Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Nástroj Continuous Integration pro školní úlohy

Bakalářská práce

2024 Michael Pinkas

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Michael Pinkas**
Osobní číslo:	**I21213**
Studijní program:	**B0688A140009 Informační technologie**
Téma práce:	**Nástroj Continuous Integration pro školní úlohy**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování

Cílem práce je vytvoření softwarového nástroje typu Continuous Integration pro podporu automatizovaného vyhodnocování školních úloh.
V teoretické části bude popsána problematika užití verzovacího systému Git ve školním prostředí pro odevzdávání programovacích úloh. Bude provedena rešerše možností samotného Gitu a dostupných platforem a nástrojů (GitHub, GitLab, ...). Dále bude představena technika Continuous Integration a možnosti její aplikace při automatizovaném vyhodnocování školních úloh.
V praktické části bude navržen a implementován nástroj (resp. rozšířen či upraven stávající CI nástroj) pro realizaci techniky Continuous Integration pro potřeby školních úloh a jejich automatizovaného vyhodnocování. Nástroj umožní konfiguraci pomocí textových souborů (JSON, YAML, apod.), kde budou specifikovány předměty, zadání úkolů, seznamy či pravidla pro nalezení konkrétních repozitářů s vypracovanými úlohami. Nástroj pak sám provede stažení vypracovaných úkolů, provede definované kroky k jejich kompilaci a otestování (resp. obecně vyhodnocení úlohy), zaznamená výsledky a případná selhání. Výsledkem bude připravený report ve formě HTML stránky či webu, kde budou přehledně zobrazeny výsledky úlohy a možnost zobrazení diffu změn provedených studenty při vypracování jednotlivých úloh proti definovanému zadání úlohy.

Rozsah pracovní zprávy: **(min. 30 stran)**
Rozsah grafických prací:
Forma zpracování bakalářské 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.
DUVALL, Paul M., Steve MATYAS a Andrew GLOVER. *Continuous Integration*. Pearson Education India, 2007. ISBN 9788131722916.

Vedoucí bakalářské práce: **Ing. Roman Diviš, Ph.D.**
Katedra softwarových technologií

Datum zadání bakalářské práce: **15. prosince 2023**
Termín odevzdání bakalářské práce: **10. května 2024**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

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

V Pardubicích dne 28. února 2024

Prohlašuji:

Práci s názvem Nástroj Continuous Integration pro školní úlohy jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

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

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

V Pardubicích dne 10. 05. 2024

Michael Pinkas v. r.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu této práce Ing. Romanu Divišovi, Ph.D. za cenné konzultace, rady, ochotu a trpělivost během zpracování této práce.

ANOTACE

Bakalářská práce se zabývá vytvořením nástroje pro automatizovanou kontrolu školních úloh. Teoretická část obsahuje popis verzovacího systému Git a existujících platforem založených na něm. Dále se zabývá koncepty Continuous Integration (CI) a Continuous Deployment (CD) a představuje nástroje používané pro tyto procesy. Je zde také diskutováno, jakým způsobem lze zmíněné technologie využít v prostředí školního vzdělávání. Praktická část pak aplikuje tyto znalosti pro vytvoření systému, který se skládá z řídící aplikace a nástroje Jenkins pro kontrolu studentských prací. Řídící aplikace dovoluje konfiguraci skrz soubor ve formátu YAML a poskytuje grafické rozhraní formou webových stránek pro zobrazení výsledků kontrol studentských prací.

KLÍČOVÁ SLOVA

Git, CI, CD, Jenkins, automatizované vyhodnocování školních úloh

TITLE

Continuous Integration tool for school assignments

ANNOTATION

The bachelor thesis deals with the creation of a tool for automated checking of school assignments. The theoretical part includes a description of the Git versioning system and existing platforms based on it. It also discusses the concepts of Continuous Integration (CI) and Continuous Deployment (CD) and introduces the tools used for these processes. It also discusses how these technologies can be used in a school environment. The practical section then applies this knowledge to create a system consisting of a management application and a Jenkins instance for checking student assignment solutions. The control application allows configuration through a YAML file and provides a graphical interface in the form of a web page to display the results of student solution checks.

KEYWORDS

Git, CI, CD, Jenkins, automated evaluation of school assignments

OBSAH

SEZNAM ILUSTRACÍ A TABULEK

SEZNAM ZKRATEK A ZNAČEK

AI	Artificial Intelligence
API	Application Programming Interface
CD	Continuous Delivery
CI	Continuous Integration
CSS	Cascading Style Sheets
DevOps	Development and Operations
HTML	Hypertext Markup Language
IDE	Integrated Development Environment
RCS	Revision Control System
REST	Representational State Transfer
SLA	Service Level Agreement
SVN	Apache Subversion
URL	Uniform Resource Locator
YAML	YAML Ain't Markup Language

ÚVOD

V dnešním době je běžné ulehčovat si práci automatizací repetitivních úloh. Školní prostředí nejsou pro toto téma výjimkou a existují procesy, které lze automatizovat. Proces automatizace lze zejména uplatnit v oborech IT, a to hlavně pro zadávání a vyhodnocování úkolů z předmětů zaměřených na programování. Vyučující těchto předmětů musí provádět manuální kontrolu vysokého množství studentských prací. Tento proces může být časově zdlouhavý a repetitivní. Vyučující si však mohou usnadnit práci automatizací těchto úloh. Tato práce se zabývá touto problematikou a nabízí její možná řešení.

V teoretické části této práce budou představeny verzovací systémy se zaměřením na distribuovaný verzovací systém *Git*. Dále budou představeny známé platformy, které poskytují hosting repozitářů pomocí tohoto verzovacího systému. Popsána bude technika CI/CD a nejrozšířenější nástroje, které umožňují implementaci této techniky. Pro tyto technologie bude popsáno, jak lze tyto nástroje využít pro zefektivnění a usnadnění kontroly studentských prací společně s možnostmi pro automatizaci vyhodnocování těchto prací.

V praktické části této práce bude navržen a implementován systém pro automatizované vyhodnocování školních úloh. Pro řešení budou využity technologie popsané v teoretické části této práce. Konkrétně bude implementovaný systém skládající se z aplikace pro jeho správu a řízení kontrol studentských repozitářů, které budou hostované na platformách založené na verzovacím systému *Git*. Tato aplikace bude využívat *MySQL* databázi pro ukládání výsledků testování a samotné testování bude probíhat v rámci CI nástroje *Jenkins*.

