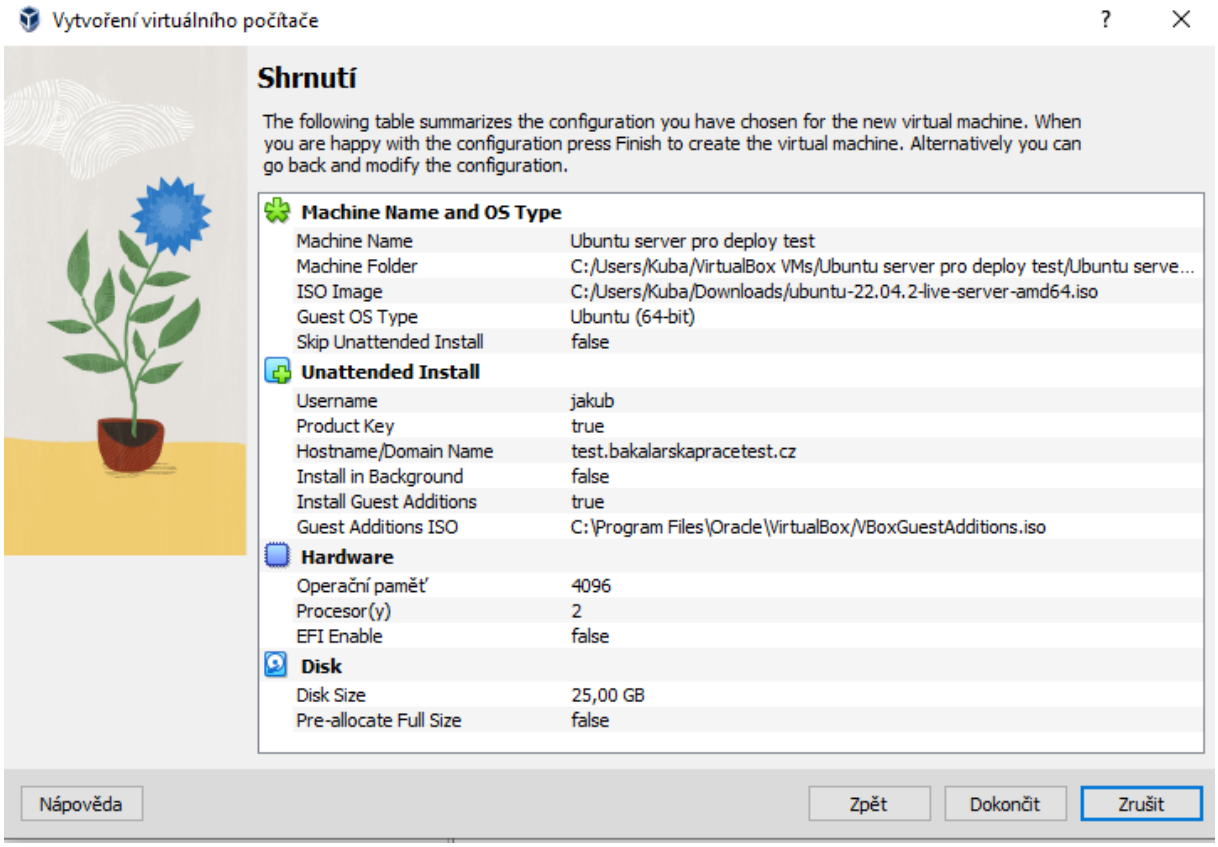
Obrázek 17 - průběh komunikace se zákazníkem (zdroj vlastní)

V návaznosti na issue týkající se veškerých projektů pro účely organizace je založeno issue týkající se prvního prototypu se zadáním pro vývojáře.

5.7.2. Založení prostředí pro zákazníka

Pro potřeby simulace procesu vývoje a nasazení softwaru v této bakalářské práci je nutné založit v GitLabu 2 hlavní vývojové větve pro spouštění 2 různých verzí softwaru na dvou různých serverech, produkčním a testovacím, jedná se o větev main a větev test. Prostředí použitá v této bakalářské práci budou běžet na serverech linuxové distribuce ubuntu 22.04.2. dostupné z https://cz.releases.ubuntu.com/releases/22.04.2/ubuntu-22.04.2-live-server-amd64.iso. Poté se vytvoří 2 virtuální stroje – produkční a testovací. Níže je popsána instalace testovacího serveru, produkční se vytvoří téměř identickým způsobem.

Ve VirtualBox se vytvoří nový počítač, vybere se image Ubuntu a složka, kde se server bude nacházet, volba „Skip Unattended instalation“ se odškrtne. Na následujících obrazovkách se nastaví uživatelské jméno a heslo na jakub, jméno hostitele na test a doménové jméno na bakalarskapracetest.cz., operační paměť RAM na 4096 MB, 2 procesory, nový disk o velikost 25 GB, výsledné nastavení vypadá následovně:

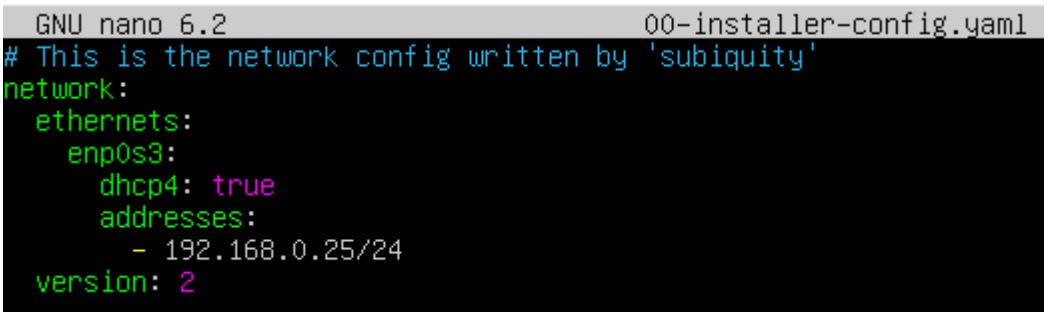


Obrázek 18 - nastavení serveru (zdroj vlastní)

Poté následuje instalace serveru pomocí instalačního průvodce. Na dalších obrazovkách se nejprve vybere jazyk „English“ poté se zvolí klávesnici a její rozložení „Czech“, poté dostačuje vybrat tlačítko „Done“ až do obrazovky s přehledem parametrů systému, na které se pokračuje pomocí tlačítka „Done“ a „Confirm destructive action“. Poté se konfiguruje uživatel – jméno Jakub, jméno serveru test (u master serveru je jméno master), uživatelské jméno jakub, heslo jakub, poté se následující 2 obrazovky dokončí pomocí tlačítka „Done“.

