UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

DIPLOMOVÁ PRÁCE

2023 Bc. Pavel Křivda

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Automatizované vyhodnocování studentských prací s využitím Git

Bc. Pavel Křivda

Diplomová práce

2023

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2021/2022

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Pavel Křivda**
Osobní číslo:	**I20209**
Studijní program:	**N0613A140007 Informační technologie**
Téma práce:	**Automatizované vyhodnocování studentských prací s využitím Git**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování

Cílem diplomové práce je navrhnout a vytvořit prostředí, které umožní automatické testování a vyhodnocování studentských prací, které jsou odevzdány do kolaboračního Git serveru.
V teoretické části bude provedena rešerše a analýza problému. Je třeba analyzovat dostupné kolaborační servery pro hostování Git repozitářů a jejich možnosti pro: (a) vytváření skupin studentů či odevzdávaných prací, (b) uplatnění sběru dat (webhooks), (c) automatizovaného testování aplikací (CI/CD) proti definované sadě testů. Na základě provedené rešerše bude vybrán server (a související komponenty) pro realizaci daného prostředí. Při realizaci je vhodné zvážit úpravy dostupných kolaboračních a CI/CD prostředí, stejně jako možnost implementovat externí aplikace pro doplnění funkcionalit.
V praktické části bude navržen a realizován systém, který umožní automatizované vyhodnocování studentských prací v prostředí kolaboračního Git serveru. Odevzdaná práce (po jejím nahrání na server) bude automaticky vyhodnocena proti definované sadě testů. Výsledné hodnocení bude dostupné, jak studentům, tak vyučujícím. Vyučující bude mít k dispozici přehled za všechna definovaná cvičení či skupiny. Vytvořené prostředí musí disponovat podporou minimálně pro Javu a C#.

Rozsah pracovní zprávy: **50 – 60 stran**
Rozsah grafických prací: **–**
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

CHACON, Scott. *Pro Git*. Praha: CZ.NIC, c2009. CZ.NIC. ISBN 978-80-904248-1-4.
SMART, John. *Jenkins: The Definitive Guide*. O'Reilly Media, 2011. ISBN 9781449305352.

Vedoucí diplomové práce: **Ing. Roman Diviš**
Katedra softwarových technologií

Datum zadání diplomové práce: **8. listopadu 2021**
Termín odevzdání diplomové práce: **20. května 2022**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

prof. Ing. Antonín Kavička, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 30. listopadu 2021

Prohlašuji:

Tuto práci 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., autorský zákon, 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. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 25. 8. 2023

Bc. Pavel Křivda

PODĚKOVÁNÍ

Chtěl bych poděkovat panu Ing. Romanu Divišovi, Ph.D., za odborné vedení práce a cenné rady, které mi pomohly tuto práci zkompletovat. Dále bych rád poděkoval mojí rodině, která mě neúnavně podporuje po celou dobu mých studií.

ANOTACE

Práce se zaměřuje na vytvoření prostředí pro automatizované vyhodnocování studentských prací s využitím Git repozitářů. V rámci tohoto prostředí má každý vyučující možnost spravovat předměty, úlohy a skupiny studentů. Pro každý předmět je možné vytvořit úlohy (projekty) v programovacích jazycích Java nebo C#, které musí studenti vypracovat. Úloha obsahuje sady testů definovaných vyučujícím, které jsou využity k vyhodnocení odevzdaných studentských prací na kolaboračním Git serveru.

V práci je popsána platforma Docker na která je vytvořen systém pro automatizované vyhodnocování studentských prací. Ten se skládá z komponent Gitlab-CE, Gitlab-runner, MySQL a vlastního integračního nástroje. Tyto služby jsou dále teoreticky popsány. V praktické části je implementována konfigurace pro Docker Compose a administrační aplikace pro automatické vyhodnocování studentských prací.

KLÍČOVÁ SLOVA

Automatizované vyhodnocování, Git, Gitlab, Docker, CI/CD

TITLE

Automated evaluation of student exercises using Git

ANNOTATION

Thesis is focused on creation an environment for automated evaluation of student work using Git repositories. Within this environment, each teacher has the ability to manage courses, assignments and groups of students. For each course it is possible to create assignments (projects) in Java or C# programming languages that students have to develop. The assignment contains sets of tests defined by the teacher, which are used to evaluate the submitted student work on the collaborative Git server.

Thesis describes the Docker platform, on which a system for automated evaluation of student works is created. It consists of Gitlab-CE, Gitlab-runner, MySQL and its own integration tool. These services are further theoretically described. In the practical part, the configuration for Docker Compose and administrative applications for the automatic evaluation of student works are implemented.

KEYWORDS

Automated evaluation, Git, Gitlab, Docker, CI/CD

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM ZKRATEK

API	Application Programming Interface
CD	Continuous Delivery
CI	Continuous Integration
CRM	Customer Relationship Management
CVS	Concurrent Version System
DNS	Domain Name System
DOM	Document Object Model
ERD	Entity-relationship model
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
ICT	informačních a komunikačních technologií
IDE	Integrated Development Environment
JJWT	JSON Web Token for Java
JSON	JavaScript Object Notation
JWT	JSON Web Token
RCS	Revision Control System
SAAS	Software as a Service
SSL	Secure Sockets Layer
TLS	Transport Layer Security
UI	User Interface
WWW	World Wide Web
YAML	YAML Ain't Markup Language

ÚVOD

V dnešní době, kdy se programování stává stále důležitější součástí vzdělávání, mají studenti často za úkol vytvářet programové kódy a odevzdávat své práce pro vyhodnocení. Tento proces vyhodnocování však může být časově náročný a zatěžující pro vyučující, zejména pokud mají spravovat velký počet studentů a jejich rozsáhlé projekty. Manuální vyhodnocování každého kódu a poskytování zpětné vazby je časově a náročné a může vést k dlouhým prodlevám mezi odevzdáním a zpětnou vazbou.

Automatizace tohoto procesu prostřednictvím využití Git repozitářů a specializovaných nástrojů pro testování a vyhodnocování může výrazně zefektivnit celý proces a přinést výhody jak studentům, tak vyučujícím. Git repozitáře poskytují prostředí pro správu kódu, které umožňuje studentům snadno odevzdávat své práce a udržovat jejich historii. Díky tomu je možné sledovat vývoj jejich kódu a přesně určit, které změny byly provedeny. To usnadňuje vyučujícím sledování pokroku a poskytování zpětné vazby.

Nástroje pro testování a vyhodnocování umožňují automatizaci procesu testování studentských prací. Jelikož vyučující mají možnost definovat sadu testů pro jednotlivé úlohy, které budou studenti implementovat. Nástroj poté automaticky spustí tyto definované testy a vyhodnotí každou studentskou práci a poskytnou výsledky testování. To umožňuje rychlou a objektivní kontrolu, zda studenti splnili požadavky úkolu. Vyučující tak mohou poskytnout rychlou zpětnou vazbu a studenti mají okamžitou informaci o úspěšnosti svého kódu. Tím se snižuje prodleva mezi odevzdáním a zpětnou vazbou, což je důležité pro efektivní výukový proces.

Automatizace vyhodnocování studentských prací s využitím Git repozitářů a nástrojů pro testování a vyhodnocování přináší výhody pro obě strany. Studenti mají přehled o svém pokroku a okamžitou zpětnou vazbu na svou práci, což jim umožňuje lépe se rozvíjet a zlepšovat své dovednosti. Vyučujícím je usnadněn proces vyhodnocování a poskytování zpětné vazby, což umožňuje věnovat více času individuálnímu doprovázení studentů a jejich lepšímu vývoji.

V této diplomové práci bude navrženo a implementováno prostředí, které umožní automatizované vyhodnocování studentských prací s využitím Git repozitářů. Cílem je zefektivnit proces vyhodnocování, usnadnit studentům odevzdávání a získávání zpětné vazby a umožnit vyučujícím efektivní sledování pokroku studentů.

1 TEORETICKÝ ÚVOD

V této kapitole jsou popsané základní technologie, které jsou nezbytné pro automatizované vyhodnocování studentských prací s využitím Git.

1.1 WWW

Základem World Wide Web (WWW) je hypertext, což představuje systém vzájemného propojení dokumentů, které jsou dostupné pomocí internetu. Tento mechanismus umožňuje zobrazování jednotlivých webových stránek, textů, obrázků a dalších multimédií, na něž odkazují webové stránky, při použití webového prohlížeče. Pro navigaci ve WWW slouží hypertextové odkazy, které umožňují uživatelům přesměrování na jiné texty nebo obecně na jiný obsah. Tyto odkazy mohou mít různé grafické podoby, jako jsou tlačítka, obrázky nebo jiné interaktivní prvky, a tím zpřístupňují uživatelům snadný a intuitivní způsob procházení informacemi na internetu. [1]

1.2 HTTP

HTTP (Hypertext Transfer Protocol) je protokol určený původně pro výměnu hypertextových dokumentů mezi webovým serverem a webovým prohlížečem. V současné verzi obvykle využívá protokol TCP na portu 80 a umožňuje přenos souborů.

HTTP je bezstavový protokol fungující na principu dotazu a odpovědi. To je důvodem proč kupříkladu nelze uložit obsah košíku v internetovém obchodě. Tento problém je možné vyřešit různými postupy, např. využitím cookies. [2]

1.3 HTTPS

HTTPS (Hyper Text Transfer Protocol Secure) je zabezpečená verze protokolu HTTP, která využívá protokol TCP na portu 443 pro komunikaci. Jeho hlavním účelem je umožnit vytvoření šifrovaného a bezpečného spojení mezi webovým serverem a webovým prohlížečem. Tato zabezpečená komunikace, zprostředkovaná protokoly TLS (dříve SSL) a HTTP, slouží k ochraně citlivých informací před neoprávněným přístupem, odcizením nebo manipulací. Díky HTTPS jsou data přenášená mezi serverem a prohlížečem šifrována a zabezpečena, což přispívá k větší důvěryhodnosti a bezpečnosti webové komunikace. [2]

1.4 Doména

Doména je identifikační název v rámci systému DNS (Domain Name System), který slouží k překladu lidsky čitelného jména (např.: seznam.cz) na IP adresu, což umožňuje jednoznačnou identifikaci a lokalizaci zdrojů na internetu. Každá doména je tvořena názvem a doménovou koncovkou, která indikuje její typ nebo geografickou oblast (například: .cz, .com). Domény mají klíčovou roli v architektuře internetu a umožňují uživatelům snadný přístup k webovým stránkám, e-mailovým serverům a dalším službám online. Díky systému domén je možné provozovat komplexní webové struktury a usnadnit identifikaci zdrojů na globální síti internetu. [3]

1.5 JWT

JWT (JSON Web Token) je standardizovaný formát pro bezpečné a spolehlivé přenosy informací mezi stranami ve formě JSON objektů. Jedná se o kompaktní token, který může být podepsán a zašifrován, což zajišťuje jeho autentičnost a důvěrnost. JWT se často používá pro ověření uživatelů a autorizaci ve webových aplikacích a API, kdy informace o uživateli nebo oprávněních jsou zakódovány do tokenu, který je podepsán serverem. Tímto způsobem je možné ověřit identitu uživatele a zabezpečit komunikaci mezi klientem a serverem, aniž by bylo nutné ukládat session data na serverové straně. JWT se stává stále více populárním řešením pro zajištění bezpečnosti a autentizace v moderních webových aplikacích a službách. [4]

1.6 Git

Git je distribuovaný verzovací systém, který slouží ke sledování změn v kódu a efektivní správě verzí softwarových projektů. Byl vyvinut Linusem Torvaldsem v roce 2005 jako nástroj pro správu vývoje jádra Linuxu a od té doby si získal velkou popularitu díky své schopnosti umožňovat nezávislou práci vývojářů bez nutnosti neustálého připojení k centrálnímu serveru. Git je vysoce efektivní, rychlý a poskytuje širokou škálu nástrojů pro správu verzí a spolupráci mezi vývojáři. Změny v Gitu jsou systematicky zaznamenávány v commit záznamech, což usnadňuje sledování historie projektu a umožňuje snadné zpětné procházení provedených změn. V důsledku svých vlastností se Git stal nezbytným a důležitým nástrojem pro profesionální správu verzí a spolupráci v odborném softwarovém vývoji. [5]

1.7 CI/CD

Continuous Integration a Continuous Deployment (CI/CD, pipeline) jsou klíčovými praktiky v oblasti softwarového vývoje, které revolučně změnily způsob, jakým se aplikace vyvíjejí, testují a nasazují. Continuous Integration se zaměřuje na pravidelné a automatizovanou integraci kódu od různých vývojářů do hlavního repozitáře, což umožňuje častou integraci nových změn, rychlé odhalení konfliktů a detekci chyb díky automatickému spouštění testů. Tím výrazně zvyšuje kvalitu a stabilitu kódu. S tímto procesem úzce souvisí Continuous Deployment, který automatizuje nasazování ověřeného softwaru do produkčního prostředí. Po úspěšném splnění všech testů v rámci CI je software automaticky nasazen, což vede ke zkrácení vývojového cyklu, rychlejšímu uvedení nových funkcionalit na trh a zvýšení spolehlivosti nasazení. CI/CD tedy nejen zvyšuje efektivitu a produktivitu vývojářského týmu, ale také poskytuje uživatelům lepší zkušenost s aplikacemi díky vyšší kvalitě a rychlosti inovací. [6]

1.8 GitLab

GitLab je webová platforma pro správu Git repozitářů a nabízí široké spektrum nástrojů pro celý životní cyklus softwarového vývoje. Mezi klíčové funkce patří možnost hostování vlastních Git repozitářů, sledování problémů (issue tracking), automatizace Continuous Integration a Continuous Deployment (CI/CD), správa verzí, wiki a další. GitLab je distribuován ve dvou verzích: open-source verzi GitLab Community Edition (CE) a komerční verzi GitLab Enterprise Edition (EE), která nabízí rozšířené funkce pro podnikové nasazení. Díky své flexibilitě, uživatelsky přívětivému rozhraní a bohatým funkcím je GitLab velmi oblíbeným nástrojem pro sdílení a spolupráci na projektech. [7]

1.9 GitHub

GitHub je největší a nejznámější hostingová platforma pro Git repozitáře. Poskytuje uživatelům centralizované místo pro sdílení a spolupráci na open-source i soukromých projektech. Uživatelé mohou snadno vytvářet vlastní repozitáře, zveřejňovat kód, sledovat problémy, provádět recenze kódu a zapojovat se do diskusí. GitHub je populární mezi vývojářskými komunitami, umožňuje širokou škálu integrací s nástroji třetích stran a poskytuje rozhraní API pro automatizaci procesů vývoje. Projekty vytvořené na GitHubu jsou často využívány jako základ pro mnoho open-source projektů. [7, 8]

1.10 Bitbucket

Bitbucket je hostingová platforma společnosti Atlassian, která umožňuje správu Git a Mercurial repozitářů. Umožňuje vytváření a sdílení veřejných i soukromých repozitářů a poskytuje nástroje pro sledování problémů a automatizaci CI/CD. Bitbucket se často používá v prostředích, která preferují jiné verzovací systémy než pouze Git, jako je například Mercurial. Díky integraci s dalšími nástroji společnosti Atlassian, jako je Jira nebo Confluence, nabízí Bitbucket ucelené prostředí pro plánování, sledování a spolupráci na projektech. [9]

1.11 Gitea

Gitea je další open-source platforma pro správu Git repozitářů, která je odvozena od projektu Gogs. Gitea je navržen tak, aby ji bylo snadné nasadit a aby poskytovala uživatelům jednoduché a intuitivní rozhraní. Podporuje širokou škálu funkcí, včetně integrace s webhooky, sledování problémů, CI/CD a API pro automatizaci. Gitea se zaměřuje na nízké využití zdrojů, proto je vhodné ji využít pro menší a střední projekty nebo pro uživatele, kteří preferují jednoduchost a minimalismus. [10]

1.12 RhodeCode

RhodeCode je komplexní platforma pro správu Git, Mercurial a SVN repozitářů, která nabízí pokročilé nástroje pro správu verzí, spolupráci a bezpečnost. Jeho webové rozhraní nabízí uživatelům vytvářet a spravovat repozitáře, provádět revizi kódu, sledovat problémy atd. Navíc RhodeCode disponuje pokročilými funkcemi pro autentizaci, autorizaci a správu uživatelů a skupin, tak aby bylo možné efektivně spravovat přístup k repozitářům. Díky své flexibilitě, podpoře různých verzovacích systémů a rozsáhlé integraci s dalšími nástroji, je RhodeCode vhodnou volbou pro střední a velké organizace, které vyžadují robustní platformu pro centralizovanou správu svých repozitářů a zlepšení spolupráce vývojových týmů. [11]

1.13 Kallithea

Kallithea je open-source platforma pro správu Git a Mercurial repozitářů, která poskytuje uživatelům komplexní nástroje pro sdílení, spolupráci a sledování vývoje softwarových projektů. S vlastním webovým rozhraním umožňuje uživatelům vytvářet a spravovat repozitáře, řešit revize kódu, sledovat problémy, což přispívá k transparentnímu a efektivnímu vývojovému procesu. Kallithea nabízí pokročilé funkce pro správu verzí, zabezpečení a integraci s dalšími

nástroji, a je vhodnou volbou pro střední a velké podniky, které hledají robustní platformu pro centralizovanou správu svých repozitářů a zvýšení produktivity vývojových týmů. [12]

2 KOLABORAČNÍ SERVERY PRO HOSTOVÁNÍ GIT REPOZITÁŘŮ

V této kapitole je provedena analýza dostupných kolaboračních serverů pro hostování Git repozitářů.

2.1 Kolaborační server

Kolaborační server s podporou Gitu je v současné době nezbytným pilířem moderního softwarového vývoje. Jedná se o online platformu, která slouží jako centrální místo pro správu verzování kódu a spolupráci týmů. Díky kolaboračním serverům se výrazně zjednodušuje a zlepšuje proces správy verzí, což vede ke zvýšení efektivity, transparentnosti a kvality vývoje software. [13]

Historie kolaboračních serverů sahá až do doby, kdy se softwarový vývoj začal rozvíjet a projekty začaly nabývat na komplexitě. Původní verzovací systémy jako RCS nebo CVS sloužily primárně k ukládání změn v kódu, ale jejich schopnosti byly omezené. S rostoucím počtem vývojářů a rozsahem projektů vznikla potřeba vyspělejších verzovacích systémů. [13]

V 90. letech minulého století se začal masivně rozšiřovat verzovací systém SVN, který přinesl vylepšenou podporu pro více uživatelů a usnadnil správu verzí. Nicméně, i přesto měl některá omezení, zejména pokud jde o práci v distribuovaných týmech a správu větví (branching). [13]

Revolučním okamžikem v historii verzovacích systémů bylo představení Gitu Linusem Torvaldsem v roce 2005. Git přinesl zcela nový decentralizovaný přístup, který umožňoval každému vývojáři mít vlastní lokální kopii celého repozitáře. To znamená, že vývojáři mohou pracovat offline, bez nutnosti neustálého spojení s centrálním serverem. Díky této revoluční vlastnosti se Git stal preferovaným verzovacím systémem, zejména pro velké a distribuované týmy. [13]

Git umožňuje vývojářům vytvářet, sledovat, spravovat a sdílet změny v kódu pomocí tzv. commitů. Většina verzovacích operací je prováděna lokálně, což zajišťuje rychlou odezvu a snižuje závislost na rychlosti a dostupnosti centrálního serveru. Díky tomu je Git mnohem efektivnější a robustnější ve srovnání s tradičními centralizovanými verzovacími systémy. [13]

S rozvojem Gitu se rovněž rozrostla řada online kolaboračních serverů, které umožňují týmům sdílet, spravovat a spolupracovat na svých repozitářích Gitu. Tyto kolaborační platformy

nabízejí širokou škálu funkcí a nástrojů, které usnadňují vývoj software. V následujících kapitolách jsou rozebrány vlastnosti a funkce kolaboračního serveru. [13]

2.1.1 Repozitář

Když hovoříme o vývoji software, často narazíme na pojem „repozitář“. Na jedné straně máme Git repozitář, což je datová struktura umožňující sledování změn v souborech a umožňující spolupráci více vývojářů. Git repozitář obsahuje historii všech změn a lze jej jednoduše klonovat a synchronizovat mezi lokálním počítačem a vzdáleným serverem. [13]

Oproti tomu kolaborační platformy, jako GitHub či GitLab nabízejí co bychom mohli nazvat „projekty“. Tyto projekty, kromě samotného Git repozitáře, zahrnují řadu dalších nástrojů a funkcí. Mezi ty patří například nástroje pro správu úkolů, wiki, diskusní fóra a integraci s nástroji CI/CD. Díky tomu umožňují komplexní správu celého software projektu. [13]

Tedy, zatímco Git se primárně stará o verzi kontrolu a sdílení změn kódu, platformy jako GitHub rozšiřují tuto funkcionalitu o řadu dalších nástrojů, které usnadňují život vývojářským týmům. V praxi to znamená, že Git je ideální pro sledování a sdílení změn v kódu, zatímco platformy typu GitHub přidávají dodatečné nástroje a funkcionalitu potřebnou pro komplexní správu vývojového projektu. [13]

2.1.2 Zabezpečení

Bezpečnostní prvky a opatření jsou nezbytnou součástí každého kolaboračního serveru s podporou Gitu. Zabezpečení zahrnuje rozsáhlé auditovatelné protokoly, které sledují a zaznamenávají veškeré akce a změny v repozitářích a systému. Pokročilá detekce hrozeb je implementována pro identifikaci možných bezpečnostních rizik a narušení. Důležitou součástí zabezpečení je podpora pro bezpečnostní standardy, což zahrnuje správu přístupových práv, šifrování dat a další bezpečnostní opatření. To vše je klíčové pro ochranu kódu, citlivých informací a předcházení potenciálním kybernetickým hrozbám. [13]