1 Git

1.1 Verzovací systémy

Git je distribuovaný verzovací systém. Verzovací systém slouží ke sledování změn v souborech různých typů. Využívá se pro kolaboraci rozsáhlých týmů, které mohou mít spolupracovníky rozprostřené po celém světě. Změny v souborech mohou být sledovány ve více paralelních verzích, které mohou být následně sloučeny, případně je možné se vracet ke změnám v historii. Nejčastější využití nachází verzovací systémy pro sledování změn ve zdrojových kódech programů, kde se mohou měnit jen malé části v každém souboru, ale může také sloužit pro sledování změn konfigurací, grafických objektů a webových stránek. [1][2]

Verzovací systémy se dělí na lokální, centralizované a distribuované. Pro lokální verzovací systémy existují data a soubory pouze na lokálním počítači. V případě, že dojde k selhání fyzického disku hrozí ztráta všech data a změna. Příkladem lokálního verzovacího systému je systém RCS. Centralizované verzovací systémy fungují na podobném principu, ale všechny změny jsou uložené na jednom centrálním serveru. Výhodou tohoto systému je možnost kolaborace s více lidmi. Ovšem všechny operace a historie jsou uložena na centrálním serveru, ke kterému je potřeba mít aktivní připojení. Opět pokud dojde k selhání serveru, jsou všechna data ztracena, jelikož je historie uložena pouze na straně serveru. Mezi centralizované verzovací systémy patří *Subversion* (SVN) nebo *Preforce*. Nejrozšířenější verzovací systémy v dnešní době jsou distribuované verzovací systémy. Největší výhodou těchto systémů je, že zdrojový repozitář je klonovaný lokálně úplně celý. Tím pádem všichni uživatelé mají vlastní kopii repozitáře a v případě selhání serveru je možné obnovit systém z jednoho klientského počítače. Výhodou je také, že není vyžadované aktivní připojení na server, jelikož většina operací je prováděna nejdřív lokálně na vlastní kopii repozitáře. Mezi nejpopulárnější implementace patří *Git* a *Mercurial*. [1]

1.2 Terminologie

1.2.1 **Repository (repozitář)**

Jedná se o složku, ve které se nachází všechny sledované soubory. Obsahuje také podsložku *.git* s metadaty a soubory repozitáře. [3]

1.2.2 Tree a working tree (strom a pracovní strom)

Strom je reprezentace libovolné verze repozitáře. Pracovní strom je strom, se kterým aktuálně pracujeme a skládá se tedy ze všech souborů, které se nacházejí v aktuálním stavu repozitáře. [3]

1.2.3 Branch (větev)

Jedná se o linii změn v repozitáři. Repozitář může obsahovat libovolný počet větví, avšak pracovní strom pracuje pouze s jednou větví. [3]

1.2.4 Commit (zaznamenaná sada změň)

Ve formě podstatného jména se jedná o jednu sadu změn verze repozitáře. Větve jsou tvořeny z jednotlivých commitů. Ve formě slovesa se jedná o provedení uložení sady změn do repozitáře. [3]

1.2.5 Tag (štítek)

Jednotlivé commity mohou být označený speciální objektem zvaným tag/štítek. Štítky jsou užitečné pro označení důležitých commitů. Ideální využití mají štítky, pokud vydáváme verzi aplikace a chceme si označit, které změny jsou zahrnuté ve vydané verzi. [3]

1.2.6 Head a HEAD

Ve formě psané malými písmeny se jedná o poslední commit ve větvi. Každá větev má jeden head commit. Ve formě psané velkými písmeny se jedná o commit, ve kterým se aktuálně nacházíme. Je pouze jeden a je aktuální pro pracovní strom. Většinou HEAD odkazuje na jeden z head commitů, avšak může odkazovat na commit libovolný. [3]

1.2.7 Object (objekt)

Jedná se o soubor, ve kterém si Git ukládá data. Objekt může být buď *commit*, *tree*, *tag* anebo *blob*. [3]

1.2.8 Blob

Blob je objekt, který nemá typ. Většinou se jedná o obsah souboru v repozitáři. [3]

1.2.9 Hash a object name (hash a název objektu)

Pro *Git* jsou tato slova zaměnitelná a znamenají to samé. Jedná se o jedinečný identifikátor objektu, který se vytváří hashovací funkcí SHA-1. [3]

1.3 Charakteristika Gitu

Git je nejrozšířenějším a nejpoužívanějším verzovacím systémem na světě. Vznikl v roce 2005 pro vývoj jádra operačního systému *Linux*. Zakladatelem *Gitu* je Linus Torvalds. Předlohou pro vývoj *Gitu* byl distribuovaný verzovací systém *BitKeeper*. *BitKeeper* byl verzovací systém využívaný pro vývoj Linuxového jádra, avšak změny v licenci a zpoplatnění přiměli *Linuse* k vytvoření *Gitu*. Hlavním cílem vzniku *Gitu* tedy byla plnohodnotná náhrada *BitKeeper* s vylepšením nedostatků, které si komunita při vývoje *Linuxového* jádra všimla. Git se tedy stal distribuovaným systém, který je rychlý a jednoduchý. Kladl se i důraz na optimalizaci pro veliké projekty s podporou vysokého počtu kolaborantů a jejich současným vývojem ve velkém počtu paralelních větví. [1][4][5]

1.4 Hlavní rozdíly s ostatními verzovacími systémy

Ostatní verzovací systémy ukládají změny mezi jednotlivými soubory. Ačkoliv se tento může zdát jako vhodný, přináší své nevýhody. Pro vytvoření rozdílu mezi dvěma soubory je potřeba nejdřív tyto soubory identifikovat. Některé verzovací systémy k této problematice přistupuji naivně a identifikují jednotlivé soubory podle jejich jmen. Během vývoje je ovšem běžné, že dochází k úpravě struktury repozitářů a krom přesouvání souborů mezi tyto úpravy patří i přejmenování souboru. Takováto úprava by mohla být vyobrazena jako odstranění přejmenovaného souboru a vytvoření nového. *Git* k této problematice přistupuje lépe a řídí se samotným obsahem souboru. *Git* si vytvoří kontrolní součet pro obsah souboru pomocí hashovací funkce, a právě tuto hash používá pro jeho identifikaci. Díky této vlastnosti jsou úpravy souborů snadno detekovatelné, včetně úprav historie. To znamená, že není snadné upravit historii souborů, aniž by se nezměnila hash těchto souboru. [1]

Rozsáhlé projekty mají dlouhou životnost, a tedy pro jeden soubor může existovat v rámci jednoho projektu stovky úprav. Zobrazení úprav pro soubor s mnoha úpravami je náročná operace v případě, že uchováváme jednotlivé změny jako jejich rozdíl. Museli bychom výsledný soubor sestavit jako posloupnost všech těchto změn. *Git* tento problém řeší uchováváním snapshotů (snímků) aktuálního stavu jednotlivých souborů. Pro optimalizaci místa jsou pro každou verzi projektu uloženy pouze změněné soubory a v případě, že obsah neobsahuje změny, *Git* si uloží pouze referenci na soubor v původní verzi projektu. Jako reference se opět používá hash souboru. [6]

2 Platformy pro správu verzí založené na Gitu

Git sám o sobě není službou, která by umožňovala centrální ukládání a sdílení kódu. Pro tyto potřeby existují platformy, které podporují hosting repozitářů přístupných z internetu. Většinou se jedná o služby poskytované zdarma a dovolují správu sdílených repozitářů. Mezi nejznámější poskytovatele patří *GitHub*, *GitLab*, a *Bitbucket*. Tyto platformy nabízejí uživatelům široké možnosti správy kódu včetně sledování změn, správy verzí a spolupráce v týmu. Díky nim lze efektivněji pracovat na projektech a snadno sdílet kód mezi vývojáři. V této kapitole budou popsány vlastnosti platforem pro správu verzí, jaké možnosti nabízejí a jejich porovnání.