Po instalaci je nutné pomocí „Reboot now“ počítač restartovat. Poté následuje změna typu připojení počítače v nástroji VirtualBox na „síťový most“ z důvodu zajištění dostupnosti PC v síti. Pro zjištění IP adresy serveru počítače se nainstaluje nástroj ifconfig pomocí příkazu sudo apt install net-tools, a pomocí ifconfig se zjistí adresa serveru, kterou je nutné staticky přiřadit na interface serveru pomocí úpravy souboru 00-installer-config.yaml příkazem sudo

etc/netplan/00-installer-config.yaml, soubor je nutné upravit přidáním statické IP adresy následujícím způsobem:



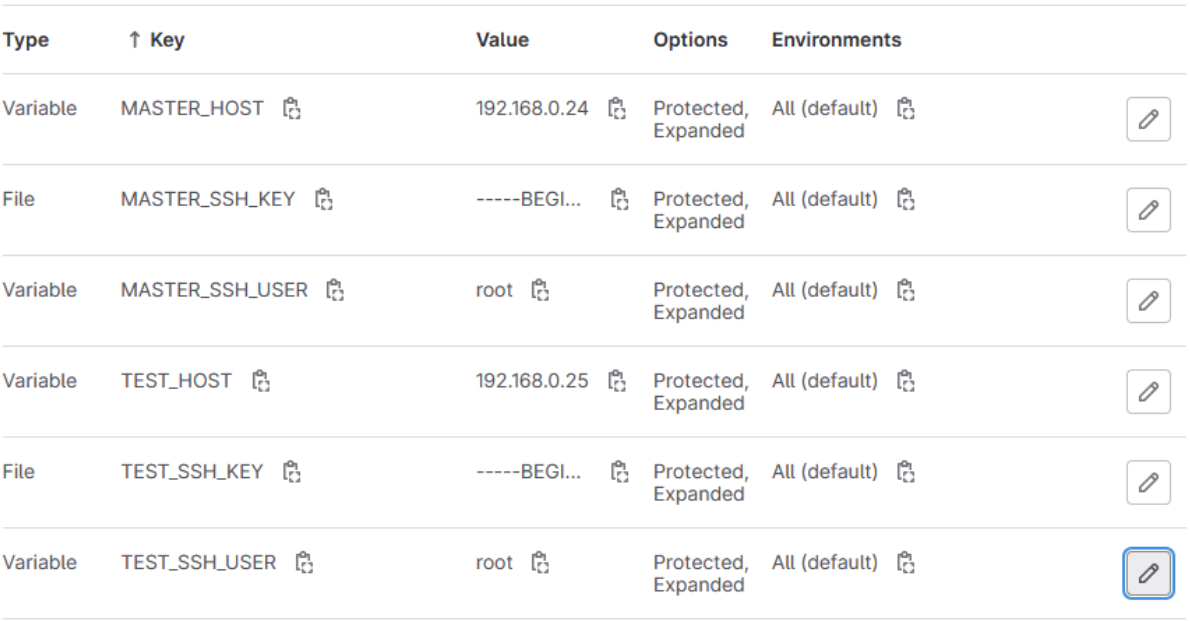
Obrázek 19 - nastavení IP adresy (zdroj vlastní)

Poté je nutné zadat příkaz Apt-get install -y curl a spustit následující skript pro instalaci supervisoru:

```
SUPERVISOR_PORT=8080 \
SUPERVISOR_USERNAME=admin \
SUPERVISOR_PWD=SecretPassword \
\
INSTALL_SCRIPT=$(curl -s "https://raw.githubusercontent.com/bouchal/docker-
master-install/master/linux.sh") \
&& CHECKSUM=$(sha256sum <(echo -n $INSTALL_SCRIPT)  | cut -d ' ' -f 1) \
&& if [ "$CHECKSUM" !=
"269522dd5e7d5cff551e6fdde3694f0d048bc496af466619858cf6654cfeddf9" ]; then
echo "INSTALL SCRIPT CHECKSUM DON'T MACH"; else \
        SUPERVISOR_PORT=$SUPERVISOR_PORT \
        SUPERVISOR_USERNAME=$SUPERVISOR_USERNAME \
        SUPERVISOR_PWD=$SUPERVISOR_PWD \
        bash <(echo "$INSTALL_SCRIPT"); \
fi;
```

skript dostupný z: [24]

Poté následuje přepnutí do režimu root pomocí sudo -i, nainstalování Dockeru pomocí sudo apt install docker.io a vygenerování klíčů ssh příkazem ssh-keygen -m PEM. Klíč je nutný pro provedení nasazení v pipeline pro přístup na server a nasazení kontejneru dockeru v GitLabu, proto je potřeba se přepnout do složky /.ssh pomocí cd .ssh a zkopírovat veřejný klíč do seznamu povolených klíčů pomocí cat id_rsa.pub >> ~/.ssh/authorized_keys. Privátní klíč je nutné zadat do proměnné GitLabu, proto cat id_rsa vypíše tento klíč do konzole. Proměnné se v GitLabu přidávají k projektu určitému projektu přístupem na volbu z menu **Settings > CI/CD** v sekci variables po kliknutí na tlačítko „Expand“, tlačítkem „Add Variable“ se přidávají jednotlivé proměnné. Nastavení proměnných by po konfiguraci produkčního a testovacího serveru mělo vypadat takto:

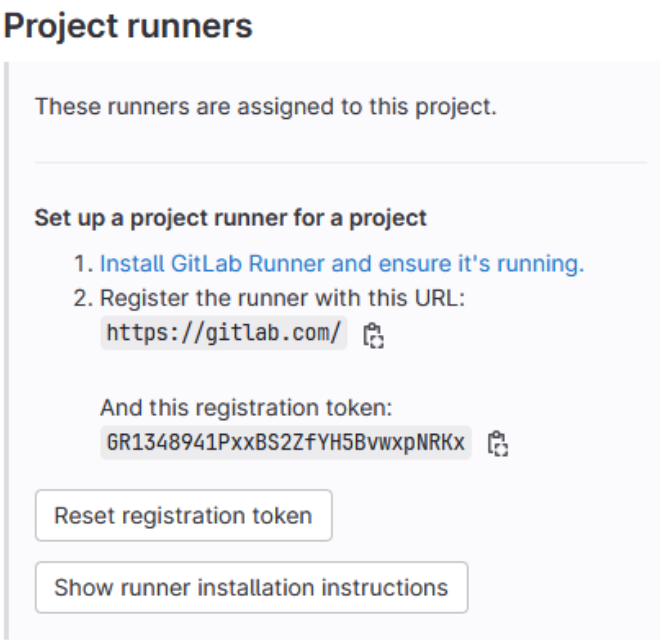


Type	↑ Key	Value	Options	Environments
Variable	MASTER_HOST	192.168.0.24	Protected, Expanded	All (default)
File	MASTER_SSH_KEY	-----BEGI...	Protected, Expanded	All (default)
Variable	MASTER_SSH_USER	root	Protected, Expanded	All (default)
Variable	TEST_HOST	192.168.0.25	Protected, Expanded	All (default)
File	TEST_SSH_KEY	-----BEGI...	Protected, Expanded	All (default)
Variable	TEST_SSH_USER	root	Protected, Expanded	All (default)

Obrázek 20 - proměnné GitLab (zdroj vlastní)

V proměnných <prostředí>_SSH_KEY se nacházejí zkopírované privátní klíče příslušných serverů.

Další důležitou částí pro nasazení projektu je exekutor, který slouží k nasazení projektu na daná prostředí, v této bakalářské práci exekutor je nainstalován na hostitelském počítači s operačním systémem Windows 10 Pro, počítač je blíže popsán v předpokladech. Exekutor je možné stáhnout z https://gitlab-runner-downloads.s3.amazonaws.com/latest/binaries/gitlab-runner-windows-amd64.exe, po umístění do složky C:\Gitlab a přejmenování souboru na gitlab-runner.exe runner je nutné spustit administrátorskou konzoli a ve složce, kde se runner nachází, spustit příkaz gitlab-runner.exe install. Po instalaci runneru následuje jeho registrace pro použití v rámci projektu, pro registraci runneru se využijí informace dostupné na stránce projektu v záložce **Settings > CI/CD** po rozkliknutí tlačítka „expand“ v sekci Runners se objeví token a adresa url potřebná k zaregistrování runneru.