2.1.3 Pull requesty/merge requesty

Pull requesty nebo také merge requesty se staly standardem v kolaborativním vývoji a představují mechanismus, který umožňuje vývojářům navrhovat změny v kódu a provádět komplexní revizi těchto změn před jejich sloučením do hlavní větve projektu. Díky těmto funkcím mohou vývojáři diskutovat o provedených úpravách, provádět automatické testy

a získávat zpětnou vazbu od ostatních členů týmu. Pull requesty usnadňují kontrolu kvality kódu a minimalizují riziko chyb, které by mohly být sloučeny do projektu. [14]

2.1.4 Integrace s CI/CD

Integrace s nástroji pro kontinuální integraci a nasazování (CI/CD) je klíčovým prvkem moderního softwarového vývoje. Díky této integraci je možné automatizovat testování, nasazování a monitorování softwaru. Při každém provedení změn v repozitáři se spouští automatizované testy, které ověřují funkčnost a stabilitu kódu. Pokud jsou testy úspěšné, dochází k automatickém nasazen do produkčního prostředí. Tímto způsobem se zajišťuje rychlé a spolehlivé vydávání softwaru, které je klíčové pro agilní a efektivní vývojové procesy. [15]

2.1.5 Zadávání úkolů a sledování problémů

Kolaborační servery umožňují zadávání úkolů a sledování problémů, což je neocenitelná funkcionalita pro řízení projektů. Týmy mohou vytvářet úkoly, rozdělovat je mezi členy týmu, prioritizovat je a sledovat jejich stav. Tato centrální správa úkolů usnadňuje plánování, koordinaci a organizaci vývojových aktivit, což vede k vyšší produktivitě a efektivitě týmu. [16]

2.1.6 Wiki a dokumentace

Wiki a dokumentace na kolaboračních serverech zajišťují snadnou dostupnost aktualizovaných a organizovaných informací o projektu. Týmy mohou vytvářet a ukládat dokumenty, návody, instrukce a další relevantní informace, které jsou důležité pro vývoj projektu. To usnadňuje rychlejší zaučení nových členů týmu, uspořádání a strukturování informací na jednom místě a tím snižuje riziko informačního chaosu. [17]

2.1.7 Integrace s externími nástroji

Kolaborační servery často podporují integraci s externími nástroji a aplikacemi, jako jsou CRM systémy, databáze nebo cloudová řešení. Tímto způsobem se zvyšuje flexibilita a rozšiřují možnosti uplatnění kolaboračního serveru, což umožňuje snadnější a efektivnější integraci s existujícími nástroji a procesy v organizaci. [18]

2.1.8 Vizualizace

Vizualizační nástroje na kolaboračních serverech poskytují grafy závislostí, vizualizace větví a analytické dashboardy, které umožňují lepší orientaci ve vývojovém procesu a poskytují hluboký vhled do stavu projektu. Díky těmto vizualizačním prvkům lze lépe monitorovat pokrok projektů, identifikovat možné problémy a optimalizovat vývojové postupy. [19]

2.1.9 Notifikace

Notifikační systémy na kolaboračních serverech zahrnují pokročilé nástroje, jako jsou integrovaní botové a automatizovaná upozornění. Tyto notifikační mechanismy udržují tým v obraze ohledně aktuálních událostí a dění v projektu. Zvyšují reaktivitu týmu a umožňují rychlou reakci na důležité situace a změny v repozitářích. [20]

2.1.10 Zálohy a obnova

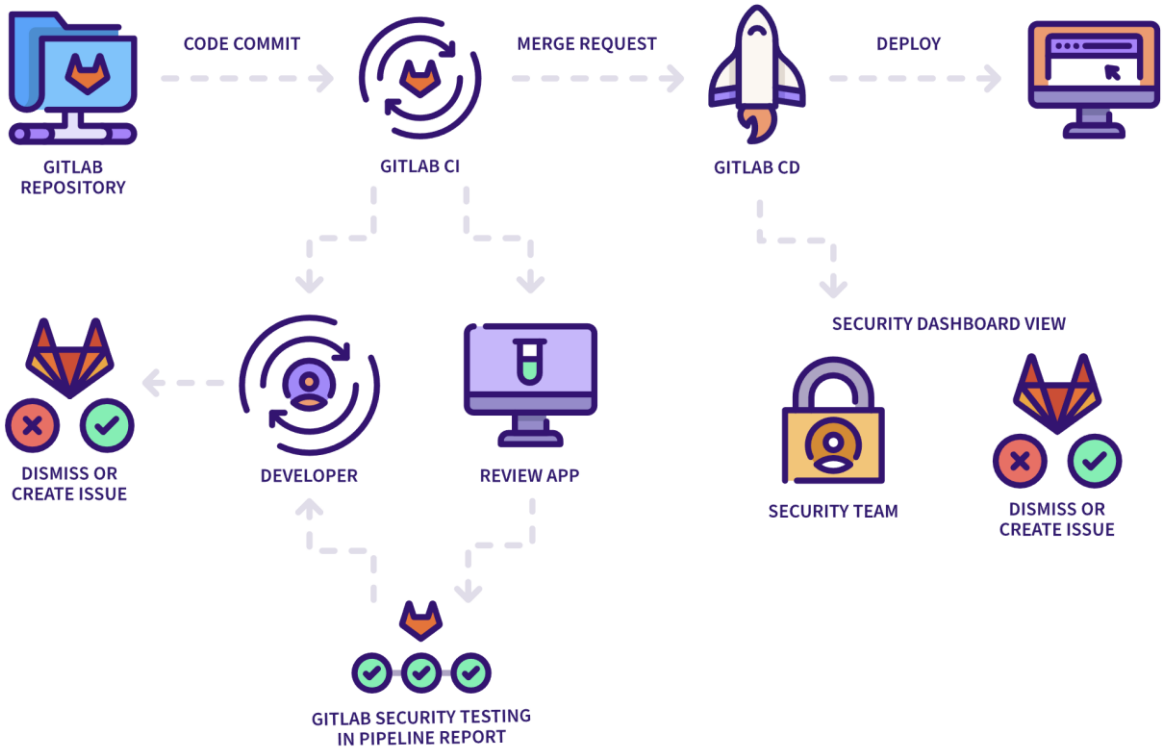
Pro zajištění bezpečnosti dat a minimalizaci rizika datových ztrát poskytují kolaborační servery robustní řešení pro zálohování a obnovu dat. To zahrnuje pravidelné zálohování repozitářů, jejich verzí a metadat. V případě nečekaných problémů nebo havárií lze snadno obnovit data z uložených záloh a minimalizovat negativní dopady na vývojový proces. [21]

2.1.11 Pokročilé zabezpečení

Mnoho kolaboračních serverů dnes nabízí pokročilé zabezpečení pro lepší ochranu citlivých dat a kódu. End-to-end šifrování zajišťuje, že data jsou chráněna po celou dobu od vytvoření až po uložení na serveru. Sledování anomálií v přístupech umožňuje detekovat neobvyklé aktivity a možné pokusy o neoprávněný přístup. Podpora pro bezpečnostní klíče poskytuje dodatečnou vrstvu zabezpečení, což zvyšuje bezpečnost celého vývojového procesu. Díky těmto technologiím mohou týmy a organizace spoléhat na bezpečný a důvěryhodný kolaborační server pro správu svých Git repozitářů. [22]

2.2 Technologie kolaboračního serveru

Pro kolaborační server schopný hostovat Git repozitáře jsou některé technologie klíčové pro zajištění jeho správného fungování, bezpečnosti a výkonu. Obrázek 1 prezentuje architektonickou strukturu kolaboračního serveru GitLab.



Obrázek 1 – Struktura kolaboračního serveru (zdroj: [23])

2.2.1 Aplikační server

Aplikační server představuje jádro a srdce kolaboračního systému. Díky svému modulárnímu designu umožňuje jednoduché přidávání a odebírání služeb a funkcionalit v souladu s konkrétními potřebami uživatelů. Hlavní úlohou aplikačního serveru je zpracovávání požadavků od klientů, manipulace s daty a poskytování výstupů v reálném čase. Aplikační server má klíčový význam pro plynulý provoz systému a jeho schopnost snadného rozšiřování kapacity je neocenitelná, zvláště pro organizace s rostoucím počtem uživatelů a náročnými projekty. Díky aplikačnímu serveru mohou týmy efektivněji organizovat svou práci, sdílet informace a spolupracovat na projektech, což v konečném důsledku vede k zvýšení produktivity a dosažení lepších výsledků. [24]

2.2.2 Databázový server

Databázový server funguje jako centrální úložiště pro veškerá data a informace v kolaboračním systému. Zde jsou ukládány uživatelské profily, oprávnění, záznamy o projektech a další klíčové informace. Pro správné fungování celého systému je zásadní rychlost, spolehlivost a bezpečnost databázového serveru. Rychlé vyhledávání dat, snadná údržba a zálohování jsou nezbytné pro plynulý provoz a efektivitu celé platformy. Bezpečnostní opatření jsou také klíčová, aby se minimalizovalo riziko neoprávněného přístupu nebo úniku citlivých informací. Dobře navržený a konfigurovaný databázový server zajišťuje, že týmy mají vždy přístup k aktuálním a relevantním datům, což zvyšuje koordinaci úkolů a umožňuje lepší plánování a řízení projektů. [24]

2.2.3 Webový server

Webový server slouží jako rozhraní mezi uživateli a aplikacemi, které umožňuje přístup k funkcím a nástrojům prostřednictvím standardních webových prohlížečů. Webové servery zajišťují dostupnost uživatelského rozhraní, statických a dynamických prvků, jako jsou texty, obrázky, formuláře a grafy. Klíčovým aspektem webového serveru je jeho rychlost a spolehlivost, aby uživatelé nemuseli čekat na načítání stránek. Bezpečnost je dalším důležitým faktorem, zvláště při přenosu dat mezi uživateli a serverem. Správně navržený a konfigurovaný webový server umožňuje uživatelům snadný a bezpečný přístup k funkcím a nástrojům kolaboračního systému, což podporuje efektivní spolupráci a interakci mezi týmy. [24]

2.3 Git repozitář

Git repozitář představuje další klíčovou komponentu kolaboračního serveru, zejména pro vývojářské týmy. Tato komponenta umožňuje verzování kódu, což je zásadní pro správu a sledování změn ve softwarových projektech. Git repozitář zaznamenává všechny změny provedené na kódu, což umožňuje vývojářům pracovat současně na různých verzích a snadno řešit konflikty. Dále usnadňuje sledování historie projektu a umožňuje vrátit se k předchozím verzím v případě potřeby. Pro vývojářské týmy je Git repozitář klíčovou komponentou, která zvyšuje efektivitu a spolupráci. Díky němu mohou týmy lépe organizovat svou práci, sledovat a řešit změny v kódu a zajišťovat kvalitní a stabilní softwarové produkty. [24]

2.4 Integrace CI/CD s kolaboračním Git serverem a externími nástroji

Integrace kontinuální integrace a nasazování (CI/CD) s kolaboračním Git serverem představuje pro vývojářské týmy řadu výhod. Metodika CI/CD se zaměřuje na automatizaci a optimalizaci procesů spojených s integrací nových změn do softwarového projektu a jejich nasazení do produkčního prostředí. Tímto způsobem je dosaženo rychlejšího a efektivnějšího vývoje softwarových produktů s minimálními chybami. [25]

2.4.1 Nativní podpora CI/CD v kolaboračním Git serveru

Současné kolaborační nástroje nabízejí nativní podporu pro CI/CD, což umožňuje týmům vykonávat celý proces CI/CD přímo v rámci jedné integrované platformy. Týmy nemusí hledat a implementovat další nástroje pro správu CI/CD, což snižuje náročnost konfigurace a údržby. Nativní integrace do kolaboračního Git serveru umožňuje automatizaci spouštění testů, sestavování a nasazován, čímž se eliminují manuální kroky a zrychlí se proces vývoje. [25]

2.4.2 Integrace externích nástrojů pro CI/CD

Kolaborační servery často poskytují API, které umožňuje týmům propojit platformu s externími nástroji a aplikacemi. Tímto způsobem je dosaženo rozšíření funkcionalit platformy a zjednodušení výměny dat a informací mezi různými aplikacemi. [25]

Integrace externích nástrojů pro CI/CD do kolaboračního Git serveru optimalizuje pracovní postupy, snižuje redundanci dat a zvyšuje celkovou efektivitu týmu. Díky této integraci jsou týmy schopny využívat preferované nástroje, které nejlépe vyhovují jejich specifickým potřebám, a dosahovat tak maximálního výkonu a efektivity v procesu týmové spolupráce. [25]

Výhodou této integrace je vytvoření komplexního ekosystému pro vývoj softwaru, který kombinuje přínosy nativní podpory CI/CD v kolaboračním Git serveru s možností využívat specializované nástroje pro CI/CD. Týmy tak mohou dosáhnout urychlení vývojového cyklu, zlepšení kvality softwarových produktů a zvýšení konkurenceschopnosti organizace na trhu. [25]

2.5 Kolaborační server versus Git server

Kolaborační server a Git server mají odlišné zaměření a funkcionality, jež je důležité rozlišovat. Kolaborační server představuje komplexní platformu, která integruje mnoho různých služeb

pro spolupráci, včetně sdílení souborů, projektové správy a dalších. Díky této komplexitě nabízí uživatelům širokou škálu nástrojů, jež usnadňují a zlepšují týmovou spolupráci. Na druhou stranu je Git server specializovaný nástroj, který se zaměřuje především na verzování kódu. Jeho hlavním úkolem je správa repozitáře, kde jsou ukládány všechny změny v kódu. Git server poskytuje výbornou podporu pro vývojářské týmy, kteří potřebují koordinovat a sledovat změny v kódu. Důležité však je poznamenat, že některé kolaborační servery mohou obsahovat také Git repozitář jako součást své nabídky služeb. To umožňuje vývojářským týmům využívat výhody obou typů nástrojů v jediném integrovaném prostředí. [26]

2.6 Výhody kolaboračních serverů

Kolaborační servery přinášejí mnoho výhod, jež výrazně zlepšují efektivitu a produktivitu týmů a organizací. Jednou z nejvýraznějších výhod je zvýšená efektivita. Díky centralizaci nástrojů a informací mají uživatelé vše potřebné na dosah ruky, což eliminuje ztrátu času hledáním a usnadňuje spolupráci. Lepší komunikace mezi členy týmu je další klíčovou výhodou. Centralizace informací zajišťuje, že všichni členové týmu mají přístup k aktuálním dokumentům a projektům. To snižuje riziko duplicitní práce a zlepšuje koordinaci úkolů. Schopnost přizpůsobit se změnám a růstu organizace je další výhodou. Kolaborační servery jsou často navrženy tak, aby byly škálovatelné a flexibilní, což umožňuje růst společně s týmem a adaptaci na nové požadavky. [27]

2.7 Nevýhody kolaboračních serverů

Přestože kolaborační servery nabízejí mnoho výhod, existují také některé nevýhody, jež je důležité zvážit. Integrace a správa celého systému mohou být náročné a vyžadovat odborné znalosti. To může být zvláště problematické pro menší organizace s omezenými zdroji. Potřeba odborného personálu pro údržbu a správu systému může zvýšit náklady na provoz, což může být problém pro menší organizace s omezeným rozpočtem. Vyšší náklady na licencování a údržbu mohou být také překážkou pro některé organizace. Před implementací kolaboračního serveru je důležité pečlivě zhodnotit náklady a přínosy. Riziko závislosti na konkrétním dodavateli nebo technologii může omezit flexibilitu a nezávislost organizace. Změna dodavatele nebo migrace na jinou platformu může být obtížná a nákladná. [27]

2.8 Analýza existujících kolaboračních serverů podporujících Git

V rámci diplomové práce bylo provedeno porovnání nesledujících kolaboračních Git serverů:

- *GitLab,*
- *GitHub,*
- *Bitbucket,*
- *Gitea,*
- *RhodeCode,*
- *Kallithea.*

2.8.1 Definování parametrů pro kolaborační server

Pro výběr kolaboračního serveru byly definovány následující požadavky:

Podpora Webhooks

Webhooks jsou automatizované zprávy odeslané z jedné aplikace do jiné v reakci na nějakou událost. Jsou zpravidla používány k propojení dvou různých systémů a informování jednoho systému o události v jiném. Kolaborační server s podporou webhooks může efektivně integrovat s externími nástroji, jako jsou systémy pro sledování chyb, CI/CD nástroje nebo dokonce CRM platformy.

Vytváření skupin uživatelů

Skupiny uživatelů umožňují efektivní a centralizovanou správu přístupových práv v rámci systému. Administrátoři mohou definovat různé úrovně oprávnění pro různé skupiny, což usnadňuje řízení toho, co mohou jednotlivé skupiny vidět, upravovat nebo vytvářet.

Vytvářet skupiny uživatelů a přiřazovat je do projektů

Pokročilá správa přístupů umožňuje nejen vytvářet skupiny uživatelů, ale také je specifikovat pro určité projekty. To je klíčové pro organizace s více týmy pracujícími na různých projektech, kde není žádoucí, aby všechny skupiny měly přístup ke všem informacím.

Lokální hosting v Docker kontejneru

Docker je platforma pro vývoj, dopravu a běh aplikací v kontejnerech. Kontejnery umožňují aplikacím běžet konzistentně v různých prostředích, izolovaně od ostatních aplikací.

Kolaborační server podporující Docker může být snadno nasazen v jakémkoliv prostředí, které podporuje Docker, což zvyšuje flexibilitu a snižuje potřebu správy.

Automatizované testování CI/CD s možností spouštět testy v docker kontejneru

CI/CD (Continuous Integration/Continuous Deployment) se vztahuje k praxi integrace kódu do hlavního repozitáře a jeho automatického nasazení. Spuštění testů v Docker kontejneru zajišťuje konzistentní testovací prostředí.

Využití více agentů pro paralelní testování více projektů

Většina CI/CD řešení využívá agentů – malých nástrojů, které spouštějí testovací skripty a nasazovací procesy. Schopnost využívat více agentů pro paralelní testování zkracuje dobu potřebnou k otestování většího počtu úprav nebo projektů.

Open-source

Open-source software je licencován tak, aby jeho zdrojový kód byl volně dostupný veřejnosti. To umožňuje nejen upravování kódu podle individuálních potřeb, ale také podporuje transparentnost a komunitní přístup k vývoji.

Licence

Některé open-source projekty jsou zcela zdarma, zatímco jiné mohou vyžadovat placení za prémiové funkce nebo podporu. Možnost využívat software zdarma může významně snížit náklady, ale je důležité pečlivě zvážit celkovou hodnotu produktu, včetně dostupnosti podpory a aktualizací.

2.9 Porovnání kolaboračních serverů

V rámci této práce byla provedena důkladná a komplexní analýza vybraných kolaboračních Git serverů s primárním cílem identifikovat optimální platformu pro automatické vyhodnocování studentských prací v akademickém prostředí. Provedená analýza byla založena na již předem definovaném seznamu Git serverů a specifikovaných požadavcích, které jsou klíčové pro úspěšnou realizaci automatického vyhodnocování studentských prací.

Průběh analýzy zahrnoval pečlivé zkoumání jednotlivých kolaboračních serverů a jejich funkcionalit s ohledem na specifické požadavky týkající se automatického vyhodnocování studentských prací. Během této analýzy byly sledovány klíčové aspekty, jako je podpora Webhooků pro sledování notifikací o událostech, schopnost vytvářet skupiny uživatelů a projektů pro efektivní organizaci prací, možnost lokálního hostování v kontejneru pro dosažení vyšší úrovně kontroly a bezpečnosti, automatizované testování prostřednictvím CI/CD procesů s využitím technologie Docker, možnost využití více runnerů pro zvýšení efektivity při vyhodnocování více studentských prací současně, a také zda se jedná o open-source platformu s možností přizpůsobení a rozšiřitelnosti.

Výsledky této analýzy jsou prezentovány přehlednou tabulkou, kde jsou uvedeny jednotlivé kolaborační servery spolu s informacemi o tom, zda podporují nebo nepodporují jednotlivé definované požadavky. Tato tabulka poskytuje užitečný přehled o podpoře jednotlivých kolaboračních serverů pro klíčové požadavky na automatické vyhodnocování studentských prací.

Na základě této podrobné analýzy byl následně vybrán nejvhodnější kolaborační server, který nejlépe odpovídá specifickým potřebám akademického prostředí a zaručuje spolehlivé a efektivní vyhodnocování studentských prací.

Tabulka 1 – Analyzované kolaborační servery – přehled funkcí [22, 28, 29, 30, 31, 32]

Kolaborační server	**Podpora Webhooks**	**Skupiny uživatelů**	**Skupiny pro projekty**	**Lokální hosting v Docker kontejneru**	**CI/CD (podpora Docker)**	**Podpora více runnerů**	**Open-source**	**Bezplatná verze**
GitLab	Ano	Ano	Ano	Ano	Ano	Ano	Ano	Ano
GitHub	Ano	Ano	Ano	Ne	Ano	Ano	Ne	Ano
Bitbucket	Ano	Ano	Ano	Ano	Ano	Ano	Ne	Ano
Gitea	Ano	Ano	Ano	Ano	Ne	Ne	Ano	Ano
RhodeCode	Ano	Ano	Ano	Ne	Ne	Ne	Ano	Ano
Kallithea	Ano	Ano	Ano	Ano	Ne	Ne	Ano	Ano

2.9.1 Výběr kolaboračního serveru

V rámci analýzy vybraných kolaboračních serverů s ohledem na automatické vyhodnocování studentských prací, se ukázalo, že GitLab je nejvhodnějším a nejperspektivnějším řešením pro tuto potřebu v akademickém prostředí.

Jedním z klíčových faktorů, který přispěl k tomuto závěru, je schopnost GitLabu vytvářet skupiny uživatelů a projektů. Tato možnost umožňuje efektivní organizaci práce, a to jak na úrovni jednotlivých uživatelů, tak na úrovni celých projektových týmů. Díky této funkci lze přesně řídit přístupová práva, což zajišťuje bezpečnost dat a usnadňuje spolupráci mezi členy týmu.

Dalším důležitým parametrem, který GitLab nabízí, je lokální hosting v kontejneru. Tato vlastnost umožňuje institucím mít plnou kontrolu nad svými daty a provozovat GitLab v prostředí, které je specificky přizpůsobeno jejich potřebám a bezpečnostním požadavkům. Tímto způsobem lze lépe kontrolovat a zabezpečit citlivá data a informace.