2.1 Běžné vlastnosti

2.1.1 Revize změn

Revize změn neboli code review, je klíčovou vlastností platforem nabízející ve vývojovém procesu softwaru. Tento proces umožňuje zobrazit a přezkoumávat navržené změny ve zdrojovém kódu před jejich začleněním do hlavní větve repozitáře. Hlavním cílem revize změn je zlepšit kvalitu kódu, zajištění konzistence a minimalizace chyb. Společně se zobrazením jednotlivých změn bývá možnost vést diskuze, které mohou být použity pro poskytnutí zpětné vazby, navržení vylepšení, identifikace potenciálních problémy. Diskuze také mohou vést k lepšímu porozumění kódu. [7]

2.1.2 Sledování požadavků

Tento nástroj umožňuje týmu sledovat a řešit nalezené chyby, žádosti o nové funkce, úkoly nebo jiné problémy v rámci projektu. V běžné praxi se pro tento nástroj vyžívá slovní spojení „issue tracking" a nahlášené chyby, nebo žádosti o nové funkce se nazývají „issues". Hlavním cílem sledování požadavků je zajistit, aby všechny potenciální problémy byly identifikovány, zaznamenány a řešeny systematicky a efektivně. Sledování požadavků pomáhá zlepšit transparentnost, komunikaci a řízení vývoje projektu. Zároveň umožňuje týmu upřednostňovat úkoly s vyšší prioritou. Jednotlivé issues nemusí vytvářet pouze vývojáři. Některé platformy dovolují vytvářet issues uživatelům. Někdy je však tato funkce přístupná pouze po přihlášení ke svému účtu. Další výhodou je, že issues se dají připojit k revizi změn, což dále zpřehledňuje a dává lepší kontext novým změnám. [8]

2.1.3 Web hosting

Některé platformy umožňují uživatelům vytvářet a hostovat statické HTML stránky. Jedná se o snadný způsob, jak sdílet statický obsah nebo webové stránky spojené s projektem. Tato

funkcionalita je užitečná pro prezentaci projektových stránek, osobních portfolií nebo jiných užitečných informací. Uživatelé mohou jednoduše vytvořit HTML, CSS a JavaScript soubory, nahrát je do svého repozitáře a ty jsou poté automaticky zpřístupněné na URL repozitáře a zobrazeny formou webové stránky. [9][10]

2.1.4 Wiki

Wiki stránky jsou užitečným nástrojem pro sdílení informací a znalostí v rámci projektů, komunit nebo organizací. Tyto stránky umožňují členům týmu vytvářet, upravovat a sdílet dokumentaci, návody, procedury, odkazy nebo jakýkoli jiný obsah, který je relevantní pro daný projekt nebo téma. Hlavní výhodou wiki stránek je, že umožňují snadnou a dynamickou úpravu obsahu přímo v prohlížeči, bez potřeby speciálního softwaru nebo znalosti složitých formátovacích jazyků. Týmy mohou vytvářet strukturované hierarchie stránek, propojovat je mezi sebou, vkládat obrázky, tabulky, kódy nebo další multimediální prvky a snadno vyhledávat a procházet obsah.

2.1.5 Integrace s CI/CD systémy

Integrace s nástroji pro kontinuální integraci a doručování (CI/CD) je klíčovým prvkem efektivního softwarového vývoje. CI/CD nástroje umožňují automatizovat procesy testování, sestavení a doručení softwarových aplikací, což vede k jejich rychlejšímu dodání. Integrace s CI/CD nástroji umožňuje automatické spuštění CI/CD pipeline při každém provedení změn ve zdrojovém repozitáři. Některé platformy poskytují CI/CD nástroje v rámci nabízených služeb.

2.1.6 Self-hosted instance

Tato služba poskytuje organizacím možnost provozovat instanci hostovací platformy na vlastních serverech. Tento přístup umožňuje plnou kontrolu nad daty a bezpečností a zároveň poskytuje všechny funkce a výhody, které nabízejí cloudové varianty těchto platforem. Hlavní výhodou hostování platforem na vlastních serverech je možnost dodržovat interní bezpečnostní politiky a předpisy, což je klíčové pro organizace s vysokými standardy v oblasti ochrany dat a soukromí. Díky internímu hostování je možné efektivněji spravovat přístupová práva, monitorovat aktivity uživatelů a integrovat se s dalšími interními nástroji a systémy. Další výhodou mohou být výhodnější cenové balíčky, jelikož firma platí za licenci hostovací platformy, a ne za služby, které by získala z cloudové verze. Těmito službami se myslí limitace ve formě maximálního počtu změn v projektu nebo množství přenesených dat.

2.1.7 Podpora AI

AI se stává stále výraznějším prvkem v softwarovém vývoji a ani platformy pro správu verzí nezůstávají pozadu v poskytování podpory a nástrojů v této oblasti. Jedním příkladem AI nástroje je automatické zpracování a doplňování kódu. Vývojáři díky těmto nástrojům mohou věnovat více energie do řešení problému, nebo návrhu nových funkcionalit a nemusí věnovat čas vytvářením a čištěním kódu. Další nedílnou vlastností AI nástrojů je podpora vysvětlení částí kódu, nebo generace jednotkových testů pro vybranou část kódu. [11][12]

2.2 Platformy

V následujících kapitolách jsou popsány nejznámější platformy poskytující systémy pro správu verzí.

2.2.1 GitHub

GitHub vznikl v roce 2009 a je jeden z nejpopulárnějších a nejpoužívanějších nástrojů pro správu verzí kódu. Jedná se o oblíbenou platformu jak pro open-source projekty, tak i pro soukromé projekty v rámci firem a organizací. Mezi hlavní služby, které jsou poskytovány patří hostování repozitářů, možnost zobrazení změn, sledování požadavků (issue tracking), hostování webových a wiki stránek. V podobě *GitHub Actions* přibyla možnost využívat nativní CI/CD nástroje v rámci *GitHubu*. *GitHub* má také podporu AI funkcionalit díky nástroji *GitHub Copilot*. Tento nástroj je možné integrovat ve všech nejpoužívanějších IDE editorech, jako je *Visual Studio*, *Visual Studio Code*, nebo editory z ekosystému *JetBrains*. [12][13][14][15]

Většina funkcionalit *GitHubu* je poskytována v rámci bezplatné edice. Ta oproti jiným platformám dovoluje využití *GitHubu* zdarma i pro organizace a nejen jednotlivce. Placené edice umožňují uživatelům širší možnosti pro správu oprávnění nad repozitáři. Dále je zvýšen maximální limit počtu minut pro využívání *GitHub Actions* a velikost úložiště pro hostované repozitáře. V případě podnikové edice je možná instalace *GitHubu* na vlastní infrastruktuře. [16]