Obrázek 21 - url a token pro registraci runneru (zdroj vlastní)

Příkazem gitlab-runner.exe register se spustí konfigurace runneru. V následujících krocích se vyplňuje url serveru, registrační token, jméno runneru, tagy, exekutor, případně v případě exekutor Docker i Docker image exekutoru. V této práci bylo vyplněno: url: https://gitlab.com/, registrační token: GR1348941PxxBS2ZfYH5BvwxpNRKx jméno: Legion, tagy nevyplněny, exekutor: Docker, Docker image: docker:stable. Po dokončení registrace je nutné upravit vzniklý konfigurační soubor změnou řádku priviledged = false na priviledged = true. Výsledná konfigurace vypadá následujícím způsobem: [23]

Obrázek 22 - konfigurace runneru (zdroj vlastní)

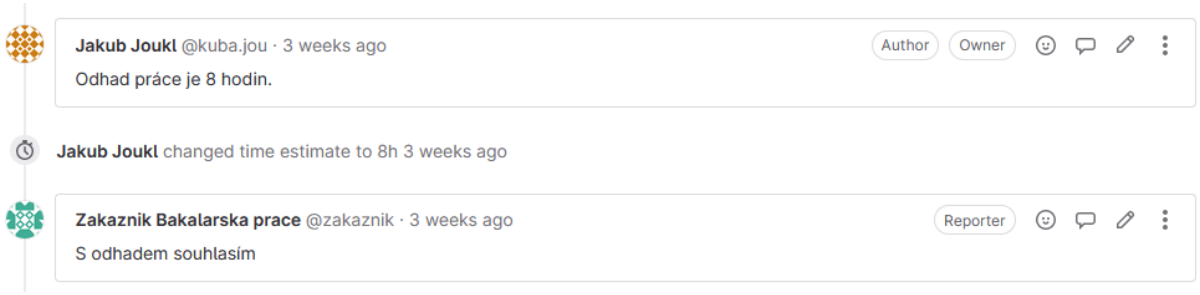
5.7.3. Nastavení CI/CD projektu

Zvolené nastavení CI/CD projektu je odlišné v závislosti na prostředí, etapa CI probíhá na obou prostředích stejně – po nasdílení změn se změny zkompilují a otestují, po úspěšné kompilaci a otestování změn následuje CD, na produkčním prostředí probíhá fáze CD jako Continuous delivery – změny jsou po nasdílení změn na větev main pouze připraveny k manuálnímu nasazení na produkční server, na testovacím prostředí probíhá etapa CD jako Continuous deployment – změny jsou na testovací prostředí po nasdílení na větev test nasazeny automaticky.

Kompletní nastavení průběhu CI/CD v této bakalářské práci je k dispozici v souboru.gitlab-ci.yml, který je uveden jako příloha 1.

5.7.4. První prototyp

Po specifikaci úvodního zadání a otestování projektu na straně vývojářů a pomocí jednotkových testů (https://gitlab.com/kuba.jou/bakalarska-prace/-/issues/3) bylo založeno nové issue „První prototyp“ se štítky prototyp, vysoká priorita, zadání, toto zadání bylo propojeno pomocí funkce „Link issue“ s hlavním zadáním a na testy byl nasazen první prototyp a s odhadem doby práce 8 hodin, který byl, vzhledem k současné absenci jakýchkoliv jiných nasazených verzí aplikace, vyvíjen přímo na větvi test.



Obrázek 23 - odhad práce na prvním prototypu (zdroj vlastní)

Po otestování prototypu zákazníkem byl zjištěn další požadavek – zákazníkovi se nelíbí přítomnost tlačítka „About“, po vyřešení této chyby je možné opravenou verzi rovnou nasadit na produkci. Po odstranění této chyby byla nasazena nová verze, bez tlačítka „About“, na testovací a produkční prostředí.

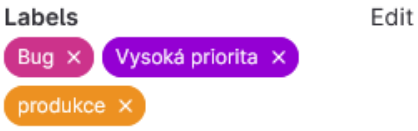
Obrázek 24 - komunikace se zákazníkem ohledně nasazení 1. prototypu (zdroj vlastní)

Zákazník po odebrání tlačítka otestoval funkcionalitu, potvrdil správnost a zavřel issue.

Stav projektu v tomto kroku je shodný se stavem v souboru bakalarska-prace-main - 1 verze 1 issue.zip.

5.7.5. Nalezení a oprava chyby, změna textu tlačítka

Při používání aplikace byla zákazníkem nalezena chyba v případě nezadání žádného textu do vstupního textového pole zprávy – místo toho, aby vstup nešel odeslat, tak se ho podařilo odeslat a návratovou hodnotou od API byl výpis chyby pocházející z jazyka Java, konkrétně ze Spring Frameworku, pro řešení této chyby založil nový incident se štítky bug, vysoká priorita, produkce a předal ho k řešení na vývojáře. V incidentu se také objevil požadavek na změnu textu tlačítka na „Send request“.



Obrázek 25 - štítky incidentu (zdroj vlastní)

Obrázek 26 - zadání incidentu (zdroj vlastní)

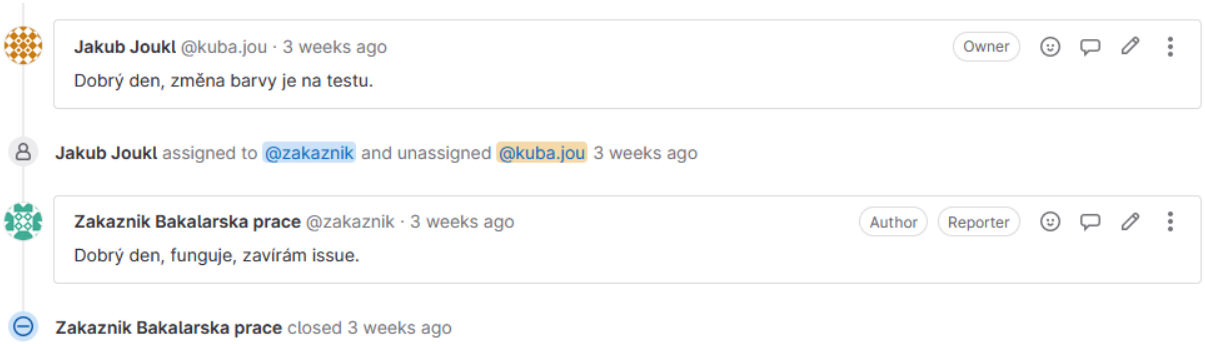
Po vzniku incidentu vznikla nová větev „incident_4_odhalene_chyby“ z větve „main“, na které byla oprava chyby realizována. Po opravě chyby a otestování byla chyba nasazena na větev test do testovacího prostředí a tam následně otestována zákazníkem, po kladné reakci zákazníka následovalo nasazení změn na produkci do větve main.

Stav projektu v tomto kroku je shodný se stavem v souboru bakalarska-prace-main - 1 verze 2 issue.zip.

5.7.6. Změna barvy na testovacím prostředí

V rámci vývoje byl zjištěn další požadavek – změna barvy testovacího prostředí. Zákazník vytvořil issue se štítky test, nízká priorita, zadání. Tato úprava se, vzhledem ke své povaze, prováděla přímo na větvi test. Po provedení a otestování úpravy byla úprava předána zákazníkovi k otestování, po schválení zákazníkem, že úprava funguje, zákazník zavřel issue.

Obrázek 27 - požadavek na změnu barvy testovacího prostředí (zdroj vlastní)

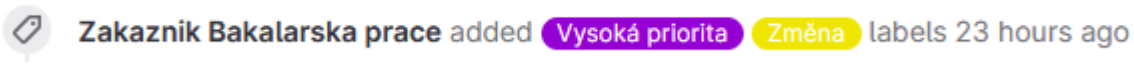


Obrázek 28 - dodání a potvrzení funkčnosti změny barvy (zdroj vlastní)

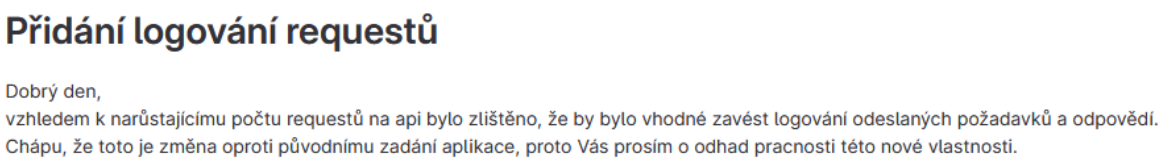
Stav projektu na větvi test je v tomto kroku je shodný se stavem v souboru bakalarska-prace-test - 1 verze 3 issue.zip.