GitLab rovněž vyniká podporou automatizovaného testování CI/CD s využitím Dockeru. Díky této funkci lze snadno a rychle provádět procesy testování, sestavování a nasazování aplikací. To výrazně zrychluje dodávání nových funkcí a oprav chyb, což má klíčový vliv na efektivitu vývoje a vyhodnocování studentských prací.

Škálovatelnost a možnost využití více runnerů jsou dalšími výhodami GitLabu, které jsou zásadní pro akademické prostředí. S možností vytvářet a spravovat více runnerů lze provádět paralelní testování více projektů současně, což výrazně zlepšuje efektivitu vyhodnocování studentských prací a zkracuje dobu dodání výsledků.

Jedním z klíčových důvodů, které potvrzují vhodnost GitLabu pro automatické vyhodnocování studentských prací, je jeho open-source charakter. To znamená, že instituce mají možnost přizpůsobit a rozšířit GitLab dle svých potřeb a požadavků. Tato flexibilita umožňuje snadnou úpravu a rozšiřitelnost systému podle specifických požadavků akademické instituce.

Důležité je rovněž zmínit, že GitLab disponuje širokou a aktivní komunitou uživatelů a vývojářů. Tato komunita poskytuje cenné zkušenosti, informace, návody a osvědčené postupy, což zajišťuje dostupnost podpory a možnost sdílení znalostí. GitLab je dnes velmi používaným kolaboračním serverem a jeho popularita je stále rostoucí, což potvrzuje jeho spolehlivost a kvalitu jako platformy pro správu verzí, kolaboraci a automatizaci vývoje.

3 PROGRAMOVÉ VYBAVENÍ

V této kapitole je uveden popis jednotlivých služeb nezbytných pro fungování systému pro automatizované vyhodnocování studentských prací s využitím Git.

3.1 Operační systém

Pro automatizované vyhodnocování studentských prací s využitím Git je zvolen Linuxový operační systém Ubuntu, který je poskytován zdarma a umožňuje úpravy v jeho zdrojových souborech. Ubuntu je založeno na distribuci Debian a díky tomu dědí rozsáhlou kolekci softwaru. Jeho klíčovou výhodou pro tento účel je pravidelné vydávání verzí s dlouhodobou podporou označovaných jako LTS (Long Term Support) každé dva roky. Tyto LTS verze nabízejí 5letou podporu, během které jsou uživatelé zajištěni bezpečnostními a systémovými aktualizacemi. Díky stabilitě a spolehlivosti se Ubuntu stává ideálním operačním systémem pro provoz automatizovaného vyhodnocování studentských prací s využitím Git. [33]

3.2 Gitlab

GitLab je webový kolaborační nástroj pro správu verzí a projektů založených na systému Git. Jedná se o open-source platformu, která umožňuje spolupracovat na vývoji softwarových projektů, sdílet kód, řízení úkolů a sledovat změny v kódu. GitLab nabízí uživatelům širokou škálu funkcí, včetně správy verzí, problémů a požadavků na změny, code review, správy uživatelů a přístupových práv, automatizace kontinuální integrace a nasazení (CI/CD) s využitím Dockeru, a mnoho dalších. Díky tomu je GitLab oblíbeným nástrojem mezi vývojáři a týmy, kteří chtějí efektivně a bezpečně spolupracovat na svých projektech a zvýšit produktivitu vývoje softwaru. [7, 8]

3.3 Gitlab runner

GitLab Runner je open-source nástroj navržený jako součást platformy GitLab pro automatizaci a provádění kontinuální integrace a nasazení (CI/CD) projektů. Slouží k vykonávání definovaných úloh (jobů) v reakci na změny v repozitáři nebo na základě definovaných pravidel. GitLab Runner umožňuje efektivní sestavování, testování a nasazování kódu v různých prostředích, včetně Docker kontejnerů pro izolaci a opakovatelnost testovacího prostředí. Díky schopnosti pracovat s více runnery umožňuje paralelní provádění úloh a zajišťuje rychlejší a spolehlivější vývoj softwaru. [34]

3.4 MySQL Server

MySQL server je relační open-source databázový systém, který umožňuje pomocí SQL příkazů ukládat, zpracovávat, získávat a spravovat data v databázi. Každá databáze obsahuje tabulky s definovanými sloupci a řádky, kde jsou uloženy záznamy odpovídajícího datového typu. Pro sítovou komunikaci využívá protokol TCP a standardní port 3306. MySQL server tak slouží jako cenný nástroj pro organizaci a manipulaci s daty ve spolehlivých relačních databázích. [35]

3.5 H2

H2 je open-source relační databázový systém napsaný v jazyce Java, který je navržen pro použití jako embeddable databáze nebo jako databáze pro lehké webové aplikace. Jedná se o malý, rychlý a plně přenosný databázový systém, který podporuje standardní SQL syntaxi a nabízí možnost provozovat databázi v paměti nebo na disku. H2 je oblíbený pro svou snadnou integraci do aplikací, jednoduché nasazení a nízké nároky na zdroje, což ho činí ideální volbou pro vývojáře, kteří potřebují malou a efektivní databázi pro své projekty. [36]

3.6 Řídící aplikace

3.6.1 Volba Jazyka

Řídící aplikace je dle zadání webovou aplikací, pro kterou bylo nutné vybrat vhodné programovací jazyky. Pro backendovou část byl zvolen programovací jazyk Java a pro frontendovou část byl vybrán JavaScript. Oba tyto jazyky byly zvoleny, jelikož byly probrány v rámci studia.

3.6.2 Volba frameworků

Spring boot

Spring Boot je moderní framework pro vývoj webových aplikací v programovacím jazyce Java. Jedná se o open-source řešení založené na Spring Frameworku, které usnadňuje a urychluje proces vývoje aplikací tím, že poskytuje před konfigurované a integrované nástroje. Spring Boot eliminuje potřebu ruční konfigurace a nastavení, což umožňuje vývojářům se soustředit na samotnou implementaci funkcionality. Díky jeho schopnosti vytvářet samostatné spustitelné aplikace s integrovaným webovým serverem je nasazení aplikací snadné a jednoduché. Spring

Boot podporuje širokou škálu funkcí, jako je správa závislostí, integrace s databázemi, zabezpečení a tvorba RESTful API, což jej činí atraktivním nástrojem pro vývoj webových aplikací a služeb. [37]

Swagger

Swagger, nyní známý jako OpenAPI, je open-source framework pro návrh, dokumentaci a testování API (Application Programming Interface). Jedná se o mocný nástroj, který umožňuje vývojářům snadno a přesně definovat rozhraní API, které jejich aplikace poskytují ostatním programům a službám. Swagger používá pro specifikace API formát založený na YAML nebo JSON, což umožňuje vytvářet srozumitelnou a automaticky vytvářející dokumentaci. Díky tomu, že Swagger generuje interaktivní dokumentaci pro API, poskytuje uživatelům jednoduché a přehledné prostředí pro vyzkoušení a testování jednotlivých koncových bodů API rozhraní. To zvyšuje efektivitu vývoje, zlepšuje spolupráci mezi týmy a usnadňuje pochopení funkcionality API pro klienty a vývojáře, což je klíčovým prvkem moderního a dobře navrženého softwarového ekosystému. [38]

Hibernate

Hibernate je open-source objektově-relační mapovací (ORM) framework pro jazyk Java, který usnadňuje interakci s databází a správu datových modelů. Tento nástroj umožňuje vývojářům pracovat s relačními databázemi prostřednictvím objektově orientovaného přístupu, což eliminuje potřebu psát přímé SQL dotazy. Hibernate mapuje Java objekty na relační tabulky v databázi a poskytuje abstrakci nad databázovými operacemi, což zjednodušuje a urychluje vývoj aplikací. Díky Hibernate mohou vývojáři snadno a efektivně pracovat s daty, provádět CRUD operace (vytváření, čtení, aktualizaci a mazání) a zajišťovat konzistenci dat mezi aplikací a databází, což přispívá k robustnímu a efektivnímu chodu webových a enterprise aplikací. [39]

3.6.3 Volba knihoven

React

React je moderní open-source JavaScriptová knihovna vyvinutá společností Facebook, která se zaměřuje na tvorbu uživatelských rozhraní pro webové aplikace. Základním principem Reactu je použití komponent, což jsou znovupoužitelné a nezávislé stavební bloky, které tvoří uživatelské rozhraní. React využívá virtuální DOM (Document Object Model) pro efektivní aktualizaci a renderování změn, což vede k vyšší efektivitě a rychlosti aplikace. S bohatým

ekosystémem knihoven a nástrojů, které poskytuje, je dnes React hojně využíván pro vývoj moderních a interaktivních webových aplikací s vysokým výkonem a uživatelskou přívětivostí. [40]

Material UI

Material-UI je populární open-source knihovna pro tvorbu uživatelských rozhraní (UI) v jazyce JavaScript a knihovnou React. Vyvinuta společností Google, Material-UI implementuje designový systém Material Design, který zdůrazňuje moderní, čistý a intuitivní vzhled aplikací. Knihovna poskytuje bohatou sadu předdefinovaných a stylizovaných komponent, jako jsou tlačítka, formulářová pole, navigační prvky a další, které umožňují rychlé a snadné sestavování a tvorbu uživatelských rozhraní. Díky své modularitě a možnosti přizpůsobení je Material-UI oblíbenou volbou pro vývojáře, kteří hledají moderní a profesionální vzhled svých webových aplikací. [41]

Lombok

Lombok je open-source knihovna pro jazyk Java, která slouží k redukci opakujícího se kódu a zjednodušení tvorby Java bean tříd a datových modelů. Lombok umožňuje vkládání tzv. anotací do kódu, které automaticky generují gettery, settery, konstruktory, metody equals, hashCode a další potřebné metody na základě definice atributů třídy. Tímto způsobem Lombok značně snižuje potřebu manuálního psaní rutinního kódu a zvyšuje produktivitu vývojářů. Díky jednoduché integraci a využití, Lombok se stal oblíbeným nástrojem pro vývoj aplikací v jazyce Java, kde přispívá k čistějšímu a efektivnějšímu kódu. [42]

Gitlab4j-api

Gitlab4j-api je open-source Java knihovna, která slouží k jednoduché integraci a interakci s GitLab API. Jako Java knihovna poskytuje vývojářům širokou škálu funkcí a nástrojů pro práci s GitLabem, což je webová platforma pro správu verzí a kolaborativní vývoj. Tato knihovna umožňuje snadnou automatizaci operací, jako jsou získávání informací o uživatelích, projektech a repozitářích, vytváření a zavírání merge requests, správu issue a dalších činností souvisejících s verzováním a správou projektů. S využitím Gitlab4j-api mohou vývojáři efektivně pracovat s GitLabem přímo z jazyka Java, což zlepšuje jejich produktivitu a umožňuje snadnou integraci GitLabu do svých aplikací. [43]

JJWT

JJWT (JSON Web Token for Java) je open-source knihovna pro jazyk Java, která umožňuje snadnou tvorbu, ověřování a správu JSON Web Tokenů (JWT). JWT je standardem pro přenos

dat mezi dvěma stranami v zabezpečené formě jako JSON objekty. JJWT poskytuje implementaci specifikací JWT, včetně podpory pro různé algoritmy šifrování a podepisování. Díky této knihovně mohou vývojáři snadno vytvářet a ověřovat JWT tokeny, což je užitečné pro autentizaci a autorizaci uživatelů v moderních webových aplikacích a službách s vysokými nároky na bezpečnost a důvěrnost dat. [44]

3.6.4 Volba nástrojů

Maven

Maven je open-source nástroj pro správu projektů a automatizaci sestavování softwarových aplikací v jazyce Java. Jako nástroj založený na konceptu řízení závislostí, Maven usnadňuje správu externích knihoven a modulů, což minimalizuje ruční konfiguraci a zvyšuje opakovatelnost vývojových procesů. S využitím souboru konfigurace (pom.xml) definuje projektovou strukturu, cíle sestavování, závislosti a další parametry, což umožňuje konzistentní a efektivní práci vývojových týmů. Maven automatizuje sestavení projektů, kontrolu kvality kódu, generování dokumentace a správu testů, což vede k vyšší produktivitě a spolehlivosti vývoje. Díky svojí popularitě a komunitní podpoře se Maven stal klíčovým nástrojem pro profesionální vývoj v jazyce Java a je součástí mnoha moderních vývojových prostředí a integrovaných vývojových prostředí (IDE). [45]

4 KONTEJNEROVÁ VIRTUALIZACE DOCKER

V této kapitole je popsáno základní seznámení s technologií Docker.

4.1 Docker

Docker je open-source platforma pro kontejnerizaci aplikací, která umožňuje vývojářům zabalení aplikací včetně jejich závislostí do izolovaných a snadno přenosných kontejnerů. Kontejnery obsahují všechno potřebné pro spuštění aplikace, včetně kódu, runtime, knihoven a dalších závislostí. Díky kontejnerizaci je možné aplikace spouštět bezproblémově na různých prostředích, bez ohledu na konfiguraci hostitelského systému. [46]

Docker využívá konceptu kontejnerové virtualizace, která umožňuje izolaci jednotlivých aplikací a procesů od ostatních aplikací na stejném systému. To zajišťuje vyšší bezpečnost, spolehlivost a výkon aplikací. Docker poskytuje snadné a jednotné rozhraní pro vytváření, správu a sdílení kontejnerů, což usnadňuje vývojový proces a zvyšuje efektivitu týmů. [46]

4.1.1 Docker kontejner

Docker kontejner je izolovaná a samostatná instance Docker image. Jedná se o prostředí pro aplikaci, které má oddělený souborový systém, procesy, síť a další zdroje od hostitelského systému. Každý kontejner má svůj unikátní identifikátor a využívá společné jádro s hostitelským systémem, což umožňuje efektivní využití systémových prostředků. [46]

4.1.2 Docker image

Docker image je read-only šablona, ze které se vytvářejí Docker kontejnery. Obsahuje všechny komponenty potřebné pro spuštění aplikace, včetně kódu, knihoven, konfigurace a dalších závislostí. Docker image je sestaven podle specifikací uvedených v Dockerfile. Image je jednotný a přenosný balík, který lze snadno sdílet a nasazovat na různých prostředích, což usnadňuje vývoj a správu aplikací. [46]

4.1.3 Dockerfile

Dockerfile je textový soubor obsahující seznam instrukcí pro sestavení Docker image. Tyto instrukce definují, jaké kroky mají být provedeny při vytváření image, včetně kopírování souborů, instalace závislostí, konfigurace prostředí a dalších úkonů. Dockerfile umožňuje

opakovatelné sestavování image, což zajišťuje konzistenci a spolehlivost procesu sestavování. [46]

4.1.4 Docker compose

Docker Compose je nástroj pro definování a správu více kontejnerových aplikací. Umožňuje jednoduše a deklarativně zadat konfiguraci všech služeb, které tvoří aplikaci, včetně jejich závislostí a konfigurace prostředí. Docker Compose umožňuje snadno spouštět, zastavovat a spravovat celou aplikaci s jejími kontejnery jedním příkazem. Tím usnadňuje a zefektivňuje proces nasazování a vývoje více kontejnerových aplikací. [47]

4.2 Automatizované vyhodnocování studentských prací s využitím Git a nástroje Docker

V této kapitole jsou popsány jednotlivé výhody a nevýhody spojené s využitím nástroje Docker pro automatizované vyhodnocování studentských prací s využitím Git.

4.2.1 Výhody využití nástroje Docker

Izolace

Docker poskytuje izolaci prostředí, což umožňuje testování jednotlivých studentských prací nezávisle na sobě. Každá práce běží ve vlastním kontejneru, který je oddělen od ostatních. To minimalizuje riziko, že by chyby v jedné práci mohly negativně ovlivnit ostatní, což zajišťuje bezpečnější a spolehlivější prostředí pro vývoj. [46]

Konfigurovatelnost

S využitím Docker obrazů lze snadno vytvořit předem nakonfigurovaná prostředí obsahující všechny potřebné nástroje, knihovny a závislosti. To znamená, že každý student může mít stejné podmínky pro testování své práce, bez ohledu na to, na jakém systému pracuje. [46]

Přenositelnost

Docker kontejnery jsou přenositelné mezi různými platformami a systémy, což umožňuje snadný přechod mezi různými prostředími. Pokud je potřeba přesunout studentské práce na jiný server, cloudovou platformu nebo pracovní stanici, stačí přesunout Docker kontejnery, aniž by bylo nutné znovu konfigurovat celé prostředí. To usnadňuje správu infrastruktury a zvyšuje flexibilitu. [46]

Reprodukovatelnost

Docker přináší reprodukovatelnost pro systémy tím, že izoluje aplikaci a její závislosti od hostitelského systému. To zajišťuje, že po každém spuštění s identickými daty obdržíme vždy stejný výsledek. [46]

4.2.2 Nevýhody použití nástroje Docker

Složitost

Docker je poměrně komplexní nástroj a jeho správa vyžaduje specifické znalosti. Učení se Dockeru může být náročné, zejména pro začátečníky, a může vyžadovat časové investice a školení, aby byl správně a efektivně využit. [48]

Výkon

Docker přináší do systému určitou míru režie vzhledem k jeho základní architektuře, kdy každý kontejner operuje ve svém izolovaném prostředí. Při potřebě spouštět velké množství kontejnerů nebo v případě omezených systémových prostředků může tato izolace negativně ovlivnit celkový výkon systému. Je zásadní provést adekvátní optimalizaci a konfiguraci, tak aby byl minimalizován dopad této režie na systémový výkon a byl zajištěn co nejefektivnější provoz kontejnerizovaných aplikací. [48]

Bezpečnost

I když jsou Docker kontejnery izolované, existuje teoretická možnost, že pokud je některý z kontejnerů napaden, může to ovlivnit celý hostitelský systém. Správná konfigurace a bezpečnostní opatření jsou nezbytné pro minimalizaci rizika. [48]

Údržba

Každý Docker kontejner vyžaduje pravidelnou údržbu, aktualizace a správu, aby byl bezpečný a efektivní. To znamená, že je nutné sledovat nové verze a opravy bezpečnostních chyb, což může vyžadovat dodatečné úsilí a zdroje. [48]

5 ANALÝZA

5.1 Analýza problému

Cílem této práce je navrhnout a implementovat systém, který umožní automatizované vyhodnocování studentských prací v prostředí kolaboračního Git serveru. Systém bude sloužit k poskytnutí zpětné vazby studentům na jejich odevzdané práce a usnadní vyučujícím proces hodnocení studentů.

Jedním z hlavních cílů je automatizované vyhodnocování studentských prací. To znamená, že po nahrání studentské práce na Git server bude systém schopen automaticky spustit definovanou sadu testů na této práci a následně vyhodnotit jejich výsledky. Díky tomu budou studenti získávat okamžitou zpětnou vazbu na svou práci, což je klíčové pro jejich učení a zlepšování se.

Dalším důležitým aspektem je dostupnost výsledného hodnocení. Systém by měl umožnit jak studentům, tak vyučujícím přístup k výsledkům hodnocení. Studenti by měli mít možnost vidět své hodnocení a případné chyby ve své práci, což jim pomůže lépe pochopit své chyby a zlepšit se. Vyučující by měli mít přehledné a snadno dostupné výsledky za všechny definované cvičení či skupiny, což usnadní správu a hodnocení studentů.

Posledním požadavkem je také podpora pro programovací jazyky Java a C#. Pro musí být implementovaný systém schopen spouštět a vyhodnocovat studentské práce napsané v těchto jazycích.

5.2 Existující systémy pro automatizované vyhodnocování studentských prací s využitím Git

V oboru informačních a komunikačních technologií (ICT) ve vzdělávání jsme v posledních letech svědky narůstající tendence implementovat nástroje, jež zefektivňují pedagogické procesy skrze automatizaci. Charakteristickým rysem této evoluce je integrace verzovacích systémů, zejména Git, do vzdělávacího prostředí. Předním zástupcem tohoto trendu je GitHub Classroom. [49]

GitHub Classroom, poskytovaný renomovanou společností GitHub, je reprezentativním příkladem SAAS (Software as a Service). Jeho primární zaměření je na efektivní správu a distribuci úkolů studentů, a to s hlubokou integrací do mateřské platformy GitHub. Klíčovými aspekty jsou jeho schopnosti sledovat změny v repozitářích studentů díky gitu a automatizovat

správu těchto repozitářů na základě pedagogických požadavků. Důraz je kladen na bezpečnost, s politikami a protokoly z mateřského GitHubu, což zajišťuje bezpečné prostředí pro všechny zúčastněné. Navíc, optimalizovaná uživatelská interakce usnadňuje přístup široké škále uživatelů, včetně těch s omezeným znalostním základem v oblasti Gitu. [49]

I když GitHub Classroom nabízí řadu předností, vlastní softwarové řešení přináší své specifické výhody. Především umožňuje neomezenou flexibilitu, poskytujíc adaptaci na konkrétní potřeby a požadavky instituce. Zároveň redukuje závislost na externích poskytovatelích, což může v praxi znamenat vyšší stabilitu a autonomii. Dále, vlastní řešení může být hluboce integrováno s ostatními interními systémy a databázemi v daném vzdělávacím kontextu. Kromě toho, možnost implementovat specifické funkcionality dává institucím unikátní příležitost vyplnit mezery, které standardní platformy nemohou pokrýt. [49]

5.3 Workflow aplikace

1. Vyučující registruje studenty

   Před zahájením semestru zadává vyučující do systému seznam studentů. Registrace může zahrnovat unikátní identifikátory, e-maily a další relevantní informace pro identifikaci studentů. Tím se zároveň studentům umožní přístup k řídící aplikaci a GitLabu.
2. Vyučující zakládá předměty

   Po registraci studentů vytváří vyučující v systému nové předměty, přičemž definuje název i popis předmětu.
3. Vyučující tvoří skupiny studentů

   V závislosti na potřebách a struktuře kurzu může vyučující vytvořit skupiny studentů, což umožní lepší organizaci, např. pro týmové projekty či dělení dle cvičení.
4. Vyučující definuje projekty pro jednotlivé předměty

   Pro každý předmět vytváří vyučující několik projektů, které mohou být například malé aplikace v jazycích Java či C#. Při vytváření úkolu specifikuje vyučující, jaká na kolik procent musí být práce splněna.
5. Vyučující přiřazuje studenty k projektům

   U jednotlivých projektů může vyučující přiřadit specifické skupiny studentů, takže například ne všichni studenti mají stejné zadání, pokud docházejí na různá cvičení.
6. Aplikace pro každého studenta vytvoří individuální fork projektu

   Systém automaticky vytváří fork hlavního repozitáře pro každého studenta ve skupině, čímž má každý svůj vlastní repozitář pro odevzdání řešení.