2.2.2 GitLab

GitLab vznikl v roce 2011 jako následovník *GitHubu* a v současnosti se stal jeho největším konkurentem. Není však orientovaný primárně jen na hostování repozitářů, ale byl od počátku tvořen s myšlenkou podpory DevOps a CI/CD nástrojů. Mezi hlavní služby, které jsou poskytované *GitLabem*, patří hostování repozitářů, sledování změn v kódu, sledování požadavků (issue tracking), hostování webových stránek, vytváření wiki stránek pro dokumentaci projektů a další. Mezi ostatními platformami však hlavně vyčnívá širokou

podporou CI/CD nástrojů. *GitLab* také nezaostává v podpoře AI nástrojů a poskytuje uživatelům *GitLab Duo*. *GitLab Duo* je obdobně jako *GitHub Copilot* dostupné pro nejrozšířenější IDE editory, jako je *Visual Studio*, *Visual Studio Code* a editory společnosti *JetBrains*. [14][15]

Bezplatná edice je přístupná pouze pro jednotlivce a obsahuje většinu základních funkcionalit. V rámci bezplatné edice je také možné hostovat instanci *GitLabu* na vlastních serverech. Placené edice navyšují úložný prostor pro repozitáře, možné množství přenesených dat a dostupný čas pro spouštění CI/CD pipeline. Dále rozšiřuje možnosti nastavení uživatelských oprávnění a rozšířené možnosti pro CI/CD nástroje. Podniková edice navíc poskytuje řadu nástrojů pro sledování zranitelností a závislostí v kódu. [11][17]

2.2.3 BitBucket

BitBucket vznikl v roce 2008 s podporou verzovacího systému *Mercurial* a v roce 2011 po odkoupení společností *Atlassian* přibyla podpora *Gitu*. Stejně jako předešlé platformy *BitBucket* poskytuje hostování repozitářů, sledování změn, sdílení wiki a webových stránek a vestavěnou integraci CI/CD nástrojů. Jeho největší předností však je, že je součástí ekosystému aplikací od společnosti *Atlassian* a má tak nativní podporu integrace s nástroji jako je *Jira*, *Confluence* a dalšími. [14][15]

Základní edice není omezena podle toho, zda je vlastníkem jednotlivec nebo organizace, ale je omezena maximálním počtem uživatelů, který činí 5 osob. Placené edice se liší velikostí prostoru pro repozitáře a počtem minut pro spouštění CI/CD pipeline. Prémiová edice umožňuje dvou fázové ověření, konfiguraci povolených IP adres s možností nastavení práv pro jednotlivé nasazení a je poskytované SLA pro 99% dostupnost. *BitBucket* umožňuje provoz vlastní instance, avšak pouze s placené edicí. [18]

2.2.4 Porovnání

Tabulka *1* – Porovnání Git platforem [14][15]

Funkce	GitHub	GitLab	BitBucket
Revize změn	✓	✓	✓
Sledování požadavků	✓	✓ Vyžaduje přihlášení	✓

Web hosting	✓	✓	✓
Wiki	✓	✓	✓
Nativní podpora CI/CD	✓	✓	✓
Self-hosted instance	✓ Vyžaduje placenou edici	✓	✓ Vyžaduje placenou edici
Soukromé repozitáře	✓	✓	✓ Maximálně 5 uživatelů

2.3 Využití ve školním prostředí

Platformy pro hostování repozitářů vznikly s hlavní vidinou podpory vývoje rozsáhlých softwarových řešení. To ale samozřejmě není jediné vhodné využití těchto systémů. Tyto systémy je možné efektivně využít i ve školním prostředí. Obzvlášť v oborech se zaměřením na informační technologie. V průběhu studia se studenti mohou setkat s předměty, kde je zadání předáno pouze formou textových souborů prostřednictvím *Moodle*, *IS/STAG* případně jiným systémem, který škola využívá. Následná kontrola prací pak může probíhat individuální prezentací a odevzdáním práce opět prostřednictvím těchto systémů. Pro prostředí, kde pro každé zadání je odevzdáno vyšší jednotky desítek prací není tento přístup ideální. Vyučující musí provést stažení všech prací ze systému a následně všechny práce projít. To může vyžadovat otevřít práci v IDE prostředí, nebo manuálně projít soubory pomocí textového editoru.

Tyto všechny problémy se dají obecně řešit využitím verzovacích systémů a platforem na nich založených. Vyučující mohou pro distribuci zadání využít sdílený repozitář. Výhodou pro zadání sdílené skrz repozitář hostovaný na některé z výše uvedených platforem je možnost úpravy zadání s viditelnou historií změn.

V případě, že by došlo k nalezení nějaké chyby v zadání, vyučující může snadno provést úpravu a student uvidí, co a kde se přesně měnilo. Tedy si může změnu integrovat do svého řešení bez nutnosti komplikovaných předání informací, co a kde je potřeba změnit. Ve stejném duchu mohou studenti odevzdávat svá řešení zadaných úkolů. Kontrola úkolu může být mnohem snazší. U jednoduchých úkolů může stačit pouze revize odevzdaného kódu přečtením jednotlivých změn. Výše uvedené platformy poskytují možnost zobrazení si jednotlivých změn, což vyučujícím může ulehčit práci. V případě, sdílení řešení úkolů prostřednictvím souborových archivů, je nutné zkontrolovat všechny soubory a dané změny hledat. Mít sdílené řešení dostupné skrz internet je také vhodné pro konzultace mezi studentem a vyučujícím. Student nemusí nijak zasílat celý projekt, nebo úryvky kódu se kterým potřebuje pomoci a vyučující se může snadno podívat na celé řešení studenta, ulehčující domluvu a rychlost vyřešení studentova problému.

Je ale dobré upozornit na možnost, že lze upravovat historii commitů a jejich obsah. Vyučující si však pomocí skriptu může naklonovat všechna řešení studentů na své lokální úložiště krátce po termínu odevzdání. Komplexnější řešení by bylo sledovat SHA-1 hash jednotlivých commitů, ale to by značně komplikovalo jednoduchost řešení.

3 CI/CD nástroje

Předešlé kapitoly byly věnovaný platformám založených na verzovacím systému *Git*. Využití těchto systémů dovoluje efektivní sdílení a verzování kódu mezi členy vývojových týmů, bez ohledu na to, jestli se jedná o tým lidí v jedné kanceláři, nebo po celém světě. V rámci životních cyklů vývoje softwaru, ale sdílení a změny kódu nejsou vše. Další fází často bývá testování provedených změn. Je důležité ověřit, že dané změny nic nerozbíjí a aplikace funguje stále stejně, nebo lépe. Mezi základní kroky patří ověření, že projekt jde sestavit, spuštění jednotkových testů, případně v reakci na předešlé kroky může následovat spuštění dalších testů jako jsou testy integrační. Verzovací systémy nedisponují možnostmi vykonávání těchto činností automaticky. To znamená, že si každý tým musí hlídat provádění těchto kroků manuálně. Tento přístup však není vhodný v prostředích, kdy během hodiny může být vytvořeno desítky změn. Znamenalo by to zároveň vytvoření určitě zodpovědnosti, že postup otestování dané změny proběhl naprosto stejně a kvalitně jako v minulosti. Mohlo by se stát, že by změny nebyly zkontrolovány adekvátně, nebo dokonce vůbec. Tím vzniká riziko, že by neotestované mohly být začleněny do hlavních větví repozitáře a případně distribuovány ve verzích aplikace. [19]