5.7.7. Logování dotazů a vytvoření automatických migračních skriptů DB

Následně se při vedení projektu objevil další požadavek na změnu ze strany zákazníka – zprovoznění logovací funkce pro logování dotazů a odpovědí pro uchovávání statistik. Zadání obsahuje štítky Vysoká priorita, změna, tento požadavek byl provázán se specifikací zadání.

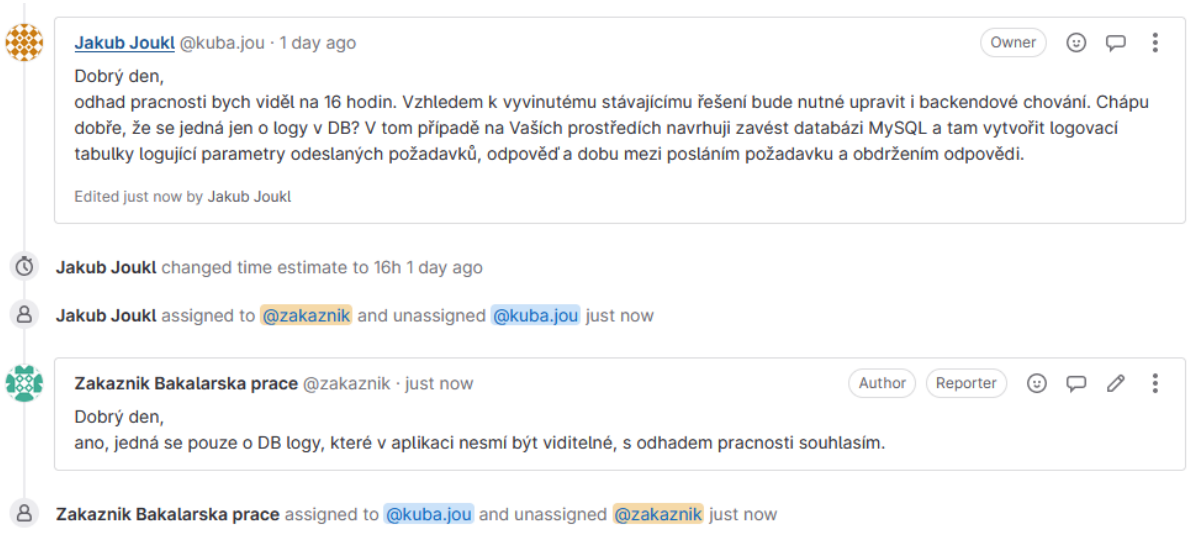


Obrázek 29 - štítky issue logování requestů (zdroj vlastní)



Obrázek 30 - zadání na logování requestů (zdroj vlastní)

Po provedené analýze bylo zjištěno, že aktuálně použitá technologie webové aplikace (Blazor WebAssembly) tuto funkci neumožňuje efektivním způsobem realizovat a bude tedy nutné předělat aplikaci s použitím podobné technologie, Blazor Server, tato skutečnost, společně s požadavkem na potvrzení správnosti pochopení zadání a odhadem pracnosti, byla oznámena zákazníkovi a zákazník odhad pracnosti potvrdil a odsouhlasil vývoj. Pro zajištění co nejvyšší efektivity a automatické synchronizace databáze bylo rozhodnuto o použití frameworku zajišťující generování a aplikování migračních skriptů ve vyvíjené aplikaci v jazyce C# – FluentMigrator.



Obrázek 31 - upřesnění zadání a jeho potvrzení (zdroj vlastní)

5.7.7.1. Konfigurace mySQL DB

Pro konfiguraci databáze je nutné na serverech provést kroky uvedené v této podkapitole. Nejprve je nutné pro aktualizaci balíčků použít následující příkaz: sudo apt-get update && sudo apt-get dist-upgrade, poté se nainstaluje mysql užitím příkazu sudo apt install mysql-server, po instalaci databáze a restartu serverů následuje vytvoření schématu OpenAiApi v DB, toho se dosáhne přepnutím do režimu root a zadáním příkazu mysql, který slouží k připojení k DB. V DB je nutné vytvořit schéma OpenAiApi pomocí SQL příkazu create database OpenAiApi. Poté je nutné nastavit přihlašování přes heslo pro uživatele root, toho se docílí pomocí příkazu use mysql, který přepne konzoli do databáze mysql a příkazem UPDATE mysql.user SET host='%' WHERE user='root'; se nastaví možnost příhlášení uživatele odkudkoliv, příkaz UPDATE mysql.user SET plugin='mysql_native_password' WHERE User='root'; nastaví uživateli identifikaci heslem a poté příkaz ALTER USER 'root' IDENTIFIED BY 'xHeslo123'; změní root uživateli heslo na xHeslo123. Před samotným používáním databáze je nutné provést konfiguraci vystavené IP adresy, na které je mysql služba spuštěna, v režimu root uživatele v kořenovém adresáři použitím příkazu sudo nano /etc/mysql/mysql.conf.d/mysqld.cnf se otevře konfigurační soubor, kde je potřeba zakomentovat IP adresu služby (řádky bind-address a mysqlx-bind-address pro předání služby k připojení na veškerá rozhraní serveru, poté je nutné restartovat servery nebo příslušné mysql služby.

5.7.7.2. Vytvoření automatických migračních skriptů

Po zprovoznění databáze následuje nastavení projektu tak, aby umožňoval generování a aplikování migračních skriptů a připojení k databázi. Připojení k databázi je při sestavování aplikace vybráno následovně – pokud je projekt spuštěn lokálně, je využívána testovací

databáze, jinak se databázoví server volí na základě názvu hostitele – pro testovací prostředí hostované z větve test na testovacím serveru se použije testovací databáze, pro produkční prostředí hostované z větve main na produkčním serveru se použije master databáze.

```
if (builder.Environment.IsDevelopment())
{
    server = "192.168.0.25";
}
else if (Environment.GetEnvironmentVariable("HOSTNAME").Equals("master"))
{
    server = "192.168.0.24";
}
else if (Environment.GetEnvironmentVariable("HOSTNAME").Equals("test"))
{
    server = "192.168.0.25";
}
else
{
    throw new Exception("Unknown environment: " + builder.Environment.EnvironmentName);
}
```

Obrázek 32 - volba IP databáze na základě prostředí (zdroj vlastní)

Spuštění automatických migrací je zajištěno pomocí kódu přímo v aplikaci:

```
builder.Services.AddFluentMigratorCore().ConfigureRunner(
    config => config
    .AddMySql5()
    .WithGlobalConnectionString(connectionString)
    .ScanIn(Assembly.GetExecutingAssembly())
    .For.All()
).AddLogging(config => config.AddFluentMigratorConsole());


var migrationService = builder.Services.BuildServiceProvider().GetService<IMigrationRunner>();
try
{
    migrationService.ListMigrations();
    migrationService.MigrateUp();
}
catch (MissingMigrationsException ex) {
    Console.WriteLine("No migrations found");
}
```

Obrázek 33 - spuštění migrací (zdroj vlastní)