7. Student získává z systému informace pro klonování projektu

   Systém poskytuje studentům instrukce, jak si klonovat repozitář na svůj počítač, což jim umožní pracovat lokálně na svém řešení.
8. Student řeší úlohu a odevzdává vypracované řešení pomocí Git:

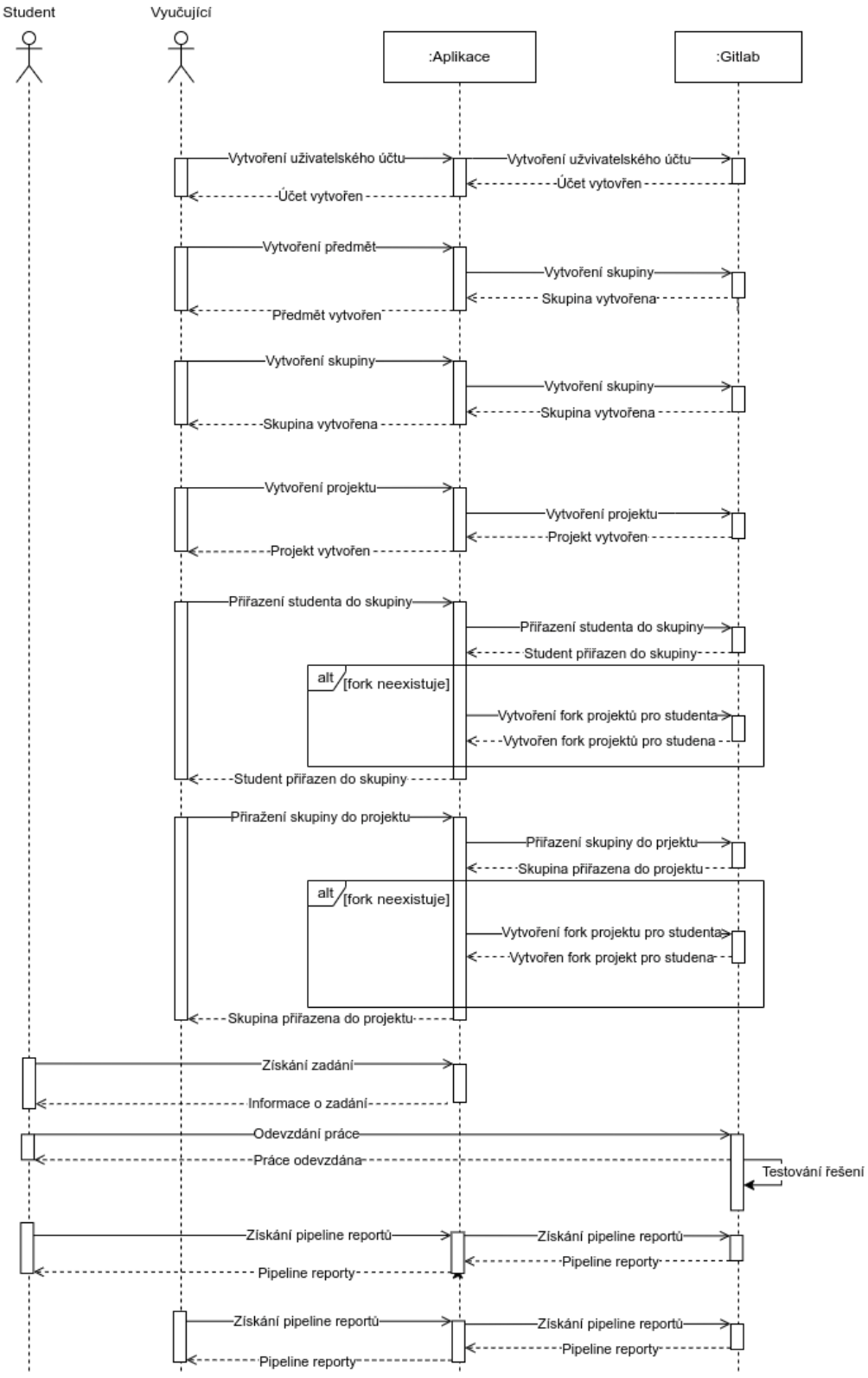
   Po vypracování úkolu použije student příkazy verzovacího nástroje Git pro odevzdání svého řešení. Odevzdané řešení obsahuje všechny upravené zdrojové kódy.
9. Systém automaticky otestuje odevzdané řešení

   Jakmile student nahraje své řešení, systém automaticky spustí řadu testů, které kontrolují kvalitu a správnost kódu podle předdefinovaných kritérií.
10. Aplikace informuje studenta o výsledcích testů

    Student je informován o tom, zda jeho kód prošel všemi testy, jaká byla procentuální úspěšnost jeho řešení a které testy byly splněny a které ne. Zároveň se dozví, zda splnil danou úlohu. Pokud ne, může zjistit, na kterých částech potřebuje zapracovat, a může odevzdat upravené řešení.
11. Vyučující získává z aplikace informace o postupu studentů

    V průběhu semestru či po uplynutí deadlineů získává vyučující přehled o řešeních jednotlivých studentů. Aplikace vyučujícímu poskytne informace o tom, kteří studenti splnili úkoly, jaká byla kvalitní jejich řešení.

Výše bylo popsáno workflow interakce mezi studentem, vyučujícím, aplikací a nástrojem GitLab. Toto workflow vysvětluje, jak jednotliví aktéři vzájemně spolupracují a ovlivňují se v rámci definovaného procesu. Důležitým doplňkem k textovému popisu je obrázek 2, na kterém je zobrazeno workflow ve formě sekvenčního diagramu. Diagram graficky znázorňuje tok informací a akcí mezi všemi aktéry: studentem, vyučujícím, aplikací a GitLabem.



Obrázek 2 – Sekvenční diagram znázorňující workflow aplikace a aktérů (zdroj: vlastní)

5.4 Definování požadavků

V této kapitole se zaměříme na proces definování požadavků pro projekt. Definování požadavků je klíčovým krokem v celém vývojovém cyklu softwarového projektu. Požadavky představují specifikace toho, co očekáváme, že software bude dělat, jakým způsobem to bude dělat a jaké vlastnosti by měl mít.

5.4.1 Funkční požadavky

- FP 1 – Správa uživatelských účtů: Tato část systému je odpovědná za vytváření a správu účtů studentů a vyučujících, přičemž lze určit různé role a oprávnění.
    - FP 1.1 – Vytváření a správa účtů: Umožňuje administrovat uživatelské profily a přiřazovat jim specifické role, jako jsou student nebo vyučující.
    - FP 1.2 – Organizace skupin a cvičení: Nabízí nástroje pro tvorbu a řízení skupin studentů, definování cvičení a jejich přiřazování ke konkrétním skupinám.
- FP 2 – Sledování změn v repozitářích: Tato funkce integruje systém s Git serverem a sleduje aktivity v repozitářích studentů.
    - FP 2.1 – Integrace s Git Serverem: Propojení systému s vybraným Git serverem pro kontinuální sledování změn v repozitářích studentů.
    - FP 2.2 – Detekce nových verzí: Systém automaticky zaznamenává odevzdání nových verzí studentských prací pro následné zpracování.
- FP 3 – Definování sady testů: Tato sekce se věnuje tvorbě a správě testů, které jsou použity k ověření studentských prací.
    - FP 3.1 – Vytváření testovacích scénářů: Umožňuje definovat a nastavit testy s očekávanými výstupy pro různé programovací jazyky, poskytující univerzální nástroj pro hodnocení kódu.
- FP 4 – Automatizované testování studentských prací: Tato část se zaměřuje na automatické spuštění testů nad odevzdanými pracemi.
    - FP 4.1 – Automatické spuštění testů: Po odevzdání práce systém automaticky spustí předdefinované testy, šetří tak čas vyučujících.
    - FP 4.2 – Vyhodnocení testů: Systém analyzuje výsledky testů a poskytuje přehlednou zprávu o výsledcích.

- FP 5 – Vyhodnocování studentských prací: Zajišťuje prezentaci výsledků a bodové hodnocení studentů.
    - FP 5.1 – Prezentace výsledků: Detailní zobrazení výsledků testů, včetně informace o úspěšných a neúspěšných pokusech.
    - FP 5.2 – Hodnocení studentských prací: Umožňuje vyučujícím vyhodnotit splnění testů u jednotlivých projektů a sledovat celkový výkon studentů během semestru.

5.4.2 Nefunkční požadavky

- NFP 1 – Bezpečnost: Tento požadavek se zaměřuje na zajišťování bezpečného přístupu a ochrany citlivých dat.
    - NFP 1.1 – Bezpečná autentizace: Implementace silných bezpečnostních protokolů pro ověření identity studentů i vyučujících při přístupu do systému.
    - NFP 1.2 – Ochrana studentských dat: Zabezpečení přístupu k studentským pracím a osobním datům tak, aby byly chráněny před neoprávněným přístupem.
- NFP 2 – Výkon a škálovatelnost: Zaměřuje se na schopnost systému zpracovávat velké objemy dat a přizpůsobit se různým provozním podmínkám.
    - NFP 2.1 – Efektivní zpracování prací: Garance rychlého zpracování a vyhodnocení velkého množství odevzdaných prací.
    - NFP 2.2 – Rychlá zpětná vazba: Poskytování promptní zpětné vazby studentům po odevzdání a testování prací.
- NFP 3 – Dostupnost a spolehlivost: Zaměření na stálý a spolehlivý provoz systému.
    - NFP 3.1 – Vysoká dostupnost: Udržování systému v provozu, s minimálními výpadky, aby byl vždy přístupný studentům a vyučujícím.
    - NFP 3.2 – Minimalizace přerušení: Implementace opatření pro rychlé obnovení provozu v případě neplánovaných přerušení nebo výpadků.
- NFP 4 – Podpora programovacích jazyků: Tento požadavek se vztahuje k schopnosti systému testovat kód napsaný v různých programovacích jazycích.
    - NFP 4.1 – Podpora různých programovacích jazyků: Systém musí podporovat testování v různých programovacích jazycích, jako jsou Java, C#, atd., což umožňuje větší rozmanitost ve výuce.

- NFP 5 – Uživatelské rozhraní: Zajištění přívětivého a intuitivního uživatelského rozhraní.
    - NFP 5.1 – Přívětivé rozhraní: Design, který usnadňuje studentům i vyučujícím orientaci v systému a jeho používání.
    - NFP 5.2 – Přehledné zobrazení výsledků: Grafické zobrazení výsledků testování a hodnocení, které je snadno srozumitelné.
- NFP 6 – Flexibilita a rozšiřitelnost: Schopnost systému přizpůsobit se změnám a novým požadavkům.
    - NFP 6.1 – Adaptace na nové požadavky: Možnost snadného rozšíření nebo úpravy systému dle nových potřeb a cílů.
- NFP 7 – Podpora více uživatelů: Zajištění, že systém může obsluhovat více uživatelů současně, aniž by to vedlo ke snížení výkonu.
    - NFP 7.1 – Paralelní využití systému uživateli: Implementace technologií a postupů, které umožňují hladkou obsluhu více uživatelů najednou, bez negativního dopadu na kvalitu služby.
- NFP 8 – Podpora různých typů testů: Rozšiřuje možnosti testování poskytováním různých typů testů.
    - NFP 8.1 – Různé typy testů: Systém nabízí možnost vytvářet různé kategorie testů, jako jsou jednotkové, integrační atd., což umožňuje komplexní ověření studentských prací.

5.5 USE-CASE DIAGRAM ŘÍDÍCÍ APLIAKCE

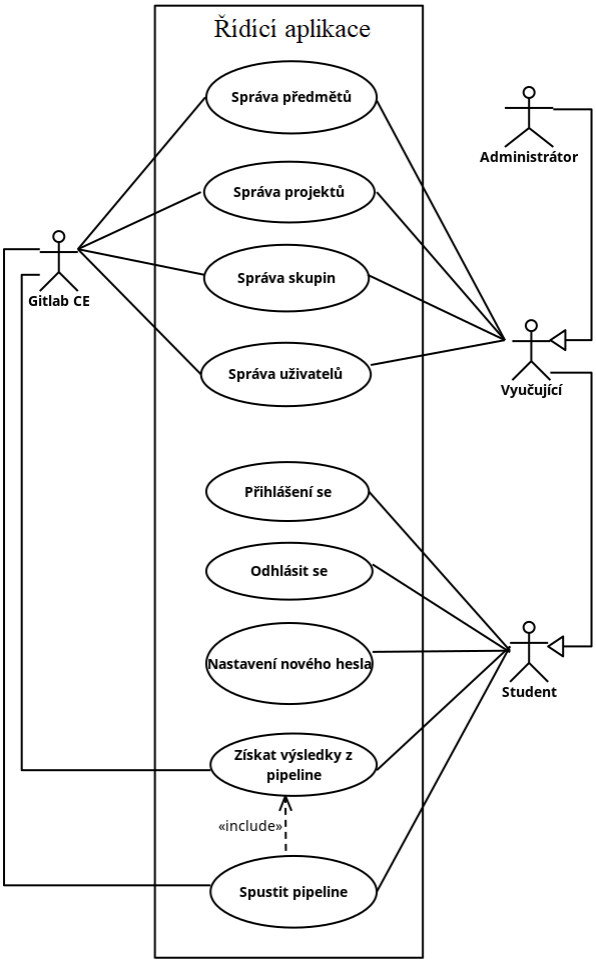
Samostatný systém bude vyvinut s cílem spravovat tři různé uživatelské role, jak je názorně znázorněno na use case diagramu. Každá role bude mít přístup k určitým funkcím a bude plnit specifické úkoly.

První rolí v systému bude "Student". Studenti budou mít možnost přihlásit se a odhlásit do systému podle svých potřeb. Budou mít také možnost změnit své přístupové heslo. Důležitou funkcí pro studenty bude spuštění pipeline pro otestování funkčnosti jejich odevzdaných projektů. Po dokončení testů budou moci získat výsledky, které jim poskytnou přehled o úspěšnosti jejich řešení a výsledcích jednotlivých testů.

Druhou rolí bude "Vyučující". Tato role bude mít pravomoc spravovat uživatele v systému, což zahrnuje registraci nových studentů a odstranění stávajících. Dále budou mít vyučující možnost spravovat skupiny uživatelů, které si vytvoří, a budou moci plánovat projekty a předměty pro studenty. V rámci těchto projektů budou vyučující schopni hodnotit studenty.

Třetí rolí bude "Administrátor". Administrátor bude mít úplný přístup ke všem funkcím systému a bude zodpovědný za celkovou správu. Bude mít pravomoc spravovat uživatele, skupiny, projekty a předměty. Bude také mít možnost provádět další konfigurace a údržbu systému.

V rámci systémové architektury bude klíčovou komponentou implementace GitLab CE, ve které se budou definovat uživatelské účty, skupiny odpovídající jednotlivým předmětům, uživatelské skupiny a specifické projekty. V kontextu těchto projektů bude možné spustit předdefinované pipeline, zaměřené na automatizované testování a vyhodnocení. Integrace tohoto řešení poskytne optimalizovanou správu verzí projektů a zefektivní proces testování a hodnocení studentů.



Obrázek 3 – User-case diagram řídící aplikace (zdroj: vlastní)

5.6 Definice testovací sady

Vyučujícím bude poskytnuta možnost definovat sadu testů, které slouží k vyhodnocení kvality studentských prací. Po odevzdání studentských prací do GitLab repozitářů se pipeline automaticky spustí a provede definované testovací scénáře. Výsledky testování budou následně zobrazeny jak studentům, tak vyučujícím. Na základě těchto výsledků získají vyučující okamžitou zpětnou vazbu ohledně splnění požadavků a funkčnosti studentských prací. Tento automatizovaný proces zajišťuje rychlou a efektivní zpětnou vazbu, což umožňuje studentům lépe pochopit a zdokonalit své projekty a vyučujícím poskytuje užitečné nástroje pro objektivní hodnocení a vedení studentů v jejich vzdělávacím procesu.

5.7 Bezpečnostní opatření

Důležitým krokem je zajistit důkladnou autentizaci a autorizaci uživatelů v systému. Bez ohledu na jejich roli (student, vyučující nebo administrátor) musí projít procesem autentizace při přihlašování do systému. Dále je nezbytné pečlivě nastavit a řídit přístupová práva tak, aby každý uživatel měl přístup pouze k datům a funkcím, které jsou pro něho relevantní.

Samotné služby systému budou provozovány v Docker kontejnerech, což umožňuje zajištění bezpečnosti celého systému. Správná izolace jednotlivých služeb v kontejnerech je však klíčová, aby se zabránilo komunikaci mezi nimi a minimalizovalo riziko šíření hrozeb mezi jednotlivými kontejnery.

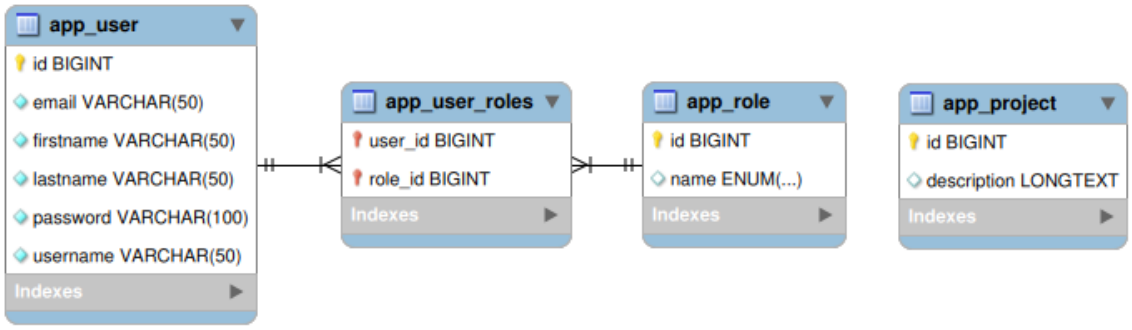
Pravidelné monitorování a auditování aktivit v Docker kontejnerech jsou nezbytné pro detekci a rychlou reakci na podezřelé činnosti nebo možné bezpečnostní incidenty. Tato opatření zvyšují celkovou bezpečnost systému a umožňují efektivní reakci na potenciální bezpečnostní hrozby.

6 NÁVRH SYSTÉMU

V této kapitole je popsán návrh systém pro automatizovaného vyhodnocování studentských prací s využitím Git.

6.1 DATOVÝ MODEL ŘÍDÍCÍ APLIKACE

Z analýzy systémových potřeb vyplynula zásadní potřeba ukládat informace o uživatelích v databázi. To umožní efektivně spravovat identitu a oprávnění jednotlivých uživatelů. ERD diagram, který je na obrázku 3, graficky znázorňuje komplexní strukturu databázového modelu. Na první pohled z něj jasně vyplývá, jak jsou jednotlivé entity vzájemně propojeny a jakým způsobem spolu komunikují.



Obrázek 4 – Databázový model řídící aplikace (zdroj: vlastní)

Kombinace těchto čtyř tabulek je nezbytná pro správnou a bezpečnou autentizaci uživatelů v řídící aplikaci. Díky tomuto modelu může aplikace ověřit identitu uživatele a poskytnout mu přístup pouze k těm částem systému, ke kterým má příslušná oprávnění

6.1.1 Entita APP_User

Hlavní stavební blok databázového modelu je entita *APP_User*. Tato tabulka obsahuje veškeré osobní údaje uživatele, které jsou nezbytné pro jeho identifikaci a komunikaci, jako je jméno, příjmení, uživatelské jméno, heslo a email. Aby bylo zajištěno, že každý uživatel je v systému jedinečný, je vybaven unikátním identifikátorem. Díky vazební tabulce *APP_User_Role* je možné uživateli přiřadit různé role, což rozšiřuje jeho možnosti v systému.

6.1.2 Entita APP_Role

Role je zásadním prvkem, který určuje, co může a nemůže uživatel v řídící aplikaci dělat. V tabulce *APP_Roles* jsou definovány různé role, například *ADMIN*, *STUDENT* nebo

TEACHER. Každá z těchto rolí má svůj jedinečný identifikátor, který slouží jako spojovací článek s tabulkou *APP_User_Role*.

6.1.3 Entita APP_User_Role

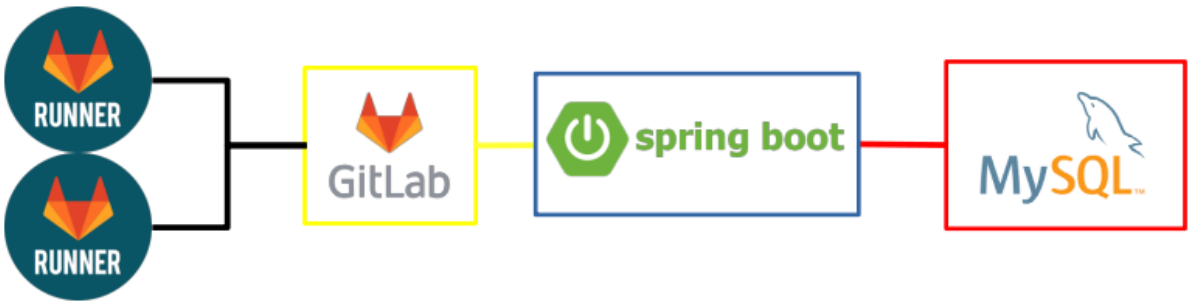
Abychom zajistili flexibilitu a mnohostrannost našeho systému, vytvořili jsme vazební tabulku *APP_User_Role*. Tato tabulka propojuje uživatele s rolemi a umožňuje tak přiřazovat jednomu uživateli více rolí. Následně může mít jeden uživatel v rámci aplikace více funkcí, a naopak jedna role může být přiřazena více uživatelům.