3.1 CI/CD Pipeline

CI/CD pipeline představuje implementaci procesů umožňující průběžnou integraci a doručování kódu. Tento proces je strukturován do fází CI (*Continuous Integration*) a CD (*Continuous Delivery*), které obsahují definované kroky a úlohy. V první fázi pipeline se provádí sestavení projektu a spuštění testů, přičemž může docházet i k analýze kódu, což zahrnuje pokrytí kódu jednotkovými testy, identifikaci možných zranitelností v závislostech projektu a detekci nekvalitního kódu. Následuje fáze doručení, která zahrnuje vytvoření instalačních balíčků, a nakonec fáze nasazení, během níž se aplikace nasadí do cílového prostředí. Tento strukturovaný přístup pomáhá automatizovat a zefektivnit proces vývoje a dodávání softwaru. Pipeline je definice více kroků nebo úloh. Příkladem jednoduché pipeline mohou být následující kroky: kompilace projektu, spuštění jednotkových testů, export výsledků testů a míry pokrytí projektu testy a následné spuštění další pipeline, která vytvoří instalátor. [21][22]

Obrázek *1* – Fáze CI/CD procesu [21]

3.2 Continuous Integration

Nástroje pro průběžnou integraci (C*ontinuous Integration*) dále jen zkráceně CI, slouží pro automatizaci, zefektivnění a zvýšení kvality vývoje softwaru. Pomáhají průběžné integrovat a testovat kód, což umožňuje rychlejší odhalení a opravu chyb, a tím i zlepšení celkové spolehlivosti a stability aplikace. Také urychlují vývoj nových funkcí a jejich doručení do nových verzí aplikace. Koncem CI pipeline bývá začlenění nových změn do hlavní větve repozitáře. [19]

Nástroje CI sledují změny ve zdrojovém repozitáři a při detekci úprav spouští vydefinované kroky. Změny mohou být obvykle sledovány dvěma způsoby. První způsob sledování změn je pomocí pravidelných kontrol zdrojového repozitáře. Druhým způsobem je použití webhooků, kdy je repozitář nastaven tak, aby automaticky oznámil CI nástroji každou novou změnu. To minimalizuje reakční dobu CI nástroje. V reakci na detekované změny jsou spuštěny pipeline. Každý projekt může mít definovaných více pipeline a mohou být spuštěny automaticky buď pomocí nějaké spouště, nebo manuálně. [19]

3.3 Continuous Delivery

Nástroje pro průběžné doručování (*Continuous Delivery*) dále jen CD, jsou rozšířením procesů CI. CI pipeline dovoluje efektivní způsob automatizace vývoje softwaru skrz testování a začlenění nových změn v rámci zdrojového repozitáře. CD část pipeline je využívána pro vytvoření instalačních balíčků, artefaktů a případně přípravu a nasazení do testovacího prostředí. Proces nasazení do prostředí může být složitý a časově náročný, obzvláště pokud došlo ke změnám v aplikaci vyžadujícím úpravy v konfiguraci nebo schématu databáze. Manuální nasazení by mohlo přinést nechtěné chyby – operátor, který provádí nasazení, může přehlédnout nastavení nových konfiguračních parametrů a aplikace by tak mohla selhat při spuštění. Tímto by operátor buď musel odhalit a opravit chybu sám, nebo se obrátit na vývojový

tým o pomoc při řešení problému. V procesu odhalování chyb by mohl operátor neúspěšně experimentovat s dalšími konfiguračními parametry, což zvyšuje riziko selhání celého nasazení. Je zřejmé, že manuální procesy nesou příliš mnoho rizik, která mohou překazit úspěšné nasazení nové verze aplikace. [19]

V rámci CD pipeline lze celý tento proces automatizovat. S CD nástroji lze definovat a spravovat různé prostředí (např. vývojové, testovací) a provádět nasazení nových verzí aplikace s minimálním manuálním zásahem. Díky tomu se snižuje riziko chyb při nasazení a zvyšuje se rychlost doručování nových funkcí a aktualizací zákazníkům. Pipeline pro nasazení aplikace může mít vydefinované nové konfigurace, či změny v databázi, a to pro všechny druhy prostředí. Celá pipeline může být otestována při nasazení na testovací prostředí a pro produkční prostředí může stačit změnit pouze hodnoty konfigurační parametrů. Pipeline pro nasazení se může skládat z následujících úloh: CI část pipeline, vytvoření instalátorů, připojení na cílové prostředí, vypnutí běžící aplikace, aktualizace konfigurace, aktualizace aplikace, spuštění aplikace, provedení smoke testu pro ověření správnosti běhu aplikace na reálném prostředí. [19]

3.4 Continuous Deployment

V předcházejících kapitolách byly zavedeny pojmy pro kontinuální integraci a doručování. Vedle kontinuálního doručování (*Continuous Delivery*) existuje ještě pojem kontinuální nasazení (*Continuous Deployment*). Oba pojmy se označují zkratkou CD, ale nemají úplně stejný význam. Jejich hlavním rozdíl je, že v případě kontinuálního doručování existuje kontrola nad tím, zda dojde k nasazení aplikace. To znamená, že se spustí pipeline, ale automaticky nepokračuje v kroku pro nasazení změn na prostředí zákazníka. Pipeline je pozastavena po fázi příprav nasazení. Naopak u kontinuálního nasazení dochází k automatickému potvrzení tohoto kroku. To znamená, že všechny provedené změny jsou automaticky promítnuty na prostředí zákazníka. Tento přístup je vhodný například při začátku vývoje aplikace, a umožňuje vývojovému týmu přímou kontrolu, jak se aplikace chová. Pro aplikace, které jsou již delší dobu v provozu se tento přístup využívá v modelu, kdy vývojový cyklus nemá termíny dodání nových funkcionalit, ale jsou nasazeny okamžitě po jejich implementaci. Zákazník tak může poskytnout zpětnou vazbu k provedeným změnám v rámci hodin. Výhodou také je, že dochází k nasazení pouze menšího počtu změn, a ne velkého balíku změn. Případné chyby je tak snazší identifikovat a řešit. [20]

3.5 Nástroje pro CI/CD

V této kapitole budou popsány nejrozšířenější CI/CD nástroje. Budou popsány hlavní vlastnosti a výhody jednotlivých nástrojů a jaké poskytují edice.