Jednotlivá migrace je samostatná třída C# s metodami Up a Down pro vykonání upgrade a downgrade databáze, odlišení pořadí vykonávání migrací a zajištění jejich unikátnosti je v této bakalářské práci zajištěno pomocí označení každé migrace číslem odpovídající aktuálnímu datumu a času ve tvaru YYYYMMDDHHmmss, kde YYYY představuje rok, MM představuje měsíc, DD představuje den, HH představuje hodinu, mm zastupuje minutu a ss značí sekundu. Migrace zajištění vytvoření tabulky ApiRequestLog pro zápis záznamu do logu ve tvaru id, request, responseTokenCount, creativity, response, timeElapsed, je vytvořena tímto způsobem:

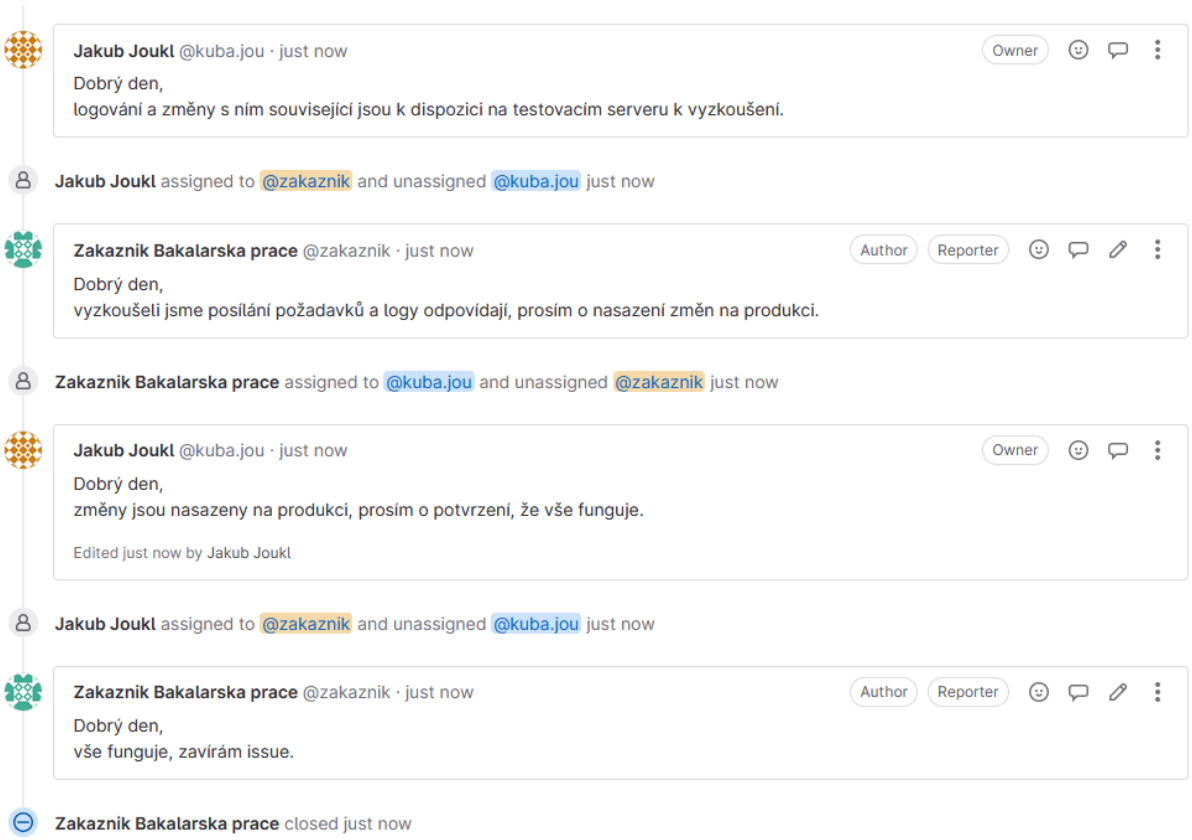
```
[Migration(20230403141601)]
Počet odkazů: 0
public class CreateLog20230403141601 : Migration
{
    Počet odkazů: 0
    public override void Down()
    {
        Delete.Table("ApiRequestLog");
    }

    Počet odkazů: 0
    public override void Up()
    {
        Create.Table("ApiRequestLog")
            .WithColumn("id").AsInt32().NotNullable().Identity().PrimaryKey()
            .WithColumn("request").AsString(8000).Nullable()
            .WithColumn("responseTokenCount").AsInt32().NotNullable()
            .WithColumn("creativity").AsInt64().NotNullable()
            .WithColumn("response").AsString(8000).Nullable()
            .WithColumn("timeElapsed").AsInt64().NotNullable();
    }
}
```

Obrázek 34 - skript pro vytvoření a smazání tabulky ApiRequestLog (zdroj vlastní)

Část Up zajišťuje vytvoření tabulky v schématu v databázi, část down zajišťuje případné odstranění změn provedených touto migrací.

5.7.7.3. Průběh úprav, předání projektu zákazníkovi k otestování

Vykonání úprav proběhlo na samostatné větvi vytvořené přímo pro tyto změny, větvi 6#Pridani_logu. Po předělání aplikace, otestování ze strany vývojářů, nastavení DB na testovacím serveru a vytvoření automatických migračních skriptů následuje nasazení úprav na testovací server a předání aplikace zákazníkovi k otestování, po otestování a přijetí úprav ze strany zákazníka následuje nasazení změn v DB a aplikaci na produkční server. Na následujícím obrázku je zobrazena komunikace vývojáře se zákazníkem ohledně průběhu testování a nasazování změn.

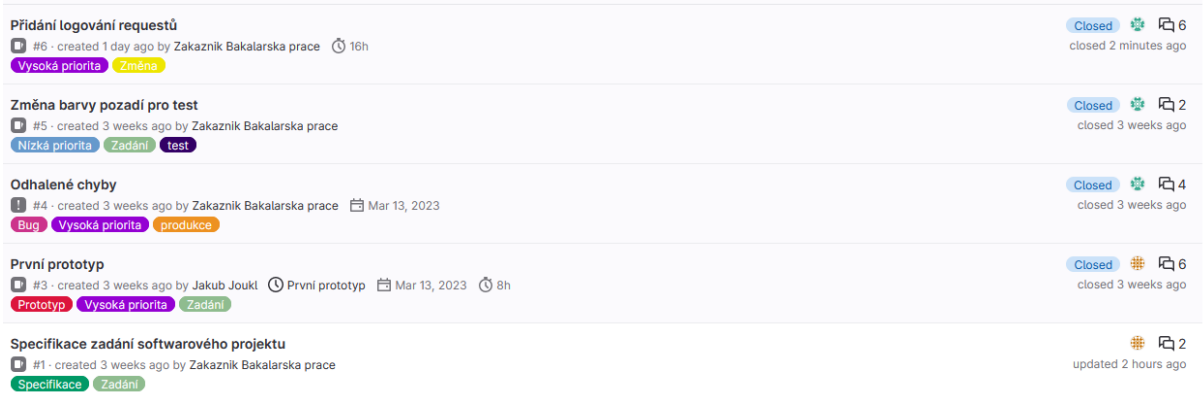


Obrázek 35 - průběh komunikace se zákazníkem ohledně testování a nasazení změn v logování (zdroj vlastní)

Stav projektu na větvi main je v tomto kroku je shodný se stavem v souboru bakalarska-prace-main - 1 verze 4 issue.zip, na větvi test je projekt shodný s bakalarska-prace-test 1 verze 4 issue.zip.