6.1.4 Entita APP_Project

Tato tabulka byla nezbytná k zajištění perzistentního uložení rozsáhlých zadání projektů, neboť nebylo možné ukládat taková zadání přímo do GitLabu z důvodu omezení na 2000 znaků pro popis projektu. Obsahuje sloupce pro unikátní ID projektu a jeho podrobný popis, což umožňuje efektivní propojení s projekty v GitLabu a zajišťuje, že celé zadání projektu je dostupné v nezkrácené formě.

6.2 Architektura systému

Navrhovaný systém bude tvořen několika klíčovými komponentami, které budou vzájemně propojeny tak, aby umožnily automatizované testování a vyhodnocování studentských prací v prostředí kolaboračního Git serveru.



Obrázek 5 – Architektura systému (zdroj: vlastní)

Navržené prostředí pro automatizované vyhodnocování studentských prací pomocí Git bude využívat virtuální sítě pro zajištění bezpečné a izolované komunikace mezi komponentami. Centrálním prvkem tohoto prostředí je GitLab Community Edition (CE), který bude propojen s dvěma GitLab Runners. Tyto komponenty budou navzájem komunikovat pomocí vlastní virtuální sítě. Dále bude zde existovat řídící aplikace, která bude komunikovat s GitLab CE po

další vlastní virtuální síti. Stejně tak pro komunikaci řídící aplikace s MySQL databází bude vytvořena další virtuální síť. Díky využití virtuálních sítí bude zajištěna bezpečnost a izolace komunikace mezi jednotlivými komponentami systému.

Celé prostředí bude postaveno na kontejnerech Docker, což umožní snadnou správu a nasazení jednotlivých komponent. Systém poskytne automatizovanou a integrovanou platformu pro vyhodnocování studentských prací, kde GitLab CE a GitLab Runners budou zajišťovat sledování a správu kódu, zatímco řídící aplikace a MySQL databáze přidají další potřebné funkcionality. Celkově tedy navržené prostředí poskytne efektivní a bezpečné řešení pro automatizované vyhodnocování studentských prací. Hlavní části architektury budou popsány v následujících kapitolách.

6.2.1 Docker

Při budování systému bude postupováno od méně komplexních k těm komplexnějším. Nejprve bude nezbytné nalézt příslušné Docker images, které budou využity jako základ pro jednotlivé služby. Následně bude vytvořen Dockerfile souboru, který slouží k dodatečné konfiguraci výsledného Docker image pro Gitlab Runner. Dalším krokem bude vytvoření souboru docker-compose.yml pro konfiguraci kontejnerů, který uchovává nastavení a pořadí jednotlivých služeb. Tento soubor docker-compose.yml usnadňuje sestavení celého systému s přesně definovanými službami a jejich vzájemnými vazbami.

6.2.2 Kolaborační Git server

Tato centrální komponenta představuje významný a důležitý prvek, který bude sloužit jako úložiště pro všechny studentské práce a zároveň nabídne možnost efektivní správy verzí pro jednotlivé projekty. Hlavním cílem této centrální platformy je umožnit studentům a jejich vyučujícím pohodlný a organizovaný způsob, jak spravovat, verzovat, sdílet a hodnotit studentské práce.

Jednou z klíčových funkcionalit, kterou tento server nabídne, je podpora vytváření skupin studentů, předmětů nebo podle jednotlivých cvičení. Tímto způsobem budou moci učitelé jednoduše organizovat a sdružovat skupiny studentů, které mají společné zadání.

Díky této centrální komponentě budou studentské práce pečlivě a systematicky evidovány. Dále systém verzí umožní uživatelům snadné porovnání a sledování změn provedených v jednotlivých projektech. Tato funkcionalita je neocenitelná při kolektivní práci na projektech,

protože umožní rychlé zjištění, jaké změny byly provedeny, kdo je provedl a kdy. Tím se minimalizuje riziko ztráty dat nebo nedorozumění ohledně provedených úprav.

6.2.3 Gitlab Runner

Pro účely automatizovaného testování (CI/CD) bude využit GitLab Runner ve spojení s nástrojem Docker, což umožní spouštění jednotlivých pipeline. Tyto pipeline budou schopny testovat projekty napsané v programovacích jazycích Java a C#. Díky použití GitLab Runnerů a nástroje Docker bude celý proces automatizovaného testování bezpečný, spolehlivý a opakovatelný.

Jednotlivé pipeline zajistí systematické spouštění testů a umožní získání výsledků po testování. Tímto způsobem vyučující získají objektivní hodnocení kvality studentských prací a studenti budou moci snáze identifikovat případné chyby nebo nedostatky ve své práci.

6.2.4 MySQL

V systému bude použita MySQL databáze pro efektivní ukládání údajů o uživatelích, kteří se registrují do řídící aplikace. Tato databáze slouží jako spolehlivé úložiště pro veškeré relevantní informace o uživatelích, což umožňuje snadné a rychlé spravování jejich dat.

V databázi budou uchovávány základní informace o uživatelích, jako jsou jméno, příjmení, e-mailová adresa, uživatelské jméno a zašifrované heslo pro zabezpečení přístupu. Dále se zde ukládají další relevantní údaje, jako jsou uživatelské role (např. administrátor, vyučující, student).

Databáze umožňuje snadnou manipulaci s daty a rychlé vyhledávání informací, což je klíčové pro efektivní řízení uživatelských účtů a přístupových práv. Díky MySQL jsou data uživatelů uložena systematicky a uspořádaně, což přispívá k plynulému fungování řídící aplikace a minimalizuje riziko ztráty informací.

6.2.5 Řídící aplikace

Řídící aplikace má za cíl poskytnout efektivní správu celého softwarového ekosystému spojeného s GitLab Community Edition (CE). Tato aplikace umožní uživatelům vytvářet a spravovat uživatele, skupiny a projekty v GitLab CE prostřednictvím intuitivního a přehledného rozhraní. Důležitým prvkem tohoto návrhu je také implementace uživatelských

rolí pro správu přístupu k API pro řídící aplikaci. V řídící aplikace budou dostupné následující uživatelské role: student, vyučující a administrátor.

Každá z uživatelských rolí bude mít definované specifické oprávnění a přístup k funkcionalitě systému. Studenti budou moci spouštět pipeline pro své projekty a získávat výsledky testování, což jim umožní monitorovat průběh svých úkolů. Vyučující budou mít možnost vytvářet a editovat projekty, skupiny a skupiny studentů, což jim umožní efektivně řídit jednotlivé předměty a sledovat pokrok studentů. Administrátoři budou mít nejvyšší úroveň oprávnění a zodpovědnost za celkovou správu aplikace a všechny její funkcionality.

Backendová část aplikace bude zodpovědná za zpracování dat a komunikaci s GitLab CE API a provádění operací spojených se správou uživatelů, skupin a projektů. Díky tomuto architektonickému řešení bude systém schopen rychle a spolehlivě reagovat na požadavky uživatelů.

Frontendová část aplikace zajišťuje správu projektů a skupin v rámci GitLab CE a poskytuje uživatelům přehled o jednotlivých projektech, předmětech a skupinách. Umožňuje snadné vytváření a správu projektů, skupin a skupin uživatelů prostřednictvím uživatelsky přívětivého rozhraní. Uživatelé mohou procházet seznam projektů, získávat detailní informace o každém projektu a sledovat stav pipeline a výsledky testování.

6.3 Testování

Testování při vývoji aplikace bude zahrnovat tři různé typy testů: unit testy, integrační testy a UI Selenium testy. Tato kombinace testování bude zajišťovat důkladné a komplexní ověření funkcionality a kvality aplikace.

Unit testy budou zaměřeny na testování jednotlivých kódových bloků a funkcí aplikace. Tyto testy jsou prováděny na izolovaných částech kódu a umožňují ověřit správnost chování jednotlivých jednotek. Díky nim bude možné identifikovat chyby na úrovni nejmenších kódových entit a zajistit, že jednotlivé části aplikace fungují správně a očekávaným způsobem.

Integrační testy budou prováděny za účelem ověření správné komunikace a spolupráce mezi různými komponentami aplikace. Tento typ testů zkoumá, jak jednotlivé části interagují mezi sebou a zda společně vytvářejí funkční a integrovaný systém. Integrační testy umožňují odhalit potenciální problémy a nekonzistence v komunikaci mezi jednotlivými částmi aplikace.

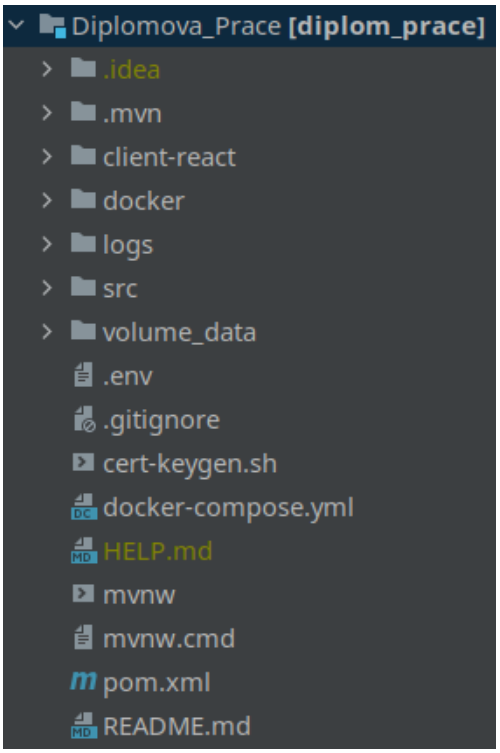
UI Selenium testy budou zaměřeny na ověření funkcionality uživatelského rozhraní aplikace. Tyto testy simulují interakce uživatele s aplikací a kontrolují, zda všechny prvky na uživatelském rozhraní pracují správně a reagují na uživatelské vstupy korektním způsobem. Díky UI Selenium testům bude zajištěno, že uživatelské rozhraní je intuitivní, přívětivé a odpovídá očekáváním uživatele.

7 IMPLEMENTACE

V této kapitole jsou popsány jednotlivé konfigurační soubory jednotlivých služeb nezbytných pro systém automatizovaného vyhodnocování studentských prací s využitím Git.

7.1 Základní struktura projektu

Projekt se skládá ze dvou konfiguračních souborů kontejnerů a z několika adresářů, ve kterých se nachází veškeré konfigurační soubory pro vytvoření kontejnerů jednotlivých služeb tak, aby byl zajištěn správný běh celého systému. Ukázka struktury projektu je uvedena níže:



Obrázek 6 – Struktura hlavního adresáře projektu (zdroj: vlastní)

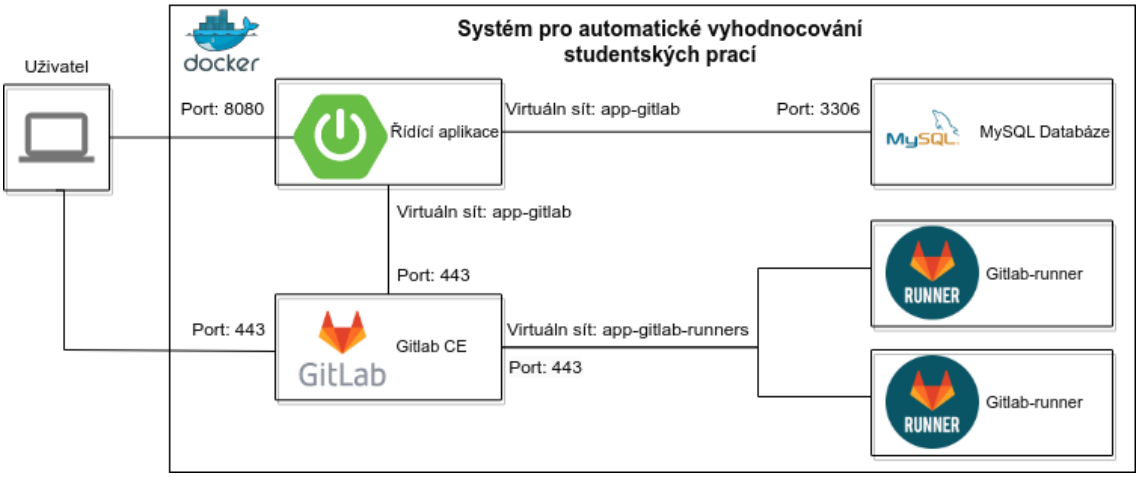
Podrobnější seznámení s jednotlivými soubory a adresáři je k dispozici v následujících kapitolách.

7.2 Globální konfigurace docker kontejnerů

Pro usnadnění správy obrazů Docker je vhodné vytvořit globální konfiguraci. Tato konfigurace slouží k jednodušší manipulaci s určitými proměnnými, které jsou v rámci projektu využívány, jako například informace o síti nebo jména jednotlivých služeb. Tímto způsobem je možné lépe centralizovat a spravovat potřebné parametry a usnadnit celkovou konfiguraci a provoz Docker kontejnerů.

7.2.1 Soubor docker-compose.yml

Jedná se o soubor obsahující komplexní specifikaci všech kontejnerů včetně nastavení proměnných prostředí načtených z konfiguračního souboru .env, portů sloužících ke komunikaci kontejneru s okolním světem, virtuálních sítí, pomocí kterých mohou jednotlivé kontejnery spolu navzájem komunikovat a volumes pro trvale uložení některých dat v kontejneru tak, aby byla data zachována i po jeho vymazání. Komunikační schéma je znázorněno na obrázku 7, kde je ilustrováno, jak spolu jednotlivé Docker kontejnery komunikují.



Obrázek 7 – Komunikační schéma Docker kontejnerů (zdroj: vlastní)

V následující sekci je představen obsah a struktura souboru docker-compose.yml.

```
1.  version: '3.9'
```

Direktiva "version" slouží ke specifikaci verze formátu souboru docker-compose.yml. Každá verze má odlišné možnosti konfigurace kontejnerů, neboť mezi jednotlivými verzemi docházelo k přidávání nebo odebírání konfiguračních možností. Třetí verze byla vytvořena s cílem udržet kompatibilitu mezi projekty vytvořenými pomocí Docker Compose a Docker Swarm.

Ukázka konfigurace kontejneru obsahujícího službu MySQL v souboru docker-compose.yml:

```
 2.  services:
 3.  …
66.    mysql:
67.      image: mysql:8.0.32
68.      command: --default-authentication-plugin=mysql_native_password
69.      container_name: ${MYSQL_CONTAINER_NAME}
70.      restart: on-failure
71.      deploy:
72.        resources:
73.          limits:
74.            cpus: "2"
75.            memory: "1g"
76.          reservations:
77.            cpus: "2"
78.            memory: "512m"
79.      environment:
80.        MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
81.        MYSQL_USER: ${MYSQL_USER}
82.        MYSQL_PASSWORD: ${MYSQL_PASSWORD}
83.        MYSQL_DATABASE: ${MYSQL_DATABASE}
84.      volumes:
85.        - mysql_db:/var/lib/mysql
86.      healthcheck:
87.        test: ["CMD", "mysqladmin" ,"ping", "-h", "localhost"]
88.        interval: 30s
89.        timeout: 10s
90.        retries: 5
91.        start_period: 1m
92.      networks:
93.        - dp-mysql
```

Pod direktivou "services" jsou specifikovány jednotlivé kontejnery, které jsou identifikovány názvem služby, za nímž následují další direktivy pro konfiguraci daného kontejneru.

- `image` – specifikace názvu docker image použitého pro kontejner, tento název je uložen v proměnné MYSQL_DATABASE_IMAGE definované souboru .env,
- `command` – specifikuje příkaz, který bude spuštěn uvnitř kontejneru po jeho vytvoření a spuštění,
- `container_name` – direktiva určuje název pro Docker image, který je uložen v proměnné "MYSQL_CONTAINER_NAME" definované v souboru ".env",

- `restart` – určí, zda se má automaticky restartovat kontejner. Lze nastavit jeden z parametrů:
    - *no* – nerestartovat automaticky,
    - *on-failure* – restartovat při chybě kontejneru,
    - *always* – kontejnery budou automaticky restartovány, pokud dojde k jejich pádu, s výjimkou případů, kdy uživatel ručně vypne kontejner. V takovém případě se kontejnery opět automaticky spustí po restartování služby Docker.
    - *unless-stopped* – podobná hodnotě "always", ale kontejnery nebudou automaticky spuštěny při restartování služby Docker. Kontejnery se spustí pouze v případě, že byly aktivní předtím, než byl nástroj Docker zastaven.
- `deploy` – slouží k definování limitů a rezervací pro zdroje kontejnerů, když jsou nasazovány.
    - `limits` – určují maximální množství prostředků, které může kontejner využít (počet jader CPU a paměť),
    - `reservations` – definují minimální množství zdrojů, které musí být kontejneru přiděleny při jeho běhu.
- `environment` – specifikace nezbytných proměnných prostředí, které slouží pro nastavení uživatelských účtu v databázi,
- `volume` – specifikuje adresář, který bude uložen na trvalém uložišti, aby nedošlo k vymazání databáze po smazání kontejneru,
- `networks` – specifikace virtuálních sítí pro jednotlivé služby tak, aby jednotlivé služby komunikovali jen s nezbytnými kontejnery.f

Ukázka konfigurace virtuálních sítí pro kontejnery v souboru docker-compose.yml:

```
277.  networks:
278.    dp-mysql:
279.      driver: bridge
280.    dp-gitlab:
281.      driver: bridge
282.    dp-gitlab-runners:
283.      driver: bridge
```

V této ukázce jsou specifikovány virtuální sítě, které umožňují komunikaci mezi jednotlivými kontejnery. Názvy těchto virtuálních sítí obsahují vždy jména služeb, které spolu komunikují přes tuto síť. S ohledem na bezpečnost jsou vytvořeny virtuální sítě, které propojují pouze nezbytné kontejnery mezi sebou.

Ukázka konfigurace trvalých uložišť pro kontejnery v souboru docker-compose.yml:

```
285.  volumes:
286.    gitlab_config:
287.        driver: local
288.    gitlab_logs:
289.        driver: local
290.    gitlab_data:
291.        driver: local
292.    gitlab-runner-1:
293.      driver: local
294.    gitlab-runner-2:
295.      driver: local
296.    mysql_db:
297.      driver: local
```

Pro zachování dat služeb uvnitř kontejnerů jsou vytvořena trvalá uložiště, která umožňují persistentní uchování dat. Díky nim nedochází ke ztrátě například databáze při odstranění kontejneru obsahujícího službu MySQL. Všechny služby, které uchovávají data uživatelů a řídící aplikace, využívají tato trvalá uložiště. Názvy těchto uložišť jsou intuitivní a umožňují snadnou identifikaci, pro kterou službu je dané uložiště určeno.

7.3 Konfigurace jednotlivých kontejnerů

V této kapitole se nachází popis konfigurace, sloužící k vytvoření nezbytných kontejnerů.

7.3.1 GitLab Runner

Konfigurace GitLab je složena z dvou klíčových konfiguračních souborů, které jsou umístěny v adresáři "*docker/dockerfile/gitlab-runner*". Prvním souborem je "*Dockerfile*", který slouží k definici docker image pro GitLab Runner. Tento soubor obsahuje instrukce pro sestavení kontejneru, včetně specifikace základního obrazu, instalace potřebných závislostí a konfiguraci prostředí pro běh GitLab Runner.

Druhým souborem je "*entrypoint.sh*", který má klíčovou roli při spouštění kontejneru. Tento skript je spuštěn v okamžiku, kdy je kontejner spuštěn, a provádí inicializační úkony a nastavení potřebných proměnných prostředí pro správný běh GitLab Runner.

Soubor Dockerfile

```
1.  FROM gitlab/gitlab-runner:alpine-v15.8.2
```

Direktiva `FROM` určí základní kontejner pro docker image obsahující *gitlab/gitlab-runner:alpine* ve verzi v15.8.2.

```
2.  COPY entrypoint.sh /docker-entrypoint.sh
```

Tato direktiva `COPY` slouží ke zkopírování souboru "entrypoint.sh" z aktuálního adresáře do adresáře "/docker-entrypoint.sh" uvnitř nově vytvářeného kontejneru. Tímto způsobem je soubor "entrypoint.sh" přidán do kontejneru a bude moci být použit jako skript, který bude spuštěn při spuštění samotného kontejneru.

```
3.  ENTRYPOINT ["./docker-entrypoint.sh"]
```

Direktiva `ENTRYPOINT` slouží k určení skriptu, který bude spuštěn jako výchozí proces v kontejneru, když je tento kontejner spuštěn. V tomto případě je jako výchozí proces určen skript s názvem "docker-entrypoint.sh".

```
4.  CMD ["run", "--user=gitlab-runner", "--working-directory=/home/gitlab-runner"]
```

Direktiva `CMD` slouží k určení výchozího příkazu, který bude spuštěn v případě, že není explicitně specifikován jiný příkaz při spuštění kontejneru. Uvedený příkaz způsobí spuštění Gitlab Runneru s parametry "--user=gitlab-runner" a "--working-directory=/home/gitlab-runner" po spuštění kontejneru.