3.5.1 GitHub Actions

GitHub Actions je CI/CD nástroj vyvíjený *GitHubem*. *GitHub Action*s zpřístupňuje CI/CD procesy a nástroje pro repozitáře hostované v rámci *GitHubu*. *GitHub Actions* je dostupný zdarma pro veřejné repozitáře a pro projekty, které využívají vlastní *GitHub Actions Runner*. Zpoplatnění pro soukromé repozitáře je zahrnuto v rámci plánů pro *GitHub* samotný a není nutné platit žádné plány navíc. Jednotlivé plány se liší v rámci počtu minut, dostupných pro spouštění pipeline. Zajímavostí je, že minuty pro spuštěné pipeline na operačních systémech *Windows* a *MacOS* stojí 2krát, respektive 10krát více než minuty pro *Linux*. To znamená, že 100 minut reálného běhu pipeline na operačním systému *MacOS* se v projektu počítá jako minut 1000. [16][23]

CI/CD pipeliny jsou definovány v konfiguračních souborech YAML. Tyto soubory jsou uložené v repozitáři ve složce *.github/workflows.* Jeden repozitář může obsahovat více konfiguračních souborů umožňující definovat více pipeline. Pipeliny se pro *GitHub* nazývají workflow a je možné je vizualizovat v rámci webového rozhraní. Konfigurační soubor obsahuje možnost nastavení názvu pro workflow, spouště pro spuštění dané workflow a jednotlivé kroky včetně definice na kterém agentovi budou vykonány. Mezi možné spouště patří například detekce nových změn nebo žádosti o začlenění nových změn do hlavní větve repozitáře. Dá se také nastavit sledování štítků pro dané změny. V rámci kroků workflow lze nastavit akce, které se mají provést. Příkladem akce může být příkaz pro spuštění testů skrze nástroj *Maven*. Zároveň se dá nastavit jaká verze *Maven* se má použít. [23]

Workflow a její kroky musí být spuštěny na nějakém serveru. V kontextu CI/CD nástrojů se používají termíny jako je agent nebo runner. V případě *GitHub Actions* se setkáváme s pojmem runner. Runner je server, který je dostupný jako služba v rámci *GitHub Actions* a zpoplatnění jeho využívání spadá do plánů popsaných výše. Hostování vlastního runneru je zdarma a tím pádem není nutné řešit maximální počet minut strávených spuštěnými workflow. Existuje ovšem limitace v podobě maximálního možného množství přenesených dat, a maximální počet požadavků mezi runnerem a repozitářem hostovaným na *GitHubu*. [23]

3.5.2 GitLab CI/CD

Nástroj *GitLab* CI/CD je přístupný v rámci služeb *GitLabu*. Je tedy možné spouštět CI/CD pipeline pro repozitáře hostované přímo v *GitLabu*. Základní funkcionality CI/CD jsou v rámci *GitLabu* zpřístupněné zdarma a pokročilejší funkcionality jsou dostupné společně s placenými edicemi *GitLabu* samotného a není zde nutno nic dokupovat. Mezi placené funkcionality mimo jiné patří možnost spouštění pipeline pro vzdálené repozitáře nebo rozšíření panely pro zobrazování nastavených pipeline. Součástí placených edic je také větší množství dostupných minut pro běhy pipeline. V případě překročení tohoto omezení je možné si jednorázově dokupovat další minuty. [24]

Definice pipeline se provádí konfiguračními soubory ve formátu YAML. Tento soubor se většinou jmenuje *.gitlab-ci.yml* a běžně se nachází v kořenové složce repozitáře. V rámci nastavení repozitáře je možné nastavit jméno pro tento soubor včetně cesty k němu. Je však možné odkazovat i konfigurace, které se nacházejí v jiném *GitLab* repozitáři. Navíc je možné rozdělit konfigurace do více souborů a zároveň odkazovat jiné konfigurace z konfigurace současné. Prostřednictvím webového rozhraní *GitLabu* je možné editovat pipeliny, ověřit správnost jejich zápisu a následně je i vizualizovat. Konfigurace samotná dovoluje definici jednotlivých kroků, které se mají provést, na kterém *GitLab* runneru se mají provést a možnosti nastavení v jaké etapě sestavění se mají kroky spustit. Mezi další nastavení patří chování pipeline v případě, že se vyskytne chyba, tedy zda se má přerušit, nebo pokračovat. Pomocí konfigurace pipeline je také možné publikovat *GitLab Pages*, které jsou popsány v předešlých kapitolách. [24]

GitLab pipeliny se spouští pomocí *GitLab Runnerů*. Je možné využít *GitLab Runnery*, které jsou poskytované přímo skrz cloud verzi *GitLabu*, u které je ale třeba řídit se počtem minut, které můžeme využít, nebo je možné si nainstalovat vlastní *GitLab Runner*. *GitLab Runner* je podporovaný pro operační systémy *Windows*, *MacOS* i distribuce *Linuxu*. Možná je také instalace v rámci *Docker* kontejnerů. [24]

3.5.3 Jenkins

Jenkins je samostatný CI/CD nástroj. Narozdíl od nástrojů v předešlých kapitolách *Jenkins* neposkytuje hostování repozitářů. Další odlišností je, že se jedná o open source projekt, který je vyvíjen komunitou dobrovolných přispěvatelů. Ve výchozí instalaci nemá ani moc funkcionalit a spousta z nich je nutné přidat v rámci pluginů. Díky pluginům je *Jenkins* více modulární, flexibilní a neobsahuje zbytečné možnosti, které uživatelé nikdy nevyužijí. Pluginy

také dovolují rozšíření o funkcionality, které mohou být vytvořené přímo pro potřeby projektu. Výrazným rozdílem od předešlých nástrojů je také, že *Jenkins* je zcela bezplatný a neexistuje žádná placené edice. Nevýhodou může ale být podpora pouze lokálních instalací, jelikož *Jenkins* není poskytován jako webová služba. [25]

Před konfigurací jednotlivých pipeline je nutné pro *Jenkins* vytvořit projekt. V kontextu *Jenkinsu* se pro projekt také využívá označení *job*. Projekty slouží pro konfiguraci celku v rámci, kterého je nastavený zdrojový repozitář, pravidla pro spouštění pipeline a konfigurace pipeline pro daný projekt. Tato konfigurace se dá udělat prostřednictvím webového rozhraní, nebo skrz REST API. [25]

Konfigurace pipeline probíhá skrz konfigurační soubory, které se nazývají *Jenkinsfile*. *Jenkinsfile* je psaný v jazyce *Groovy* a existují dvě formy zápisu. Zápis ve formě skriptované pipeline je starší z této dvojice. Zápis v deklarativní syntax je novější styl zápisu a oproti skriptované formě je lépe čitelný a nabízí více funkcí. Jako *Jenkinsfile* se bude v tomto textu označovat konfigurace s deklarativní syntaxí. *Jenkinsfile* se musí nacházet buď ve zdrojovém repozitáři a v rámci konfigurace projektu je k němu uvedena cesta, nebo je možné vydefinovat pipeline přímo v konfigurace projektu. Samotný *Jenkinsfile* dovoluje definici etap pipeline, agentů pro spouštění pipeline a kroků. Dále lze definovat nástroje, což mohou být různé verze *Javy*, *npm* a dalších spustitelných programů, které lze využívat a spouštět v rámci definovaných kroků. Také je možné definovat a spouštět odlišné příkazy pro systémy *Windows* a *Linux*. [25]