5.7.8. Výsledná podoba struktury zadání a požadavků projektu

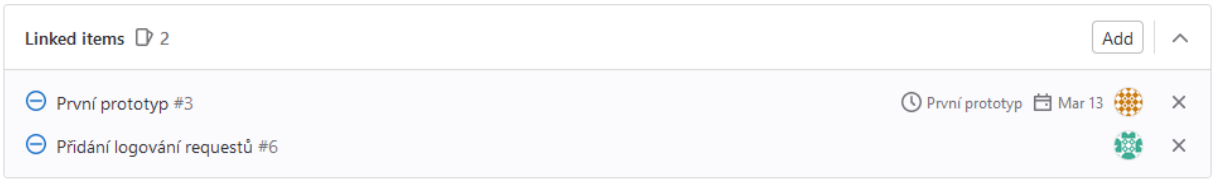
Konečná struktura zadání projektu a požadavků zákazníka vypadá následovně:



Obrázek 36 - konečná struktura zadání projektu (zdroj vlastní)

Jediným otevřeným zadáním projektu v době dokončení projektu je „Specifikace zadání softwarového projektu“, které slouží jako rozcestník pro veškerá existující zadání nových

vlastností a změn v chování projektu pomocí vlastnosti „Link issue“, kterou GitLab poskytuje, propojená zadání a požadavky na změnu se nachází v sekci „Linked items“.

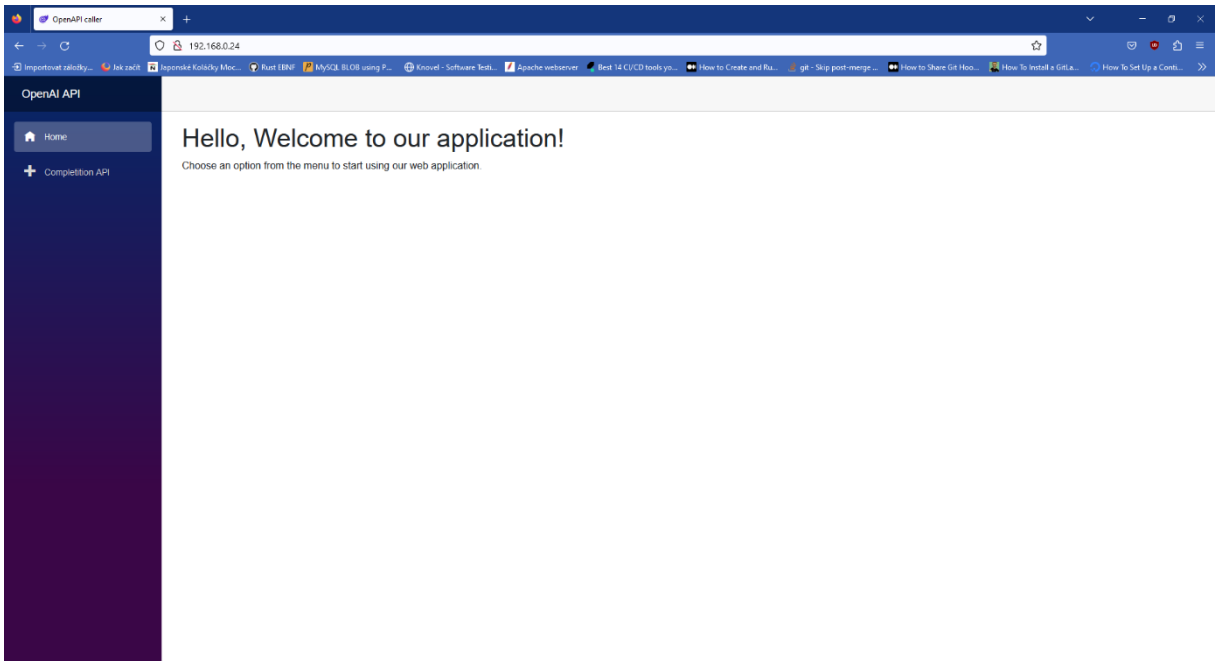


Obrázek 37 - propojená zadání a požadavky na změnu (zdroj vlastní)

5.7.9. Výsledná podoba vyvíjeného projektu na produkčním prostředí

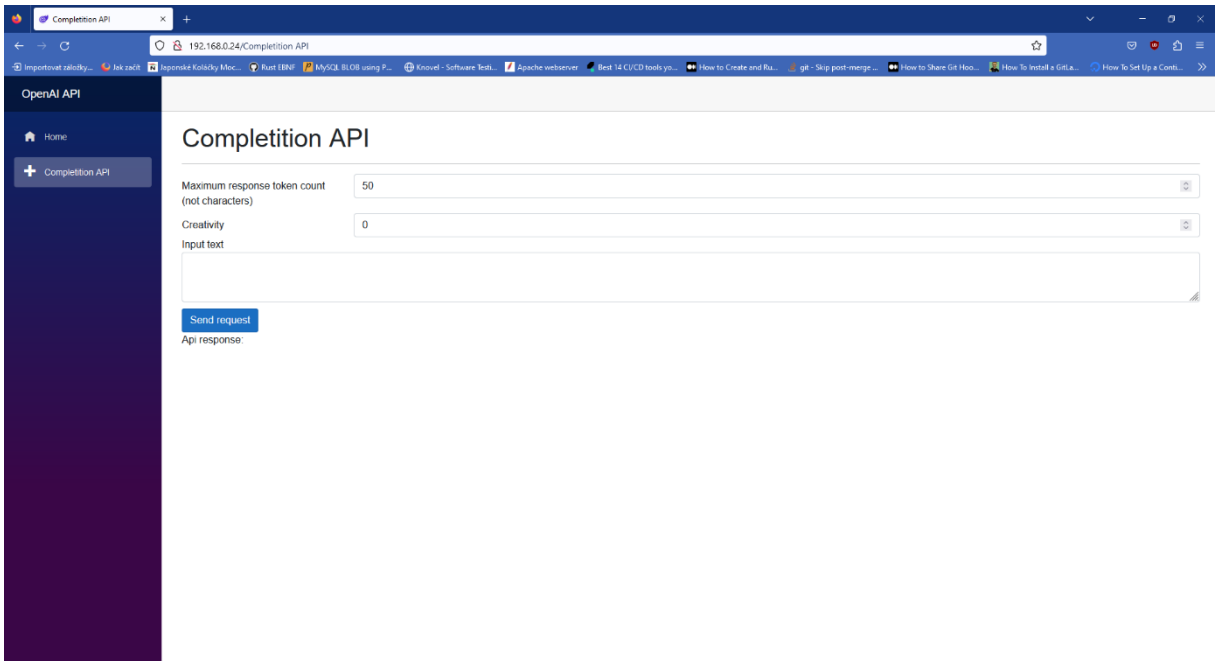
Výsledný vyvinutý software, vycházející z veškerých požadavků zákazníka, obsahuje dvě obrazovky:

1. Obrazovku s uvítáním

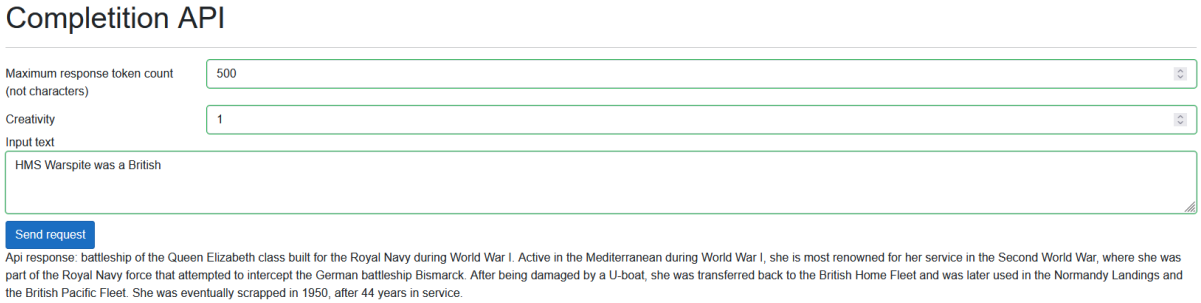


Obrázek 38 - obrazovka s uvítáním (zdroj vlastní)

2. Obrazovku s voláním API, na které je možné zadat dotaz a na základě dotazu dostat odpověď s tím, že dotaz a odpověď API se zalogují do databáze.