Soubor entrypoint.sh

```
1.  GITLAB_SERVER_DOMAIN="https://$GITLAB_SERVER_DOMAIN"
2.  INITIAL_WAIT=180
3.  RETRY_INTERVAL=5
```

Pro komunikaci s GitLabem jsou stanoveny specifické konfigurační proměnné. Pro proces registrace Gitlab Runner v rámci GitLabu jsou však využívány proměnné, jež jsou inicializovány z hodnot definovaných v systémových proměnných prostředí.

```
4.  echo "Starting Gitlab-runner and attempting to connect to Gitlab-ce"
5.  STATUS=$(curl -s -o /dev/null -k -w "%{http_code}" "$GITLAB_SERVER_DO-
    MAIN/users/sign_in")
6.
7.  if [ $STATUS -ne 200 ]; then
8.      echo "Failed to connect. Waiting $INITIAL_WAIT seconds before retrying..."
9.      sleep $INITIAL_WAIT
10.     while true; do
11.         echo "Gitlab-runner - connecting to the Gitlab-ce ($INITIAL_WAIT s)"
12.         STATUS=$(curl -s -o /dev/null -k -w "%{http_code}" "$GITLAB_SERVER_DO-
    MAIN/users/sign_in")
13.         if [ $STATUS -eq 200 ]; then
14.             break
15.         fi
16.         INITIAL_WAIT=$(($INITIAL_WAIT + 5))
17.         sleep 5
18.     done
19. fi
20. done
```

Tato část skriptu ověřuje dostupnost serveru GitLab pomocí HTTP dotazu a čeká na úspěšné připojení. Pokud je server dostupný a komunikuje s Gitlab Runner (návratový kód HTTP odpovědi je 200), skript pokračuje dál. Pokud není server dostupný, skript počká zadaný počet sekund a pak znovu zkusí navázat komunikaci s GitLab serverem, dokud není dosaženo úspěšného připojení. Tímto způsobem se zajistí, že Gitlab Runner bude spuštěn až poté, co bude server GitLab server plně dostupný a připraven na příchozí požadavky.

```
21.  echo "Gitlab-runner - connected to the Gitlab-ce"
22.  echo "URL: https://gitlab-ce"
23.  echo "TOKEN: $GITLAB_RUNNER_TOKEN"
24.  echo "EXECUTOR: $GITLAB_RUNNER_EXECUTOR"
25.  echo "DOCKER_IMAGE: $GITLAB_RUNNER_DOCKER_IMAGE"
26.  echo "DESCRIPTION: $GITLAB_RUNNER_DESCRIPTION"
27.  echo "CONFIG: $GITLAB_RUNNER_DESCRIPTION"
28.  echo "MAINTENANCE_NOTE: $GITLAB_RUNNER_DESCRIPTION"
29.  echo "TAG_LIST: $GITLAB_RUNNER_DESCRIPTION"
30.  echo "RUN_UNTAGGED: $GITLAB_RUNNER_DESCRIPTION"
31.  echo "LOCKED: $GITLAB_RUNNER_DESCRIPTION"
32.  echo "ACCESS_LEVEL: $GITLAB_RUNNER_DESCRIPTION"
```

Poté skript vypíše informace o nastavení Gitlab Runneru, mezi které patří: URL, token, executor, Docker image a další konfigurační proměnné.

```
33.  if [ ! -f "$GITLAB_RUNNER_CONFIG" ]; then
34.      echo "Start register Gitlab-runner to the Gitlab-ce"
35.      gitlab-runner register \
36.        --non-interactive \
37.        --url "$GITLAB_SERVER_DOMAIN" \
38.        --registration-token "$GITLAB_RUNNER_TOKEN" \
39.        --executor "$GITLAB_RUNNER_EXECUTOR" \
40.        --description="$GITLAB_RUNNER_DESCRIPTION" \
41.        --config="$GITLAB_RUNNER_CONFIG" \
42.        --docker-image $GITLAB_RUNNER_DOCKER_IMAGE \
43.        --maintenance-note "$GITLAB_RUNNER_MAINTENANCE_NOTE" \
44.        --tag-list "$GITLAB_RUNNER_TAG_LIST" \
45.        --run-untagged="$GITLAB_RUNNER_RUN_UNTAGGED" \
46.        --locked="$GITLAB_RUNNER_LOCKED" \
47.        --access-level="$GITLAB_RUNNER_ACCESS_LEVEL"
48.
49.      echo "Finished register Gitlab-runner to the Gitlab-ce"
50.  fi
```

Tato část skriptu zkontroluje, zda existuje konfigurační soubor pro Gitlab Runner. Pokud tento soubor neexistuje, skript provede registraci Gitlab Runneru do GitLab prostřednictvím příkazu "gitlab-runner register". Během registrace jsou nastaveny různé parametry, jako je URL GitLab serveru, registrční token, executor (prostředí, ve kterém bude GitLab Runner spuštěn), popis

Runneru a další. Po dokončení registrace je vypsána zpráva "Finished register Gitlab-runner to the Gitlab-ce".

```
47.  exec /usr/bin/dumb-init /entrypoint "$@"
```

Nakonec skript spustí samotný Gitlab Runner pomocí příkazu "exec /usr/bin/dumb-init /entrypoint "$@"". Tímto způsobem je Gitlab Runner spuštěn s příslušnými nastaveními a je připraven provádět úlohy, které mu budou přiděleny.

7.3.2 Řídící aplikace

Konfigurace řídicí aplikace spočívá ve dvou konfiguračních souborech nacházejících se v adresáři "docker/dockerfile/spring-app". "Dockerfile" je prvním z nich a jeho účelem je definovat kontejnerový image řídicí aplikace. Detailně popisuje kroky vytvoření tohoto obrazu, včetně výběru základního obrazu, instalace nezbytných závislostí a nastavení prostředí pro chod aplikace. Na druhé straně "entrypoint.sh" je skript, který hraje kritickou roli při inicializaci kontejneru. V momentě spuštění kontejneru je tento skript aktivován, kde řídí počáteční procesy a konfiguruje nezbytné proměnné pro optimální fungování řídicí aplikace.

Soubor Dockerfile

```
1.  FROM node:20 AS frontend-build
2.
3.  WORKDIR /frontend
4.  COPY client-react ./
5.  RUN npm install
6.  RUN npm run build
```

V této části je použit docker image *node:20*, který je označen jako *frontend-build* a slouží k sestvení frontend části řídící aplikace. V rámci kontejneru je nastavený pracovní adresář */frontend*, do kterého se nakopíruje obsah adresáře client-react z hostitelského systému. Poté se v kontejneru instalují všechny potřebné npm balíčky pomocí příkazu *npm install*. Nakonec se spustí build pro frontend aplikace pomocí příkazu *npm run build*.

```
8.  FROM maven:3.8.1-openjdk-17-slim AS backend-build
9.
10. WORKDIR /app
11. COPY pom.xml ./pom.xml
12. COPY src ./src
13. COPY --from=frontend-build /frontend/build /app/src/main/resources/static/
14.
15. RUN mvn package -P environment-prod -Dmaven.test.skip=true
```

V této části je použit docker image *maven:3.8.1-openjdk-17-slim*, který je označen jako *backend-build* a slouží k sestavení řídící aplikace. V rámci kontejneru je nastavený pracovní

adresář na */app*, do kterého se nakopírují soubory *pom.xml* a adresář *src* z hostitelského systému. V následujícím kroku se provede proces kopírování frontend části z předchozí fáze do resources řídící aplikace. Po dokončení kolorování všechny nezbytných souborů se v kontejneru spouští maven balíček se specifikovaným profilem *environment-prod*, přičemž je v této fázi přeskočeno testování pomocí parametru *-Dmaven.test.skip=true*.

```
17.  FROM openjdk:17-jdk-alpine
18.
19.  COPY docker/dockerfile/spring-app/entrypoint.sh /app/entrypoint.sh
20.  COPY --from=backend-build /app/target/diplom_prace-1.0.0.jar /app/ridici_apli-
     kace.jar
21.
22.  ENTRYPOINT ["sh", "/app/entrypoint.sh"]
```

Tato sekce využívá docker image *openjdk:17-jdk-alpine* a je primárně určena pro kontejnerizaci řídící aplikace. Pro sestavení image je zde provedeno nakopírování entrypoint.sh skriptu umístěného v adresáři docker/dockerfile/spring-app/ na hostitelském systému, do interního úložiště kontejneru. Současně se do kontejneru nakopíruje JAR archiv z předchozího procesu. Dále aby byla zajištěna automatická inicializace skriptu entrypoint.sh po spuštění kontejneru, je implementován direktivní příkaz `ENTRYPOINT`.

Soubor entrypoint.sh

```
1.  if [ ! -f "$GITLAB_RUNNER_CONFIG" ]; then
2.      echo "Start register Gitlab-runner to the Gitlab-ce"
3.      gitlab-runner register \
4.        --non-interactive \
5.        --url "$GITLAB_SERVER_DOMAIN" \
6.        --registration-token "$GITLAB_RUNNER_TOKEN" \
7.        --executor "$GITLAB_RUNNER_EXECUTOR" \
8.        --description="$GITLAB_RUNNER_DESCRIPTION" \
9.        --config="$GITLAB_RUNNER_CONFIG" \
10.       --docker-image $GITLAB_RUNNER_DOCKER_IMAGE \
11.       --maintenance-note "$GITLAB_RUNNER_MAINTENANCE_NOTE" \
12.       --tag-list "$GITLAB_RUNNER_TAG_LIST" \
13.       --run-untagged="$GITLAB_RUNNER_RUN_UNTAGGED" \
14.       --locked="$GITLAB_RUNNER_LOCKED" \
15.       --access-level="$GITLAB_RUNNER_ACCESS_LEVEL"
```

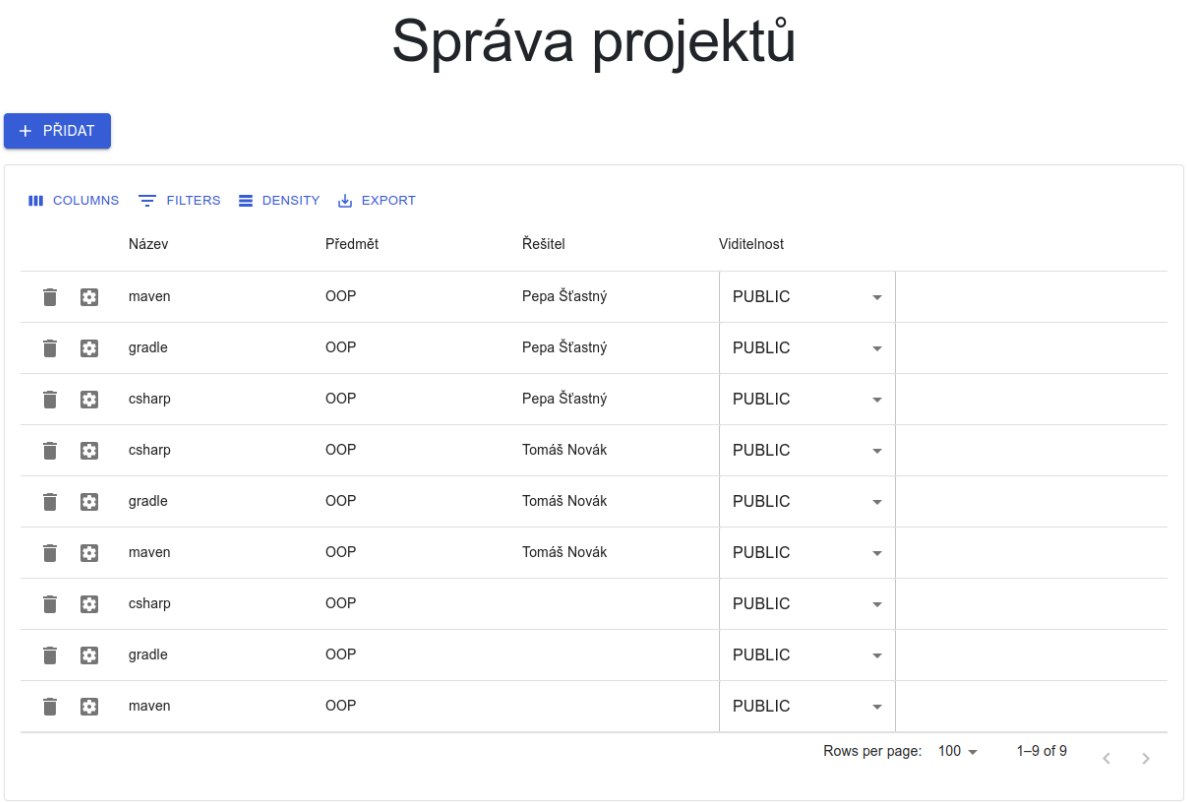
Skript ověřuje dostupnost MySQL databáze na portu 3306 a GitLab na specifikované adrese pomocí nástrojů *nc* a *curl*. Pokud jsou obě služby nedostupné, skript čeká 5 sekund a zvyšuje interní časovač před dalším pokusem o ověření. Jakmile jsou obě služby dostupné, skript spustí Java aplikaci ze souboru ridici_aplikace.jar.

7.4 Řídící aplikace

Tato část se věnuje návrhu a implementaci řídící webové aplikace, která slouží ke správě a ovládání jednotlivých služeb na serveru. V předchozích kapitolách byly detailně popsány jednotlivé služby a jejich konfigurace, spolu s jejich možnostmi. Nyní následuje podrobný popis řídící aplikace, včetně všech jejích částí, jak se zobrazují v uživatelském rozhraní. Tato webová aplikace umožňuje uživatelům spravovat a monitorovat všechny dostupné služby a provádět různé akce a operace v jednoduchém a uživatelsky přívětivém rozhraní.

7.4.1 Nabídka Projekty

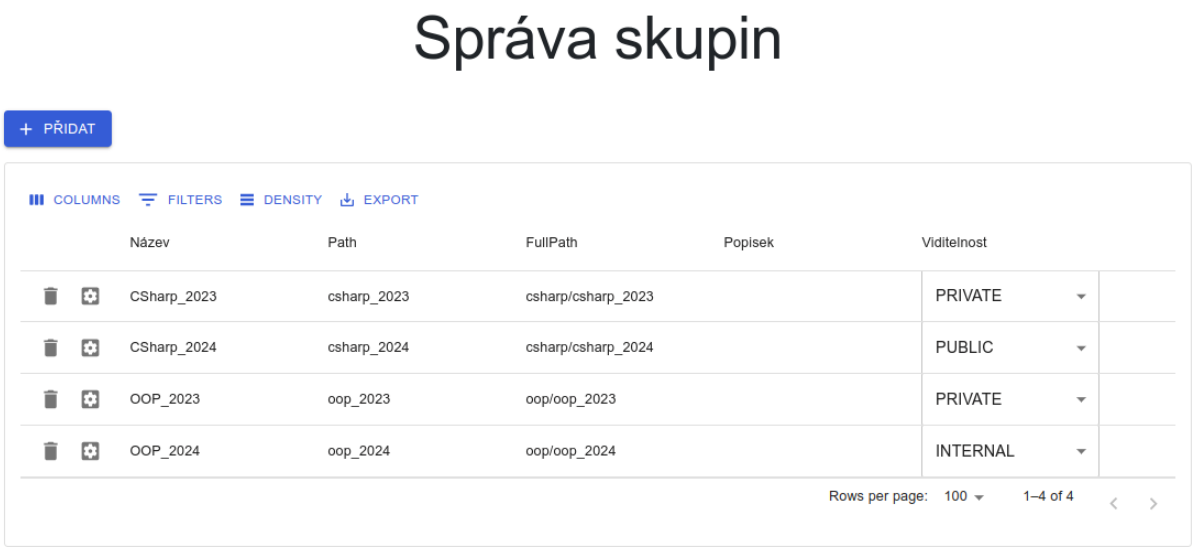
Slouží k poskytnutí přehledu všech dostupných projektů v systému. Vyučující zde mají možnost vytvářet nové projekty a spravovat existující. Tato stránka také umožňuje zadávat projekty pro studenty, což poskytuje jasný rámec pro jejich práci a zároveň umožňuje sledovat postup a výsledky jednotlivých projektů.



Obrázek 8 – Ukázka nabídky správy projektů (zdroj: vlastní)

7.4.2 Nabídka Skupin

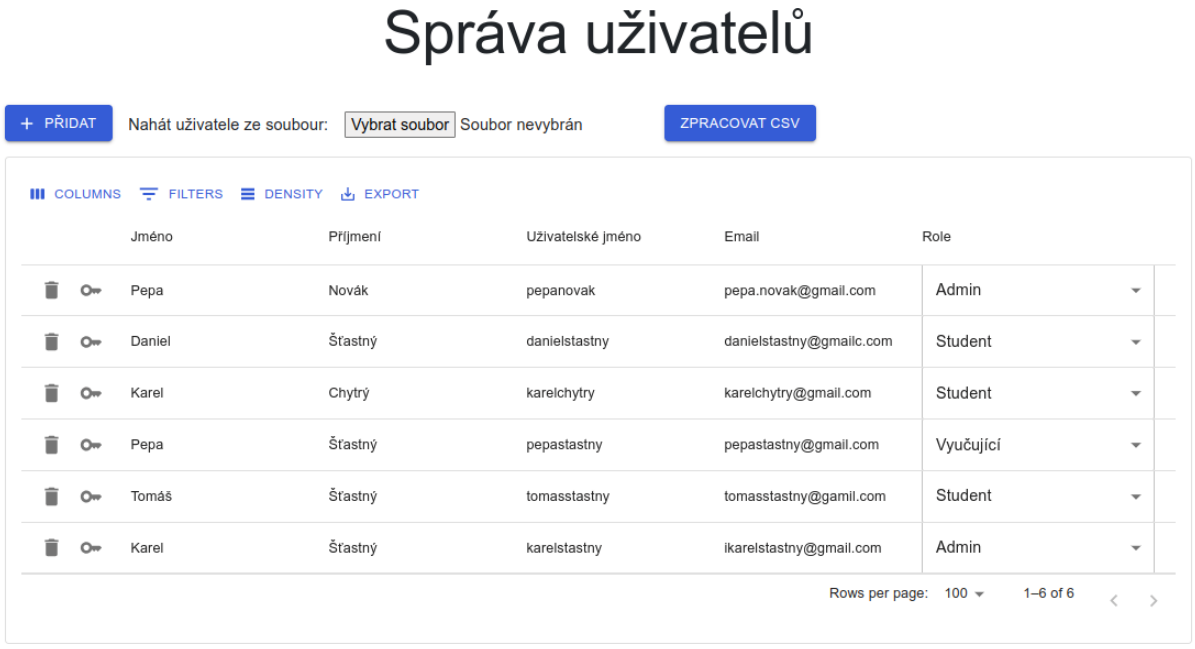
Poskytuje přehled dostupných skupin studentů v systému. Vyučující mají možnost vytvářet nové skupiny a spravovat již existující. Tato stránka umožňuje efektivní organizaci práce studentů ve skupinách, což usnadňuje spolupráci a sdílení zdrojových kódů mezi členy skupiny.



Obrázek 9 – Ukázka nabídky správy skupin (zdroj: vlastní)

7.4.3 Nabídka Uživatelé

Poskytuje přehled všech uživatelů, kteří mají přístup do systému. Vyučující a administrátoři mohou na této stránce vytvářet nové uživatele a spravovat jejich účty. Zde lze také nastavit role uživatelů (student, vyučující, administrátor) a přizpůsobit jim přístupová práva.



Obrázek 10 – Ukázka nabídky správy uživatelů (zdroj: vlastní)

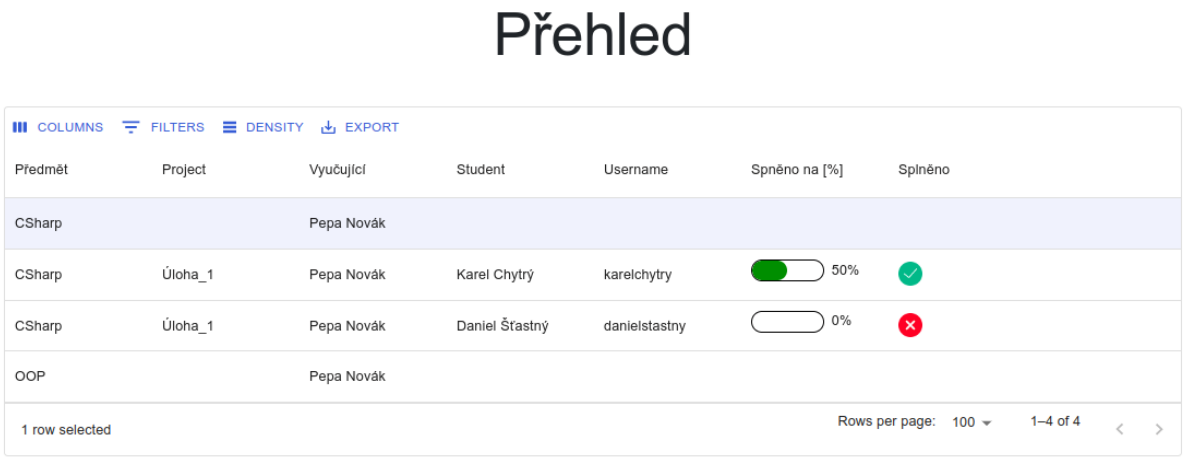
7.4.4 Nabídka Předměty

Poskytuje přehled všech předmětů dostupných v systému. Vyučující mají možnost vytvářet nové předměty a spravovat již existující. Tato stránka slouží k účinné organizaci výuky a přehlednému přidělování projektů a úkolů v rámci jednotlivých předmětů.

Obrázek 11 – Ukázka nabídky správy předmětů (zdroj: vlastní)

7.4.5 Nabídka Přehled

Nabízí podrobný přehled o jednotlivých předmětech a jejich projektech. Zde jsou zobrazeny výsledky testů z pipeline, které byly spuštěny na projektech studentů. Tato stránka umožňuje vyučujícím objektivně hodnotit kvalitu studentských prací a získat důležité informace o průběhu projektů.



Obrázek 12 – Ukázka nabídky pro přehledu studentských prací (zdroj: vlastní)

7.5 Testování systému

Vývoj a nasazení jakéhokoli nového systému je proces s řadou proměnných, vyžadující pečlivé plánování a realizaci. Klíčovými kroky tohoto procesu jsou validace a optimalizace systému, které zajišťují jeho schopnost efektivně a spolehlivě plnit stanovené úkoly a cíle. Pro dosažení tohoto cíle je nezbytné provést důkladné a promyšlené testování.

Testování probíhalo na zařízení s omezenými hardware zdroji. I přes tato omezení byl kladen důraz na optimální testovací podmínky s ambicí zajistit stabilní chod systému v různých testovacích scénářích. Pro simulaci skutečného uživatelského prostředí bylo vytvořeno 50 simulovaných uživatelských účtů, což umožnilo zkoumat chování systému v různých interakčních situacích. Dále byly vytvořeny čtyři specifické projekty zaměřené na různé aspekty a funkce systému. Během testování byla zkoumána například funkce vytváření fork projektů, což je časově a zdrojově náročná úloha. Přesto tento krok během testovací fáze úspěšně proběhl, což vedlo k vytvoření všech 200 fork projektů. Během testování bylo také ověřeno správné vyhodnocení studentských prací pomocí předpřipravených Gitlab Pipeline.