Spouštění jobů a jejich pipeline je možné přímo na *Jenkins* serveru, což ale není doporučovaný přístup, jelikož sebou nese řadu rizik. V případě závadných pipeline by mohlo dojít k poškození souborového systému nebo ztrátě dat. Proto je doporučené využít *Jenkins* agenty. Ty dovolují spouštět pipeline na separátním serveru a v případě, že by závadná pipeline provedla změny například na souborovém systému nepřijdeme o důležitá data. Agenti jsou běžné servery bez speciální *Jenkins* aplikace. *Jenkins* server se na tento server připojí a vykonává na něm kroky pipeline. Pro toto se většinou využívá SSH. Pokud nechceme využívat celé servery jako agenty, je možné využít *Docker* kontejnerů. [25]

3.5.4 Porovnání

Všechny z těchto nástrojů poskytují podobné funkcionality, avšak se liší v menších rozdílech, které jsou klíčové během rozhodování, který z nástrojů je v hodné využít. Jednou z nejdůležitějších vlastností je integrace a podpora verzovacích systémů. *GitHub Actions* společně s *GitLab CI/CD* nabízí spouštění pipeline pro repozitáře, které jsou hostované na *GitHubu*, respektive *GitLabu*. Tyto nástroje je vhodné zvolit pro projekty, které mají své repozitáře již vytvořené na jedné ze jmenovaných platforem, nebo je plánují využít. Pro *Jenkins* pipeline je naopak nutné definovat zdrojový repozitář a nevzniká tak pevná vazba na jednu z platforem. *Jenkins* navíc podporuje i jiné verzovací systémy, než je Git prostřednictvím dodatečných pluginů. Pluginy a rozšíření také hrají roli při rozhodování, který z nástrojů využít. V tomto směru exceluje *Jenkins*, který disponuje širokou nabídkou pluginů, které jsou komunitně vyvíjené. *GitHub Actions* mají podporu rozšíření pomocí *GitHub Marketplace*, který taktéž nabízí rozšíření vyvíjené komunitou. Tato vlastnost dovoluje využití existujících řešení, což napomáhá časové efektivitě při vytváření pipeline. *GitLab CI/CD* v tomto směru trochu zaostává, ačkoliv podporuje vytváření vlastních skriptů a integraci s externími nástroji. Dalším z důležitých prvků je podpora vlastní instalace CI/CD nástroje. Požadavkem na CI/CD nástroj může být bezpečnost dat a využití cloudových služeb nemusí být možné. *Jenkins* a *GitLab CI/CD* nabízejí možnost hostování instancí CI/CD nástrojů na vlastních server zdarma minimálně pro nenáročné uživatele. Naopak *GitHub Actions* na vlastních serverech je možné využívat pouze s placenou edicí. [26]

Tabulka *2* – Porovnání CI/CD nástrojů

Funkce	GitHub Actions	GitLab CI	Jenkins
Zabudovaný VCS	✓	✓	✗
Konfigurace pipeline	✓ YAML	✓ YAML	✓ Groovy/Jenkinsfile DSL
Validace pipeline	✓ Prostřednictvím pluginů	✓	✓ Prostřednictvím pluginů

Podpora pluginů	✓ GitHub Marketplace	✗ Integrace s externími nástroji	✓ Pluginy
Online-hosted	✓	✓	✓ Služby 3. stran
Self-hosted instance	✓ Placená verze	✓	✓

3.6 Využití ve školním prostředí

Již bylo popsáno téma využití verzovacích systémů ve školním prostředí. Verzovací systémy umožnily snazší studentských prací pomocí zobrazení provedených změn. Zároveň bylo usnadněn přístup k zadání a řešení úkolů. Učitel však stále musí manuálně provést stažení studentského repozitáře a provést kontrolu řešení studenta v podobě sestavění projektu a spuštění jednotkových testů. Pro další usnadnění kontroly studentských řešení lze provést automatizaci tohoto procesu za využití CI/CD nástrojů.

CI nástroje umožňují vytvořit pipeline, která bude automaticky kontrolovat studentské repozitáře a evidovat výsledky provedených testů. V řešení bez využití pipeline si musel vyučující hlídat studentské repozitář manuálně. CI pipeliny je však možné nastavit, aby byly spuštěny při detekci změn ve studentském repozitáři. To nám umožňuje okamžitou reakci a vyhodnocení studentské práce bez nutnosti zásahu učitele. Současně musel vyučující provést sběr studentských repozitářů po stanoveném termínu odevzdání, aby se zabránilo možnosti, že studenti budou provádět změny po jeho uplynutí. I tento proces lze automatizovat pomocí plánování CI pipeline. Lze vytvořit pipeline, která se spustí v čase termínu odevzdání a po jejím dokončení může být automaticky spuštěna další pipeline, která provede jejich vyhodnocení. Další možností je také možné provést tento proces v jednom kroku a vyhodnocovat studentská řešení ihned. V případě, že kontroly prací trvají déle, poslední studentská práce však může být vyhodnocena několik desítek minut po termínu odevzdání, což by dovolilo studentům odevzdávat práce s menším zpožděním.

Po spuštění pipeline je provedena kontrola studentského řešení. Tento krok se ve většině případů skládá ze sestavění projektu a spuštění jednotkových testů. Ale již v tomto kroku je nám umožněno komplexnější řešení. Někdy nemusí být chtěné, aby student měl k dispozici jednotkové testy. To může být z důvodu, aby si ověřil funkcionalitu kódu vlastními testy, nebo mohou jednotkové testy obsahovat nápovědy pro řešení úlohy. V rámci pipeline je možné vydefinovat krok, který nakopíruje testy ze soukromého repozitáře a provede jejich spuštění. Další výhodou kontroly prováděné v rámci pipeline je možnost automatické ověření detailů implementace. V rámci zadání může být vyžadováno využití specifické syntaxe, klíčových slov jazyka, nebo naopak jejich nevyužití. Pipeline může obsahovat krok, který provede analýzu odevzdaných souborů a může ověřit, zda kód splňuje tyto požadavky. Dalším krokem může být i analýza pokrytí jednotkovými testy, zda student dostatečně pokryl všechny třídy.

Výsledky je pak možné publikovat prostřednictvím sdíleného informačního kanálu. Tím může být webová stránka, repozitář obsahující soubory s výsledky jednotlivých testů, nebo prostřednictvím emailu. Vygenerované výsledky mohou být v jakémkoliv formátu a mohou obsahovat informace o splnění či nesplnění jednotlivých částí zadání. Výsledná zpráva může obsahovat například informace o pokrytí testy, kde bylo provedeno dostatečné pokrytí a kde nikoliv.

4 Praktická část

4.1 Analýza problému

Cílem této práce je navrhnout a implementovat nástroj pro *Continuous Integration* (CI), který bude určen pro potřeby školních úloh a jejich automatizovaného vyhodnocování. Tento nástroj, se bude skládat z nově vytvořené řídící aplikace a existujícího CI nástroje. Řešení umožní konfiguraci pomocí textových souborů (YAML), ve kterých budou specifikovány informace o předmětech, úkolech a pravidlech pro vyhledávání repozitářů s vypracovanými úlohami.