Obrázek 39 - obrazovka s voláním API (zdroj vlastní)



Obrázek 40 - ukázka výstupu z aplikace po zavolání API (zdroj vlastní)

ZÁVĚR

Tato závěrečná práce seznamuje čtenáře s metodikami vývoje softwaru, životním cyklem informačního systému a softwaru, s testováním a jeho fázemi, s verzovacími systémy a zejména s verzovacím systémem Git, principy a důvody k zavedení CI/CD a nejčastěji používanými nástroji pro realizaci CI/CD, s principy a zásadami vývoje/kulturou/filozofií DevOps. V další fázi byl představen a popsán scénář realizující tyto principy a postupy vývoje, který popisuje návrh, implementaci systému automatického deploymentu a automatických migračních skriptů a nasazení jednoduchého softwarového projektu na dva různé servery s využitím CI/CD nástroje GitLab.

V teoretické části bylo vysvětleno, jakým způsobem, podle jakých metodik a s využitím jakých postupů a nástrojů se současné rozsáhlé profesionální komerční projekty organizují, jaké nástroje se aktuálně nejvíce využívají pro zajištění efektivního a integritního vývoje softwarového projektu. Pro lepší přehlednost byla vytvořena tabulka obsahující porovnání těchto nástrojů. Práce se dála zabývala principy vývoje/kultura/filozofie DevOps při kompletním řešení softwarového projektu.

V praktické částí práce seznamuje čtenáře s implementací scénáře projektu popisující kompletní řešení realizující principy agilní metodiky vývoje softwaru v rámci návrhu a implementace jednoduché aplikace v jazyce C# s využitím teoretických poznatků z oblasti systému automatického deploymentu a způsobů vedení a organizace týmových projektů uvedených v této bakalářské práci, tento systém je obohacen i o systém Git hooks a o systém zajišťující automatické aplikování databázových migračních skriptů. V popisovaném scénáři je zachycen a znázorněn pohled jak ze strany zákazníka, který nejprve přesně neví, jaké jsou jeho požadavky na vyvinutý software a z toho důvodu je v průběhu vývoje projektu rozvíjí, tak i pohled ze strany vývojářů, kde je zachycena nutnost reagovat na zákazníkem odhalené chyby a nově vzniklé požadavky, v popisovaném scénáři je zachycen průběh komunikace mezi zákazníkem a vývojářem.

POUŽITÁ LITERATURA

[1] AWAD, M. A. A comparison between agile and traditional software development methodologies. *University of Western Australia*, 2005, 30: 1-69. [2] MORAN, Alan. *Managing Agile*. 2015. Cham: Springer International Publishing, 2015. ISBN 3319162616. Dostupné z: doi:10.1007/978-3-319-16262-1

[3] JIRAVA, Pavel. *System development life cycle* [online]. Univerzita Pardubice, 2004 [cit. 2023-02-04]. Dostupné z: http://hdl.handle.net/10195/32471

[4] HAMBLING, Brian, Peter MORGAN, Angelina SAMAROO, Geoff THOMPSON a Peter WILLIAMS. *Software testing*. 2nd ed. Swindon, U.K: BCS, The Chartered Institute for IT, 2010. ISBN 9781906124762.

[5] HLAVA, Tomáš. Fáze a úrovně provádění testů. Testování softwaru: Portál zabývající se problematikou ověřování kvality software. Manuální i automatizované testování. [online]. 21.8.2011 [cit. 2023-02-12]. Dostupné z: http://testovanisoftwaru.cz/tag/akceptacnitestovani

[6] LOURIDAS, P. Version control. *IEEE software* [online]. Los Alamitos: IEEE, 2006, **23**(1), 104-107 [cit. 2023-04-03]. ISSN 0740-7459. Dostupné z: doi:10.1109/MS.2006.32

[7] PRESSMAN, Roger S., 2010. *Software engineering: A practitioner's approach*, 7 ed. New York, USA: McGraw-Hill Higher Education, 2010. ISBN 9780073375977.

[8] ZOLKIFLI, Nazatul Nurlisa, Amir NGAH a Aziz DERAMAN. Version Control System: A Review. *Procedia computer science* [online]. Elsevier B.V, 2018, **135**, 408-415 [cit. 2023-02-15]. ISSN 1877-0509. Dostupné z: doi:10.1016/j.procs.2018.08.191

[9] PILATO, C. Michael, Ben COLLINS-SUSSMAN a Brian W. FITZPATRICK. *Version control with subversion*. 2nd ed. Beijing: O'Reilly, 2008. ISBN 0596510330.

[10] SOMASUNDARAM, Ravishankar. Git: Version Control for Everyone Beginner's Guide. Packt Pub., 2013., http://kim.marcelandkim.com/git/Git_-_Version_Control_for_Everyone.pdf

[11] What is CI/CD?. [online]. Copyright © 2023 Red Hat, Inc. [cit. 15.02.2023]. Dostupné z: https://www.redhat.com/en/topics/devops/what-is-ci-cd

[12] Best 14 CI/CD tools you must know | Updated for 2023. *Katalon Software Quality Management Platform* [online]. Copyright © 2023 Katalon, Inc. All rights reserved. [cit. 16.02.2023]. Dostupné z: https://katalon.com/resources-center/blog/ci-cd-tools

[13] What Is Jenkins? Key Concepts, Architecture & Pros/Cons. *Codefresh | The Developer First CI/CD Platform with GitOps* [online]. Copyright © [cit. 16.02.2023]. Dostupné z: https://codefresh.io/learn/jenkins/

[14] SHETH, Himanshu, 2022. 38 Best CI/CD Tools For 2022 *38 Best CI/CD Tools For 2022 | LambdaTest* [online]. web log. [cit. 16.02.2023]. Dostupné z: https://www.lambdatest.com/blog/best-ci-cd-tools/

[15] GitLab., Features | GitLab [online] [cit. 16.02.2023]. Dostupné z: https://about.gitlab.com/features/

[16] TeamCity: the Hassle-Free CI/CD Tool by JetBrains. *JetBrains: Essential tools for software developers and teams* [online] [cit. 18.2.2023]. Dostupné z: https://www.jetbrains.com/teamcity/

[17] TeamCity vs Jenkins vs Bamboo: A Breakdown | BrowserStack. *Most Reliable App & Cross Browser Testing Platform | BrowserStack* [online]. Copyright © 2011 [cit. 18.02.2023]. Dostupné z: https://www.browserstack.com/guide/teamcity-vs-jenkins-vs-bamboo

[18] Understanding the Bamboo CI Server | Bamboo Data Center and Server 9.2 | Atlassian Documentation. [online]. Copyright © [cit. 18.02.2023]. Dostupné z: https://confluence.atlassian.com/bamboo/understanding-the-bamboo-ci-server-289277285.html

[19] Supported platforms | Bamboo Data Center and Server 9.2 | Atlassian Documentation. [online]. Copyright © [cit. 18.02.2023]. Dostupné z: https://confluence.atlassian.com/bamboo/supported-platforms-289276764.html

[20] Pricing and Plan Information - CircleCI. *Continuous Integration and Delivery - CircleCI* [online]. Copyright © 2023 Circle Internet Services, Inc. [cit. 18.02.2023]. Dostupné z: https://circleci.com/pricing/