V průběhu testování byly identifikovány některé slabiny systému. Pokud by například všichni studenti odevzdávali práce současně, vyhodnocení za použití dvou Gitlab Runnerů by se mohlo stát výrazně časově náročným. Toto zpoždění je způsobeno omezeným počtem dostupných runnerů pro zpracování odevzdaných úloh. Tento problém může být řešen přidělením adekvátních hardwarových zdrojů. S dostatečnými zdroji lze situaci řešit dvěma způsoby: buď přidělením více operační paměti GitLab CE, anebo přidáním dalších Gitlab Runnerů, což by zvýšilo paralelní zpracování úloh a urychlilo vyhodnocení studentských prací.

Z provedeného testování bylo získáno mnoho cenných informací týkajících se chování systému v různých podmínkách. Tyto poznatky mohou sloužit jako výchozí bod pro budoucí testy a aktualizace, čímž přispějí k udržení aktuálnosti a efektivity systému.

8 KONFIGURACE A INSTALACE

8.1 Soubor .env

Slouží pro specifikaci všech proměnných pro daná prostředí tak, aby při spuštění kontejnerů byli nastaveny např. účty pro přihlášení k databázím a k řidící aplikaci. Jednotlivé systémové proměnné jsou popsány níže:

```
1.  APP_CONTAINER_NAME=app
2.  MYSQL_CONTAINER_NAME=mysql
3.  GITLAB_CONTAINER_NAME=gitlab-ce
4.  GITLAB_RUNNER_1_CONTAINER_NAME=gitlab-runner_1
5.  GITLAB_RUNNER_2_CONTAINER_NAME=gitlab-runner_2
```

Pomocí těchto proměnných jsou specifikovány názvy kontejnerů. Navíc tyto názvy slouží k identifikaci kontejner v počítačové síti (hostname), proto jsou nezbytné pro komunikaci všech služeb v rámci systému.

```
8.  # APP
9.  SERVER_PORT=8080
10. APP_JWT_SECRET=SecretKey
11. APP_JWT_EXPIRATION=86400
12. APP_ROOT_FIRSTNAME=Pepa
13. APP_ROOT_LASTNAME=Novák
14. APP_ROOT_USERNAME=pepanovak
15. APP_ROOT_EMAIL=pepa.novak@gmail.com
16. APP_ROOT_PASSWORD=nbio4e1D15jWPI8HonsVk69r0LZGU1
17. APP_LOG_FILE_PATH=/app/logs
18. APP_LOG_FILE_NAME=application-audit
19. APP_LOG_FILE_EXTENSION=.log
20. APP_LOG_MAX_FILE_SIZE=20MB
21. APP_LOG_MAX_HISTORY=10
22. APP_LOG_TOTAL_SIZE_CAP=500MB
23. PIPELINE_FILE_PATH=.gitlab-ci.yml
24. PIPELINE_FILES_PATH=/app/pipelines
25. PIPELINE_FILES_EXTENSIONS=yml
```

Proměnné sloužící ke konfiguraci řídící aplikace, ve které se pomocí následujících proměnných specifikuje:

- `SERVER_PORT` – definuje číslo síťového port, na kterém naslouchá a přijímá požadavky pro řídící aplikaci.
- `APP_JWT_SECRET` – JWT secret je tajný klíč, který slouží k podepisování a ověřování JWT tokenů. Tato hodnota by měla být dostatečně dlouhá a náhodná, aby

minimalizovala riziko odhalení a zneužití, a měla by být uchovávána v bezpečném a důvěryhodném prostředí, abychom zajistili integritu a autentizaci uživatelů v systému.

- `APP_JWT_EXPIRATION` – definuje dobu platnosti JWT tokenu. Tato hodnota je obvykle udávána v sekundách a stanovuje, jak dlouho je JWT token platný od jeho vytvoření. Správná hodnota by měla být zvolena s ohledem na bezpečnostní požadavky a funkčnost aplikace. Příliš krátká platnost může způsobit nežádoucí vypršení tokenu během běžné interakce s aplikací, zatímco příliš dlouhá platnost může zvýšit riziko útoků s ukradenými tokeny.
- `APP_ROOT_FIRSTNAME` – křestní jméno administrátora aplikace,
- `APP_ROOT_LASTNAME` – příjmení administrátora aplikace,
- `APP_ROOT_USERNAME` – uživatelské jméno administrátora aplikace, které slouží k přihlašování do systému,
- `APP_ROOT_EMAIL` – e-mailovou adresu administrátora aplikace,
- `APP_ROOT_PASSWORD` – heslo administrátora aplikace, které je používáno pro autentizaci a přístup k účtu,
- `APP_LOG_FILE_PATH` – definuje cestu, kde budou logovací soubory uloženy,
- `APP_LOG_FILE_NAME` – název logovacího souboru,
- `APP_LOG_FILE_EXTENSION` – přípona logovacího souboru,
- `APP_LOG_MAX_FILE_SIZE` – maximální velikost každého logovacího souboru,
- `APP_LOG_MAX_HISTORY` – definuje kolik logovacích souborů se má uchovávat,
- `APP_LOG_TOTAL_SIZE_CAP` – maximální velikost všech logovacích souborů,
- `PIPELINE_FILE_PATH` – definuje název a umístění souboru .gitlab-ci.yml v repozitáři,
- `PIPELINE_FILES_PATH` – odkud se budou načítat připravené pipeline konfigurace (zadaná cesta může být relativní nebo absolutní),
- `PIPELINE_FILES_EXTENSIONS` – obsahuje přípony souborů pro pipeline konfigurace. Ve výchozím nastavení je použita přípona yml, což odpovídá konfiguračnímu souboru .gitlab-ci.yml. Nicméně je možné libovolně přidat další přípony, pokud jsou používány jiné typy konfiguračních souborů pro pipeline. Alternativně lze také použít zástupný znak * pro načtení všech souborů bez ohledu na koncovku.

```
21.  # MySQL database
22.  MYSQL_ROOT_PASSWORD=FtntjS7kCvcRiShLfdJluAcfdkSA93
23.  MYSQL_USER=user
24.  MYSQL_PASSWORD=8kDPQ02092dspR2CyyfdHYhfsi7CFF
25.  MYSQL_DATABASE=test
```

Proměnné sloužící ke konfiguraci služby MySQL ve které se pomocí následujících proměnných specifikuje:

- `MYSQL_ROOT_PASSWORD` – nastavení přístupového hesla pro uživatele "root", který je automaticky vytvořen při instalaci služby MySQL, umožňuje uživateli plný přístup ke všem databázím a uživatelům, včetně práva měnit jejich přístupová práva.
- `MYSQL_DATABASE` – název databáze jenž bude obsahovat veškeré tabulky nezbytné pro správné fungování systému,
- `MYSQL_USER` a `MYSQL_PASSWORD` – definují přihlašovací údaje uživatele, na jejichž základě bude řídící aplikace vytvářet potřebné schéma pro běh aplikace a pod tímto uživatelem bude aplikace spravovat data v databázi.

```
27.  # Gitlab-ce
28.  GITLAB_EXTERNAL_DOMAIN=gitlab.example.com
29.  GITLAB_LISTENING_PORT=443
30.  GITLAB_LISTEN_HTTPS=true
31.  GITLAB_REDIRECT_HTTP_TO_HTTPS=true
32.  GITLAB_LETSENCRYPT_ENABLE=false
33.  GITLAB_REGISTRY_EXTERNAL_URL=https://registry.example.com
34.  GITLAB_REGISTRATION_TOKEN=Vi92kNPBSc8xsaJm99ek
35.  GITLAB_USER=root
36.  GITLAB_ROOT_PASSWORD= Q6Lq4JY0mp9BzJBiQePwH8qaitYcCA
37.  GITLAB_ROOT_EMAIL=myemail@example.com
38.  GITLAB_IGNORE_CERTIFICATE_ERROR=true
```

- `GITLAB_EXTERNAL_DOMAIN` – název domény, pod kterou je GitLab hostován jako externí služba,
- `GITLAB_LISTENING_PORT` – definuje číslo síťového port, na kterém GitLab naslouchá a přijímá požadavky,
- `GITLAB_LISTEN_HTTPS` – definuje, zda GitLab naslouchá na zabezpečené komunikaci pomocí protokolu HTTPS,
- `GITLAB_REDIRECT_HTTP_TO_HTTPS` – definuje, zda GitLab přesměrovává nezabezpečené HTTP požadavky na zabezpečený protokol HTTPS,
- `GITLAB_LETSENCRYPT_ENABLE` – definuje, zda je povoleno použití Let's Encrypt pro získání a obnovu SSL certifikátu pro doménu GitLabu,
- `GITLAB_REGISTRY_EXTERNAL_URL` – externí URL adresa pro GitLab Registry, která slouží k ukládání Docker obrazů,

- GITLAB_REGISTRATION_TOKEN – registrační token, který umožňuje registraci GitLab-runnerů do Gitlab,
- GITLAB_USER – uživatelské jméno administrátora GitLabu,
- GITLAB_ROOT_PASSWORD – heslo administrátora GitLabu,
- GITLAB_ROOT_EMAIL – e-mailovou adresu administrátora GitLabu,
- GITLAB_IGNORE_CERTIFICATE_ERROR – určuje, zda má být ignorováno chybové hlášení certifikátu SSL. Pokud má hodnotu true, GitLab bude ignorovat neplatné certifikáty SSL, což může být použito při testování nebo vývoji, ale není doporučeno pro produkční nasazení.

```
40.  GITLAB_LOGGING_LOGROTATE_FREQUENCY=weekly
41.  GITLAB_LOGGING_LOGROTATE_ROTATE=52
42.  GITLAB_LOGGING_LOGROTATE_COMPRESS=compress
43.  GITLAB_LOGGING_LOGROTATE_METHOD=copytruncate
44.  GITLAB_LOGGING_LOGROTATE_DELAYCOMPRESS=delaycompress
```

- GITLAB_LOGGING_LOGROTATE_FREQUENCY – definuje rekvenci rotace logovacích souborů systému GitLab. Díky tomu je možné nastavit různé časové intervaly pro rotaci logů. Alternativní hodnoty mohou zahrnovat:
    - "daily" - denní rotace,
    - "monthly" - měsíční rotace,
    - "yearly" - roční rotace.
- GITLAB_LOGGING_LOGROTATE_ROTATE – počet archivovaných logovacích souborů, které mají být uchovány systémem GitLab.
- GITLAB_LOGGING_LOGROTATE_COMPRESS – definuje, zda mají být archivované logovací soubory komprimovány nebo ne. Pokud jsou soubory archivovány dojde k vyšší úspoře místa na disku. Alternativní hodnoty mohou zahrnovat:
    - "compress" – komprese bude použita,
    - "nocompress" – komprese nebude použita.
- GITLAB_LOGGING_LOGROTATE_METHOD – definuje metodu rotace logovacích souborů kdy se staré logovací soubory nahrazují novými, aby se zabránilo nekontrolovanému růstu velikosti logů a zároveň se uchovávaly starší záznamy pro pozdější analýzu nebo audit. Alternativní hodnoty mohou zahrnovat:

- "copytruncate" – soubor s logy je nejprve zkopírován a poté je původní soubor zkrácen. Díky tomu se zachovají všechny stávající otevřené file handlery nebo procesy, které zapisují do logovacího souboru.
- "nocopytruncate" – při rotaci logů zahrnuje přejmenování nebo přesunutí původního logovacího souboru na nové místo s novým jménem, následované zkrácením původního souboru na nulu, čímž se zachovají stávající handle pro zápis dat a minimalizuje se riziko ztráty informací.

- `GITLAB_LOGGING_LOGROTATE_DELAYCOMPRESS` – definuje, zda má být komprese archivovaných logovacích souborů zpožděna. Jako alternativní hodnoty se používají:
  - "delaycompress" – logovací soubory, které byly přejmenovány nebo přesunuty během rotace (například při použití metody copytruncate), nebudou ihned komprimovány. Místo toho zůstanou nekomprimované, dokud není provedena další rotace logů. Teprve při další rotaci se provede komprese těchto souborů.
  - "nodelaycompress" – logovací soubory, které byly přejmenovány nebo přesunuty během rotace (například při použití metody copytruncate), budou ihned komprimovány.

```
46.  GITLAB_PROMETHEUS_MONITORING_ENABLE=false
47.  GITLAB_PROMETHEUS_LISTEN_ADDRESS=0.0.0.0:9090
48.  GITLAB_GRAFANA_ENABLE=false
49.  GITLAB_GRAFANA_ADMIN_PASSWORD=toomanysecrets
50.  GITLAB_GRAFANA_DISABLE_LOGIN_FORM=false
51.  GITLAB_GRAFANA_ALERTING_ENABLED=true
52.  GITLAB_GRAFANA_ALLOW_USER_SIGN_UP=true
53.  GITLAB_GRAFANA_SMTP_ENABLED=true
54.  GITLAB_GRAFANA_SMTP_HOST=localhost:25
55.  GITLAB_GRAFANA_SMTP_USER=nil
56.  GITLAB_GRAFANA_SMTP_PASSWORD=nil
57.  GITLAB_GRAFANA_SMTP_CERT_FILE=nil
58.  GITLAB_GRAFANA_SMTP_KEY_FILE=nil
59.  GITLAB_GRAFANA_SMTP_SKIP_VERIFY=false
60.  GITLAB_GRAFANA_SMTP_FROM_ADDRESS=admin@grafana.localhost
61.  GITLAB_GRAFANA_SMTP_FROM_NAME=Grafana
62.  GITLAB_GRAFANA_SMTP_EHLO_IDENTITY=dashboard.example.com
63.  GITLAB_GRAFANA_SMTP_STARTTLS_POLICY=nil
```

- `GITLAB_PROMETHEUS_MONITORING_ENABLE` – definuje, zda je povoleno monitorování pomocí Prometheus v GitLabu,

- GITLAB_PROMETHEUS_LISTEN_ADDRESS – síťová adresa a port, na kterém bude GitLab naslouchat pro monitorovací požadavky od Prometheus,
- GITLAB_GRAFANA_ENABLE – definuje, zda je povoleno používání monitorovacího nástroje Grafana v GitLab,
- GITLAB_GRAFANA_ADMIN_PASSWORD – heslo administrátora pro přístup k monitorovacímu nástroji Grafana v GitLab,
- GITLAB_GRAFANA_DISABLE_LOGIN_FORM – definuje, zda je povoleno přihlášení uživatelů pomocí přihlašovacího formuláře v monitorovacím nástroji Grafana,
- GITLAB_GRAFANA_ALERTING_ENABLED – definuje, zda je povoleno používání zasílání notifikací z monitorovacího nástroje Grafana,
- GITLAB_GRAFANA_ALLOW_USER_SIGN_UP – definuje, zda je povoleno uživatelům registrovat se do monitorovacího nástroje Grafana,
- GITLAB_GRAFANA_SMTP_ENABLED – definuje, zda je povoleno používání SMTP pro odesílání e-mailů z monitorovacího nástroje Grafana,
- GITLAB_GRAFANA_SMTP_HOST – adresu a port SMTP serveru, pomocí kterého se budou odesílat e-mail notifikace z monitorovacího nástroje Grafana,
- GITLAB_GRAFANA_SMTP_USER – uživatelské jméno pro přihlášení k SMTP serveru (pokud je vyžadováno),
- GITLAB_GRAFANA_SMTP_PASSWORD – heslo pro přihlášení k SMTP serveru (pokud je vyžadováno),
- GITLAB_GRAFANA_SMTP_CERT_FILE – definuje cestu k souboru s certifikátem pro zabezpečené připojení k SMTP serveru (pokud je vyžadováno),
- GITLAB_GRAFANA_SMTP_KEY_FILE – definuje cestu k souboru s privátním klíčem pro zabezpečené připojení k SMTP serveru (pokud je vyžadováno),
- GITLAB_GRAFANA_SMTP_SKIP_VERIFY – definuje, zda má být ověřování certifikátu SMTP serveru ignorováno,
- GITLAB_GRAFANA_SMTP_FROM_ADDRESS – e-mail adresa, která bude použita jako odesílatel pro e-mail notifikace z monitorovacího nástroje Grafana,

- GITLAB_GRAFANA_SMTP_FROM_NAME – jméno odesílatele, které bude zobrazeno u e-mail notifikací z montovacího nástroje Grafana,
- GITLAB_GRAFANA_SMTP_EHLO_IDENTITY – identita serveru, která bude odeslána jako součást EHLO příkazu během navazování spojení se SMTP serverem,
- GITLAB_GRAFANA_SMTP_STARTTLS_POLICY – definuje, politiku pro zabezpečené spojení přes STARTTLS. Výchozí hodnota nil znamená, že se použije standardní nastavení SMTP serveru. Alternativně lze použít hodnoty "never", "always", nebo "require", aby se specifikovalo, zda má být zabezpečené spojení povoleno, zakázáno nebo povinné.

```
65.  # Gitlab-runner
66.  GITLAB_RUNNER_IMAGE=gitlab-runner
67.  GITLAB_RUNNER_EXECUTOR=docker
68.  GITLAB_RUNNER_DOCKER_IMAGE=docker:latest
69.  GITLAB_RUNNER_DESCRIPTION=docker-runner
70.  GITLAB_RUNNER_CONFIG=/etc/gitlab-runner/config.toml
71.  GITLAB_RUNNER_MAINTENANCE_NOTE=Free-form maintainer notes about this runner
72.  GITLAB_RUNNER_TAG_LIST=docker
73.  GITLAB_RUNNER_RUN_UNTAGGED=true
74.  GITLAB_RUNNER_LOCKED=false
75.  GITLAB_RUNNER_ACCESS_LEVEL=not_protected
76.  TLS_VERIFY=false
77.  PRIVILEGED=false
78.  DISABLE_ENTRYPOINT_OVERWRITE=false
79.  OOM_KILL_DISABLE=false
80.  DISABLE_CACHE=false
81.  NETWORK_MODE="host"
```

- GITLAB_RUNNER_IMAGE – Docker image, pro GitLab Runner,
- GITLAB_RUNNER_EXECUTOR – definuje typ executoru, který GitLab Runner použije k vykonávání pipeline jobs,
- GITLAB_RUNNER_DOCKER_IMAGE – Docker image, který bude použit pro spuštění pipeline jobs v rámci "docker" executoru,
- GITLAB_RUNNER_DESCRIPTION – popis pro GitLab Runner instance,
- GITLAB_RUNNER_CONFIG – cesta ke konfiguračnímu souboru "config.toml" pro GitLab Runner, ve kterém jsou definovány další nastavení,
- GITLAB_RUNNER_MAINTENANCE_NOTE – volitelné poznámky pro správce týkající se GitLab Runner instance,

- GITLAB_RUNNER_TAG_LIST – seznam značek (tags), které jsou přiřazeny ke GitLab Runner instanci, a umožňuje tak ovlivnit, na které pipeline jobs bude tato instance reagovat,
- GITLAB_RUNNER_RUN_UNTAGGED – definuje, zda bude tato instanci GitLab Runner spouštět pipeline jobs, které nejsou označeny žádnou značkou (tags),
- GITLAB_RUNNER_LOCKED – definuje, zda bude tato GitLab Runner instance uzamčena. Pokud bude instance uzamčena nebude možnost spustit žádné nové pipeline jobs na této instanci Gitlab Runner.
- GITLAB_RUNNER_ACCESS_LEVEL – definuje úroveň přístupu ke GitLab Runner instanci. Tato proměnná může nabývat následujících hodnot:
    - not_protected – GitLab Runner nemá žádná omezení a může být používán k provádění jakýchkoli úkolů,
    - ref_protected – GitLab Runner může být používán pouze k provádění pipeline jobs na refencích, které nejsou chráněné,
    - project_protected – GitLab Runner může být používán pouze k provádění pipeline jobs na projektech, které nejsou chráněné,
    - private – GitLab Runner může být používán pouze k provádění pipeline jobs na projektech s přístupem nastaveným na "Private".
- TLS_VERIFY – definuje, zda bude GitLab Runner ověřovat platnost SSL certifikátů při komunikaci s GitLab serverem,
- PRIVILEGED – definuje, zda bude GitLab Runner spouštěn v privilegovaném režimu. V případě, že je tato proměnná nastavena na hodnotu true, GitLab Runner bude mít vyšší úroveň oprávnění a přístup k některým systémovým funkcím a zdrojům, které běžné kontejnery nemají. To může zahrnovat možnost manipulovat s síťovými nastaveními, zapisovat do některých systémových složek nebo provádět některé akce, které vyžadují vyšší úroveň oprávnění. Je třeba používat tuto funkci s opatrností, aby nedošlo k narušení bezpečnosti nebo stabilitě systému.
- DISABLE_ENTRYPOINT_OVERWRITE – Tato proměnná určuje, zda bude zakázáno přepisování výchozího entrypoint. Jedná se o spouštěcí skript nebo program, který je vykonáván při spuštění kontejneru.

- `OOM_KILL_DISABLE` – definuje, zda bude zakázáno automatické ukončení kontejneru při nedostatku operační paměti,
- `DISABLE_CACHE` – definuje, zda bude zakázáno používání cache při vykonávání pipeline jobs,
- `NETWORK_MODE` – Tato proměnná určuje režim sítě pro kontejner spuštěný GitLab Runnerem. Tato proměnná může nabývat následujících hodnot:
    - "host" – GitLab Runner sdílí síťový prostor s hostitelským strojem, což znamená, že běží v téže síti jako hostitelský systém a má přístup ke všem jeho síťovým rozhraním a službám.
    - "bridge" – GitLab Runner využívá výchozí síťové nastavení a vytváří vlastní izolovanou síť, která je oddělena od hostitelského prostředí. Tento režim umožňuje běžet kontejnerům ve vlastním síťovém prostředí.
    - "none" – GitLab Runner nemůže komunikovat s jinými službami ani s hostitelským prostředím.