[21] AZAD, Nasreen a Sami HYRYNSALMI. DevOps critical success factors — A systematic literature review. *Information and software technology* [online]. Elsevier B.V, 2023, **157**, 107150 [cit. 2023.02.19]. ISSN 0950-5849. Dostupné z: doi:10.1016/j.infsof.2023.107150

[22] FAUSTINO, João, Daniel ADRIANO, Ricardo AMARO, Rubén PEREIRA a Miguel Mira SILVA. DevOps benefits: A systematic literature review. *Software, practice & experience* [online]. Hoboken, USA: John Wiley & Sons, 2022, **52**(9), 1905-1926 [cit. 19.02.2023]. ISSN 0038-0644. Dostupné z: doi:10.1002/spe.3096

[23] How to install, register and start GitLab Runner on Windows - Learn [Solve IT]. *Learn [Solve IT] - Hands-on on Windows, macOS, Linux, Azure, GCP, AWS* [online]. Copyright © 2016 [cit. 08.03.2023]. Dostupné z: https://techdirectarchive.com/2021/09/28/how-to-install-register-and-start-gitlab-runner-on-windows/

[24] BOUCHAL, Antonín, 2020. Deploy app through SSH to VPS with Gitlab Ci [online] [cit. 08.03.2023]. Dostupné z: https://gist.github.com/antoninbouchal/8bdc29ed995252c001070c3f0c7c767e

[25] Git - Git Hooks. *Git* [online]. Dostupné z: https://git-scm.com/book/en/v2/Customizing-Git-Git-Hooks

[26] ATLASSIAN, Git Hooks | Atlassian Git Tutorial [online] [cit. 11.03.2023]. Dostupné z: https://www.atlassian.com/git/tutorials/git-hooks

[27] Notification emails | GitLab. *GitLab Documentation* [online]. Dostupné z: https://docs.gitlab.com/ee/user/profile/notifications.html

[28] ATLASSIAN, Bamboo - pricingAtlassian [online] [cit. 29.03.2023]. Dostupné z: https://www.atlassian.com/software/bamboo/pricing

SEZNAM PŘÍLOH

PŘÍLOHA 1: konfigurační soubor .gitlab-ci.yml (zdroj vlastní)

```
stages:
  - build
  - test
  - deploy
before_script:
  - |
    # docker variables for name and tag of new image
    export DOCKER_TAG="${CI_COMMIT_SHA:0:8}"
    export DOCKER_REPO="$CI_REGISTRY_IMAGE"
    export DOCKER_IMAGE="${DOCKER_REPO}:${DOCKER_TAG}"
    export DOCKER_CACHE_IMAGE="${DOCKER_REPO}:${CI_COMMIT_REF_NAME}"
# Run unit test on dotnet core sdk image
test:
  stage: test
  image: mcr.microsoft.com/dotnet/sdk:7.0
  script:
    - dotnet restore "${CI_PROJECT_DIR}/UnitTests/UnitTests.csproj"
    - dotnet test "${CI_PROJECT_DIR}/UnitTests/UnitTests.csproj"
  only:
    - main
    - test
# Build of docker image and push it to project Container registry
build:
  # Lightweight image with support of running docker in docker
  image: docker:stable
  services:
    - docker:dind
  variables:
    DOCKER_TLS_CERTDIR: ""
  stage: build
  #when: manual
  script:
    # Login to Container registry
    - docker login -u "$CI_REGISTRY_USER" -p "$CI_REGISTRY_PASSWORD"
"$CI_REGISTRY"
    # Load existing images for build optimisation
    - docker pull "$DOCKER_CACHE_IMAGE" || docker pull "${DOCKER_REPO}:main"
|| true # allow failure
    - docker pull "$DOCKER_IMAGE" || true # allow failure
    # Build our docker image
    - docker build --pull --cache-from "$DOCKER_IMAGE" --cache-from
"$DOCKER_CACHE_IMAGE" --cache-from "${DOCKER_REPO}:$CI_COMMIT_BRANCH" -t
"$DOCKER_IMAGE" .
    # Save built image to Container registry under branch name tag and commit
hash tag
    - docker push "$DOCKER_IMAGE"
    - docker tag "$DOCKER_IMAGE" "$DOCKER_CACHE_IMAGE"
```

```
    - docker push "$DOCKER_CACHE_IMAGE"
  only:
    - main
    - test
deploy:
  # Lightweight image with ssh
  image: kroniak/ssh-client
  stage: deploy
  #when: manual
  script:
    # Set right chmod on SSH key file
    - chmod 400 $MASTER_SSH_KEY
    # Login to Gitlab Container registry
    - ssh -o StrictHostKeyChecking=no -i $MASTER_SSH_KEY
"${MASTER_SSH_USER}@${MASTER_HOST}" "docker login -u ${CI_DEPLOY_USER} -p
${CI_DEPLOY_PASSWORD} ${CI_REGISTRY}"
    # Remove old containers and images if exists
    - ssh -o StrictHostKeyChecking=no -i $MASTER_SSH_KEY
"${MASTER_SSH_USER}@${MASTER_HOST}" "docker rm -f ${CI_PROJECT_NAME} || true"
    - ssh -o StrictHostKeyChecking=no -i $MASTER_SSH_KEY
"${MASTER_SSH_USER}@${MASTER_HOST}" "docker rmi \$(docker images -q
${DOCKER_REPO}) || true"
    # Download and run new image
    - ssh -o StrictHostKeyChecking=no -i $MASTER_SSH_KEY
"${MASTER_SSH_USER}@${MASTER_HOST}"
      docker run
      --name=$CI_PROJECT_NAME
      --restart=always
      --network="host"
      -v "/app/storage:/src/storage"
      -d $DOCKER_IMAGE
  only:
    - main
  when: manual
deploy-test:
  # Lightweight image with ssh
  image: kroniak/ssh-client
  stage: deploy
  #when: manual
  script:
    # Set right chmod on SSH key file
    - chmod 400 $TEST_SSH_KEY
    # Login to Gitlab Container registry
    - ssh -o StrictHostKeyChecking=no -i $TEST_SSH_KEY
"${TEST_SSH_USER}@${TEST_HOST}" "docker login -u ${CI_DEPLOY_USER} -p
${CI_DEPLOY_PASSWORD} ${CI_REGISTRY}"
    # Remove old containers and images if exists
    - ssh -o StrictHostKeyChecking=no -i $TEST_SSH_KEY
"${TEST_SSH_USER}@${TEST_HOST}" "docker rm -f ${CI_PROJECT_NAME} || true"
```

```
    - ssh -o StrictHostKeyChecking=no -i $TEST_SSH_KEY
"${TEST_SSH_USER}@${TEST_HOST}" "docker rmi \$(docker images -q
${DOCKER_REPO}) || true"
    # Download and run new image
    - ssh -o StrictHostKeyChecking=no -i $TEST_SSH_KEY
"${TEST_SSH_USER}@${TEST_HOST}"
      docker run
      --name=$CI_PROJECT_NAME
      --restart=always
      --network="host"
      -v "/app/storage:/src/storage"
      -d $DOCKER_IMAGE
  only:
    - test
```

PŘÍLOHA 2: dílčí fáze projektu (zdroj vlastní)

V elektronických přílohách se nacházejí stavy projektu v různých fázích rozpracování a na různých větvích, jedná se o přílohy:

1. bakalarska-prace-test 1 verze 4 issue.zip
2. bakalarska-prace-test - 1 verze 3 issue.zip
3. bakalarska-prace-main - 1 verze 4 issue.zip
4. bakalarska-prace-main - 1 verze 4 issue.zip
5. bakalarska-prace-main - 1 verze 1 issue.zip