8.2 Generování Self-Signed SSL Certifikátu pro GitLab

Použití certifikátu pro běh GitLab je nezbytné z důvodu zabezpečení komunikace mezi uživateli a GitLab serverem. SSL certifikát umožňuje šifrování dat, která jsou přenášena mezi prohlížečem klienta a webovým serverem (v tomto případě GitLab). To zajišťuje, že citlivé informace, jako jsou přihlašovací údaje nebo obsah repozitářů, jsou přenášeny po zabezpečeném kanálu a nemohou být snadno zachyceny nebo dešifrovány neoprávněnými osobami.

Bez použití certifikátu by data byla přenášena v otevřeném textovém formátu, což by znamenalo riziko odposlechu, manipulace nebo úniku citlivých informací. Certifikát také slouží jako důkaz identity serveru, čímž uživatelé mohou ověřit, že se opravdu připojují ke správnému GitLab serveru, a ne k nějakému neautorizovanému serveru podobného názvu.

Zabezpečený provoz důležitých aplikací, jako je GitLab, je nezbytným krokem v ochraně dat, zachování soukromí uživatelů a prevenci kybernetických útoků. Použití certifikátu SSL/TLS je standardní praxí v dnešním internetovém prostředí a zlepšuje celkovou bezpečnost a důvěryhodnost služby, kterou poskytuje GitLab.

```
82.  #!/bin/bash
```

Informace pro operačnímu systému, že má spustit tento skript pomocí interpretačního prostředí Bash.

```
 3.  DOMAIN="gitlab.example.com"
 4.  REG_DOMAIN="registry.example.com"
 5.  DIRECTORY="./volume_data/gitlab-ce/certs"
 6.  RSA_KEY_BITES=4096
 7.  EXPIRATION=365
 8.  COUNTRY=CZ
 9.  STATE=Prague
10.  LOCALITY=Pardubice
11.  ORGANIZATION=School
12.  COMMON_NAME=$REG_DOMAIN
13.  EMAIL_ADDRESS=school@gmail.com
```

Specifikace proměnných pro vygenerování self-signed SSL certifikátu mezi které patří:

- `DOMAIN` – název domény, pro kterou bude generován certifikát.
- `REG_DOMAIN` – název domény pro GitLab Registry, která bude zahrnuta do certifikátu.
- `DIRECTORY` – adresář, do kterého budou uloženy vygenerované certifikáty a klíče.
- `RSA_KEY_BITES` – délka klíče RSA v bitech. Čím větší hodnota, tím silnější klíč.
- `EXPIRATION` – platnost certifikátu v počtu dní.
- `COUNTRY` – kód země, ve které se nachází server.
- `STATE` – stát nebo region, kde se server nachází.
- `LOCALITY` – město nebo lokalitu, kde se server nachází.
- `ORGANIZATION` – název organizace, která spravuje server.
- `COMMON_NAME` – společné jméno (Common Name) certifikátu.
- `EMAIL_ADDRESS` – kontaktní e-mailovou adresu, která bude součástí certifikátu.

```
15.  rm -rf $DIRECTORY
```

Tímto příkazem se odstraní (rekurzivně) obsah adresáře určeného v proměnné `DIRECTORY`, pokud existuje, aby se vyčistil prostor pro nové certifikáty.

```
16.  mkdir -p $DIRECTORY
```

Tímto příkazem se vytvoří adresář, který byl definován v proměnné `DIRECTORY`.

```
17.  cd $DIRECTORY
```

Příkazem cd se změní aktuální pracovní adresář na adresář definovaný v proměnné `DIRECTORY`.

```
20.  openssl genrsa -out $DOMAIN.key $RSA_KEY_BITES
```

Tento příkaz vygeneruje privátní klíč pro server s názvem, který je uložen v proměnné `DOMAIN`. Klíč bude uložen v souboru s příponou ".key".

```
22.  openssl req -new -key $DOMAIN.key -out $DOMAIN.csr -subj "/C=$COUN-
     TRY/ST=$STATE/L=LOCALITY/O=$ORGANIZATION/CN=$COMMON_NAME/emai-
     lAddress=$EMAIL_ADDRESS"
```

Tento příkaz vygeneruje certifikační žádost (CSR) pro server. CSR je podpisový soubor, který obsahuje informace o serveru a jeho certifikátu, včetně země, státu, organizace, společného jména atd. CSR je podepsán soukromým klíčem serveru uloženým v souboru `$DOMAIN.key`.

```
24.  openssl x509 -req -extfile <(printf "subjectAltName=DNS:$DOMAIN,DNS:www.$DO-
     MAIN")  -days $EXPIRATION -in $DOMAIN.csr -signkey $DOMAIN.key -out $DO-
     MAIN.crt
```

Tento příkaz podepíše CSR vytvořený v předchozím kroku pomocí soukromého klíče a vytvoří samotný certifikát. Certifikát bude platný po dobu určenou v proměné `EXPIRATION` a bude obsahovat alternativní názvy DNS pro doménu. Výsledný certifikát bude uložen v souboru s příponou ".crt".

8.3 Instalované technologie

V rámci vývojových a testovacích fází systému byly využity následující technologie:

- Operační systém: Ubuntu 22.04.2 LTS
- Kontejnerizační nástroj: Docker, verze 24.0.5
- Správce závislostí a nástroj pro build projektů: Apache Maven, verze 3.8.7
- Pro backend řídící aplikace: Java JDK, verze 17
- Pro frontend řídící aplikace: Node.js, verze 20.5.0

ZÁVĚR

Cílem této diplomové práce bylo vytvořit komplexní systém umožňující automatizované vyhodnocování studentských prací s využitím moderní technologie Git. Práce se zabývala detailní analýzou a návrhem, které vedly k vytvoření funkčního řešení s mnoha praktickými výhodami.

Úvodní část práce se věnovala teoretickému základu, nezbytnému pro pochopení celého konceptu. Tato kapitola podrobně rozpracovávala různé pojmy, včetně World Wide Web, protokolů HTTP a HTTPS, významu domén, autorizace pomocí JWT tokenů, a samozřejmě i technologie Git a jejího významu pro sledování verzí kódu a spolupráci na projektech. Dále byly představeny koncepty spojené s kontinuální integrací a kontinuálním nasazením (CI/CD). Kapitola také detailně mapovala známé kolaborační platformy, jako jsou GitLab, GitHub, Bitbucket, Gitea, RhodeCode a Kallithea.

Druhá kapitola se zaměřila na hlubší analýzu dostupných kolaboračních serverů a technologických aspektů, které je provázely. Byly pečlivě rozebrány jednotlivé komponenty, tvořící tyto kolaborační servery, včetně jejich technologií a architektury. Kapitola také detailně zkoumala výhody a nevýhody těchto platforem a provedla jejich podrobné porovnání. Na základě této analýzy byl vybrán optimální kolaborační server, nejlépe vyhovující požadavkům na hostování Git repozitářů a automatizované vyhodnocování studentských prací.

Třetí kapitola se zaměřila na programové vybavení systému. Byly zde detailně popsány klíčové služby nezbytné pro provoz systému pro automatizované vyhodnocování studentských prací s využitím Git. Mezi tyto služby patří GitLab pro správu repozitářů a sledování verzí, GitLab Runner pro automatizované testování, MySQL Server a H2 pro ukládání dat a různé technologie pro řídící aplikace. Tato kapitola poskytuje komplexní pohled na programové prvky systému, které společně zajišťují jeho funkčnost a efektivitu.

Čtvrtá kapitola poskytla základní seznámení s nástrojem Docker a jeho klíčovými koncepty, zahrnujícími kontejnery, docker obrazy, dockerfile a docker-compose. V této části byly rozebrány výhody a nevýhody využití nástroje Docker pro budovaný systém automatizovaného vyhodnocování studentských prací pomocí Git.

Pátá kapitola se zaměřila na detailní analýzu samotného problému. Byly zde podrobně popsány různé aspekty, které byly dále použity k definování funkčních a nefunkčních požadavků. Kapitola obsahovala také vytvoření use case diagramu, zobrazujícího interakce mezi aktéry v rámci systému, a popis databázového modelu, sloužícího k ukládání dat.

Šestá kapitola se věnovala návrhu celého systému. Byl zde detailně popsán způsob, jak byly jednotlivé komponenty systému navrženy, tak, aby optimálně spolupracovaly. Kapitola zdůrazňovala, jak systém provádí vyhodnocování studentských prací, jak je zajištěna jeho zabezpečenost a jaké postupy jsou použity pro testování funkcí řídící aplikace.

Sedmá kapitola byla věnována praktické implementaci systému. Byly zde detailně popsány kroky pro nasazení jednotlivých komponent za pomocí konfiguračních souborů, včetně konfigurace kontejnerů a řídící aplikace. Kapitola také prezentovala celkovou strukturu projektu a detailně rozebrala jednotlivé služby a jejich konfiguraci. V rámci této kapitoly byla, kromě implementace, zvláštní pozornost věnována testování systému. Hlavním cílem testovací fáze bylo odhalit možné slabé stránky, prověřit výkon, stabilitu a vzájemnou kompatibilitu jednotlivých komponent. V rámci tohoto procesu byly použity různé testovací scénáře, které přinesly podrobný pohled na chování systému v reálném provozu. Z těchto testů vyplynuly klíčové poznatky, které poslouží v budoucnu jako vodítko k dalším vylepšením a optimalizacím.

Osmá kapitola se zabývala instalací a spuštěním systému. Byl zde poskytnut podrobný návod pro instalaci nástroje Docker na různých operačních systémech a popsán proces generování certifikátů, které jsou nezbytné pro správné fungování systému.

Poslední praktická kapitola představuje plně funkční systém pro automatizované vyhodnocování studentských prací s využitím technologie Git. Tento systém umožňuje rychlé vyhodnocení prací ihned po jejich odevzdání a vyniká svou modularitou pro snadné rozšíření o další funkcionality. Díky využití Dockeru je systém energeticky efektivní a škálovatelný, což umožňuje jeho provoz na samostatném serveru nebo s využitím orchestrace kontejnerů na více serverech.

Toto téma bylo mnou zvoleno zejména kvůli mé touze se podrobněji seznámit s moderními technologiemi, které jsou nutné pro provoz systému pro automatické vyhodnocení studentských prací a dále si rozšířit znalosti s platformou Docker. Všechny tyto znalosti mohou znamenat velký přínos pro mou budoucí práci se zaměřením na konfiguraci serverů.

POUŽITÁ LITERATURA

[1] Ing. Michaela, Petráková. Co je to WWW a jak funguje. *Collabim.cz. SEO Akademie Collabim*. [online]. 11. 04. 2023 [cit. 2023-08-07]. Dostupné z: https://www.collabim.cz/akademie/knihovna/co-je-to-www-a-jak-funguje/

[2] Barbora, Koďousková. HTTPS V KOSTCE: CO TO JE, JAK FUNGUJE A JAK NA NĚJ PŘEJÍT. *Rascasone*. [online]. 17. 11. 2021 [cit. 2023-08-07]. Dostupné z: https://www.rascasone.com/cs/blog/co-je-https-http-ssl-tls

[3] Jan, Dvořák. Slovníček IT pojmů: Co je doména? To je první krok k vlastnímu webu. *Webglobe*. [online]. 19. 08. 2022 [cit. 2023-08-07]. Dostupné z: https://www.webglobe.cz/blog/co-je-domena

[4] Suresh, Kumar. How JWT (JSON Web Token) authentication works? *DEV Community*. [online]. 25. 04. 2022 [cit. 2023-08-07]. Dostupné z: https://dev.to/kcdchennai/how-jwt-json-web-token-authentication-works-21e7

[5] Robert, Sheldon. Git. *IT Operations*. [online]. únor 2023 [cit. 2023-08-07]. Dostupné z: https://www.techtarget.com/searchitoperations/definition/GitX

[6] Isaac, Sacolick. What is CI/CD? Continuous integration and continuous delivery explained. *InfoWorld*. [online]. 15. 04. 2023 2023 [cit. 2023-08-07]. Dostupné z: https://www.infoworld.com/article/3271126/what-is-cicd-continuous-integration-and-continuous-delivery-explained.html

[7] Co je to GitLab a k čemu slouží? *techfocus.cz*. [online]. 29. 11. 2023 2023 [cit. 2023-08-07]. https://techfocus.cz/komercni-sdeleni/4655-co-je-to-gitlab-a-k-cemu-slouzi.html

[8] Ben, Lutkevich. GitHub. *IT Operations*. [online]. únor 2023 [cit. 2023-08-07]. Dostupné z: https://www.techtarget.com/searchitoperations/definition/GitHub

[9] Roja, Metla. What is Bitbucket? *EDUCBA*. [online]. 12. 04. 2023 [cit. 2023-08-07]. Dostupné z: https://www.educba.com/what-is-bitbucket/

[10] The Inedo Team. GITEA: How to Painlessly Self-Host Your Organization's Source Code. *Inedo*. [online]. 01. 08. 2023 [cit. 2023-08-07]. Dostupné z: https://blog.inedo.com/self-hosted/gitea/

[11] RhodeCode Enterprise. RhodeCode. *Enterprise 4.27.1 Documentation.* [online]. ©2022 [cit. 2023-08-07]. Dostupné z: https://docs.rhodecode.com/RhodeCode-Enterprise/index.html

[12] Kallithea Documentation. *Kallithea 0.7.99 documentation.* [online]. ©2023 [cit. 2023-08-07]. Dostupné z: https://kallithea.readthedocs.io/en/latest/index.html

[13] Scottem Chaconem. Documentation. Git [online]. 19. 07. 2023 [cit. 2023-08-07]. Dostupné z: https://git-scm.com/doc

[14] John Terra. Pull Request vs. Merge Request: Definition, Differences, and More. *Simplilearn.com*. [online]. 07. 08. 2023 [cit. 2023-08-07]. Dostupné z: https://www.simplilearn.com/pull-vs-merge-request-definition-differences-benefits-article

[15] Isaac Sacolick. What is CI/CD? Continuous integration and continuous delivery explained. *InfoWorld*. [online]. 15. 04. 2022 [cit. 2023-08-07]. Dostupné z: https://www.infoworld.com/article/3271126/what-is-cicd-continuous-integration-and-continuous-delivery-explained.html

[16] Tyler Hakes. (2023, March 21). GitHub issues: the what, why, and how. *7pace*. [online]. 12. 07. 2022 [cit. 2023-08-07]. Dostupné z: https://www.7pace.com/blog/github-issues

[17] Basalo, A. (2022, January 26). Documenting the project with a GitHub wiki. *Medium*. [online]. 15. 12. 2021 [cit. 2023-08-07]. Dostupné z: https://albertobasalo.medium.com/documenting-the-project-with-a-github-wiki-fe2465a84e3d

[18] GitHub Integrations: How to optimize your workflows. *Exalate*. [online]. 20. 07. 2023 [cit. 2023-08-07]. Dostupné z: https://exalate.com/blog/github-integrations/
[19] Tomas Martinčić. Structuring a CI/CD workflow in GitLab (Node.js example). *Lloyds digital*. [online]. 24. 01. 2022 [cit. 2023-08-07]. Dostupné z: https://lloyds-digital.com/blog/web-application-ci-cd-pipeline-architecture-gitlab
[20] Talha. (2022). Setting Up GitLab Webhooks: 4 Easy Steps. *Hevo*. [online]. 29. 12. 2022 [cit. 2023-08-07]. Dostupné z: https://hevodata.com/learn/gitlab-webhook/
[21] Daria Kulikova. GitLab Backup And Restore Best Practices [Step-by-step TUTORIAL]. *GitProtect.io.* [online]. 04. 08. 2022 [cit. 2023-08-07]. Dostupné z: https://gitprotect.io/blog/gitlab-backup-best-practices/
[22] GitLab. *GitLab documentation*. [online]. ©2023 [cit. 2023-08-07]. Dostupné z: https://docs.gitlab.com/
[23] Erica Lindberg. Announcing GitLab for DevSecOps. *GitLab*. [online]. 20. 07. 2019 [cit. 2023-08-07]. Dostupné z: https://about.gitlab.com/blog/2019/06/20/announcing-gitlab-devsecops/
[24] Fahd Agodzo Mohammed. System design of GitLab. *OpenGenus IQ: Computing Expertise & Legacy*. [online]. ©2023 [cit. 2023-08-07]. Dostupné z: https://iq.opengenus.org/system-design-of-gitlab/
[25] Zachary Flower. How to pick the right third-party CI/CD tool for the cloud. *Techtarget Data Center*. [online]. 11. 03. 2020 [cit. 2023-08-07]. Dostupné z: https://www.techtarget.com/searchcloudcomputing/tip/How-to-pick-the-right-third-party-CI-CD-tool-for-the-cloud
[26] Richard Silipigni. Git, GitHub, and GitLab: what's the difference? *BairesDev Blog: Insights on software development & tech talent.* [online]. 26. 10. 2022 [cit. 2023-08-07]. Dostupné z: https://www.bairesdev.com/blog/git-github-and-gitlab-whats-the-difference/
[27] Chowdary, C., & Chowdary, C. (2023). GitLab vs GitHub: Key Differences Between GitHub and GitLab. *Intellipaat Blog*. [online]. 26. 06. 2023 [cit. 2023-08-07]. Dostupné z: https://intellipaat.com/blog/gitlab-vs-github-difference/
[28] GitHub.com help documentation. *GitHub Docs*. [online]. ©2023 [cit. 2023-08-07]. Dostupné z: https://docs.github.com/en
[29] Bitbucket Server documentation. *Atlassian Documentation*. [online]. 08. 04. 2019 [cit. 2023-08-07]. Dostupné z: https://confluence.atlassian.com/bitbucketserver061/bitbucket-server-documentation-968674217.html
[30] Documentation. *GITEA Documentation*. [online]. ©2023 [cit. 2023-08-07]. Dostupné z: https://docs.gitea.com/
[31] Documentation. *RhodeCode Documentation*. [online]. ©2023 [cit. 2023-08-07]. Dostupné z: https://docs.rhodecode.com/
[32] Kallithea Documentation — *Kallithea 0.7.99 documentation*. [online]. ©2023 [cit. 2023-08-07]. Dostupné z: https://kallithea.readthedocs.io/en/default/
[33] Robert Sheldon. Ubuntu. *Techtarget Data Center*. [online]. srpen 2023 [cit. 2023-08-07]. Dostupné z: https://www.techtarget.com/searchdatacenter/definition/Ubuntu
[34] GitLab Runner. *EDUCBA*. [online]. 17. 04. 2023 [cit. 2023-08-07]. Dostupné z: https://www.educba.com/gitlab-runner/
[35] What is MySQL? A Beginner-Friendly explanation. *Kinsta* [online]. 03. 07. 2023 [cit. 2023-08-07]. Dostupné z: https://kinsta.com/knowledgebase/what-is-mysql/

[36] F.Marchioni. H2 Database Tutorial and expert Tips - Mastertheboss. [online]. 20. 01. 2023 [cit. 2023-08-07]. Dostupné z: https://www.mastertheboss.com/jbossas/jboss-datasource/h2-database-tutorial/?utm_content=cmp-true

[37] Spencer Bos. Java Basics: What is Spring Boot? *JRebel*. [online]. 05. 08. 2020 [cit. 2023-08-07]. Dostupné z: https://www.jrebel.com/blog/what-is-spring-boot

[38] Danielle Ellis. What is Swagger? A Beginner's Guide. *HubSpot Blog*. [online]. 26. 07. 2022 [cit. 2023-08-07]. Dostupné z: https://blog.hubspot.com/website/what-is-swagger

[39] Priya Pedamkar. What is Java Hibernate? *EDUCBA*. [online]. 26. 06. 2023 [cit. 2023-08-07]. Dostupné z: https://www.educba.com/what-is-java-hibernate/

[40] David Herbert. What is React.js? (Uses, Examples, & More). *HubSpot Blog*. [online]. 27. 06. 2022 [cit. 2023-08-07]. Dostupné z: https://blog.hubspot.com/website/react-js

[41] Tabnine Team. What is Material-UI and how to use it correctly? *The Official Tabnine Blog*. https://www.tabnine.com/blog/material-ui-and-how-to-use-it/ [online]. 17. 04. 2022 [cit. 2023-08-07]. Dostupné z: https://www.tabnine.com/blog/material-ui-and-how-to-use-it/

[42] Ranjani Harish. When should I use Project Lombok? *reflectoring.io*. [online]. 10. 04. 2022 [cit. 2023-08-07]. Dostupné z: https://reflectoring.io/when-to-use-lombok/

[43] gitlab4j/gitlab4j-api. *GitHub*. [online]. 15. 06. 2023 [cit. 2023-08-07]. Dostupné z: https://github.com/gitlab4j/gitlab4j-api

[44] Patel Viral. Create and Validate JWT Token in Java using JJWT - *ViralPatel.net*. [online]. 30. 05. 2020 [cit. 2023-08-07]. Dostupné z: https://www.viralpatel.net/java-create-validate-jwt-token/

[45] Rajesh Kumar. What is Maven and How it works? An Overview and Its Use Cases? *DevOpsSchool.com*. [online]. 16. 03. 2022 [cit. 2023-08-07]. Dostupné z: https://www.devopsschool.com/blog/what-is-maven-and-how-it-works-an-overview-and-its-use-cases/

[46] Lucero del Alba. What Is Docker? sitepoint [online]. 09. 12. 2022 [cit. 2023-08-07]. Dostupné z: https://www.sitepoint.com/what-is-docker/

[47] B. Arunachalam. (2023). What is Docker Compose? How to Use it with an Example. *freeCodeCamp.org*. [online]. 07. 04. 2023 [cit. 2023-08-07]. Dostupné z: https://www.freecodecamp.org/news/what-is-docker-compose-how-to-use-it/

[48] Manu Menon. Disadvantages of Containerization Docker. Bobcares. [online]. 26. 06. 2023 [cit. 2023-08-07]. Dostupné z: https://bobcares.com/blog/disadvantages-of-containerization-docker/

[49] GitHub Classroom. *Classroom*. [online]. ©2023 [cit. 2023-08-07]. Dostupné z: https://classroom.github.com/