Navrhovaný systém bude mít schopnost automaticky stahovat a vyhodnocovat studentské úkoly z definovaných repozitářů. Bude provádět kompilaci, testování a obecné vyhodnocení úloh na základě specifikovaných kroků. Bude také schopen systematicky zaznamenávat výsledky a identifikovat případné chyby. Výsledky budou prezentovány v podobě přehledného reportu ve formě webové stránky, která umožní zobrazení rozdílů mezi provedenými změnami studenty a zadáním úlohy.

Tento nástroj by měl usnadnit proces hodnocení a zpětné vazby na studentské práce, a to prostřednictvím automatizace procesů, sledování vývoje a poskytování přehledných výsledků.

4.2 Požadavky

- *R01 – Podpora textových formátů pro konfiguraci*: Systém by měl podporovat textové formáty pro konfiguraci, pro specifikaci předmětů, úkolů a pravidel pro vyhledávání repozitářů.
- *R02 – Automatická kontrola úloh*: Systém by měl umožnit automatické stahování úloh ze specifikovaných repozitářů studentů.
- *R03 – Konfigurovatelné kroky vyhodnocení*: Systém by měl mít umožnit definovat kroky pro kompilaci, testování a obecné vyhodnocení úloh, které budou provedeny během testování.
- *R04 – Zaznamenávání výsledků a chyb*: Systém by měl systematicky zaznamenávat výsledky vyhodnocení úloh a identifikovat případné chyby, které se vyskytly během procesu.

- *R05 – Generování přehledných reportů*: Po vyhodnocení úloh by systém měl generovat přehledné reporty ve formě webových stránek, které budou obsahovat výsledky úloh a možnost zobrazení rozdílů mezi provedenými změnami studenty a zadáním úlohy.
- *R06 – Aplikace běží na pozadí systému*: Aplikace by měla běžet na pozadí operačního systému jako služba.

4.3 Řešení požadavků

4.3.1 Podpora textových formátů pro konfiguraci

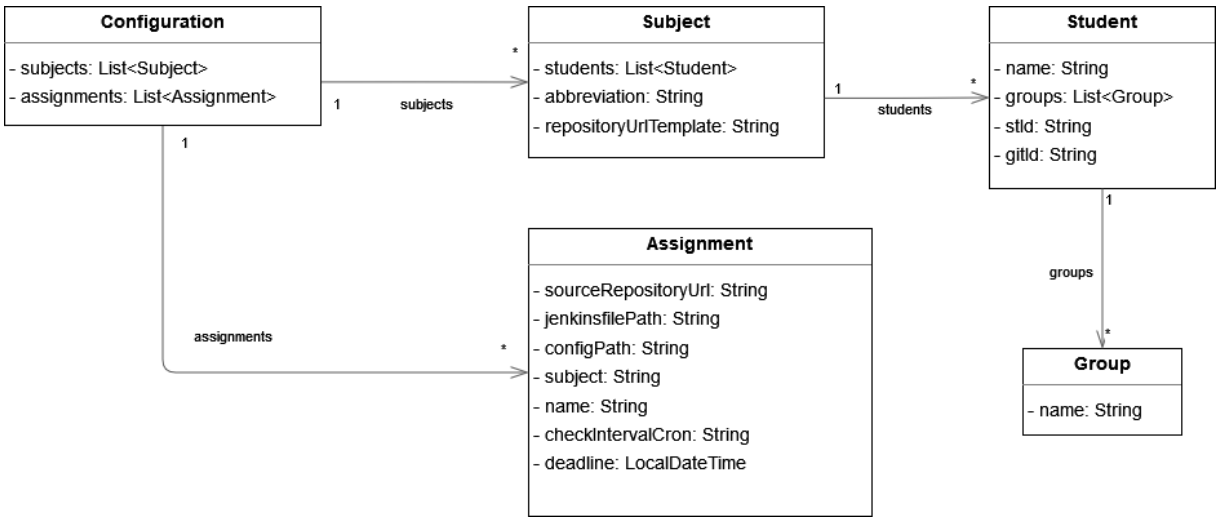
Pro konfiguraci byl zvolen formát YAML. YAML je lidsky čitelný jazyk pro serializaci dat, který se často používá pro zápis konfiguračních souborů. [27] Pro lepší čitelnost je konfigurace prováděna v rámci několika souborů. Existuje jeden hlavní konfigurační soubor, který obsahuje odkazy na ostatní konfigurační soubory. Odkazované konfigurační soubory jsou využité pro konfiguraci jednotlivých předmětů, studentů a zadání. Tyto konfigurační soubory jsou uložené na vzdáleném *Git* repozitáři. To dovoluje jednoduchou správu a verzování jednotlivých konfigurací. Tento *Git* repozitář je sledován v pravidelném intervalu, avšak je možné manuální vynucení kontroly. Po detekci změn v konfiguračním repozitáři si aplikace načte novou konfiguraci a začne ji využívat.

Jak již bylo naznačeno v rámci konfiguračních souborů probíhá definice jednotlivých předmětů, seznam zadání pro daný předmět a seznam studentů. Předmět je identifikovaný zkratkou jeho názvu a dále obsahuje nastavení šablony pro vytvoření URL studentského repozitáře a seznam všech studentů. Šablona URL pro studentské repozitáře je vytvořena ve formátu pro získání vzdáleného repozitáře, avšak části pro ID vlastníka repozitáře a název repozitáře jsou nahrazeny zástupnými řetězci (placeholder), které se následně doplňuje konkrétními hodnotami. Příklad šablony pro repozitáře hostované na platformě *GitHub*: *https://github.com/%STUDENT-GIT-ID%/%ASSIGNMENT-NAME%.git*.

V rámci konfigurace studenta je pro jeho identifikaci využito studentské ID. Dále definuje jméno studenta, název účtu, který student používá pro odevzdávání řešení úloh a výpis skupin, ve kterých je student přiřazen. Skupiny jsou využívány v rámci jednoho předmětu pro snadné filtrování studentů, například podle cvičení, na které dochází.

Konfigurace dále obsahuje definici všech zadání pro daný předmět. Každé zadání je identifikováno pomocí zkratky předmětu, pro který je určené a názvem úkolu. Dále je možnost konfigurace termínu odevzdání řešení studenty, jak často mají být studentské repozitáře kontrolovány, zdrojový repozitář zadání a cesty ke konfiguračním souborům pipeline pro otestování studentských prací.

Níže je uvedený diagram struktury konfiguračních souborů pro definici předmětů, zadání a studentů.



Obrázek *2* – Diagram struktury YAML konfigurace

4.3.2 Automatická kontrola úloh

Systém tento požadavek řeší periodickým sledováním všech studentských repozitářů. Interval kontrol je možné zadat v rámci konfigurace a případně je využitá výchozí hodnota pro kontrolu každých šest hodin. Repozitáře jsou pravidelně testovány jen v případě, že předchozí testování neproběhlo v pořádku. V případě, že testování proběhlo pro daný repozitář v pořádku a bez chyb, aplikace si repozitář poznamená jako úspěšně vyhodnocený a již neprovádí jeho kontroly. Pro další optimalizaci je před spuštěním samotného testování repozitáře zkontrolován SHA-1 hash posledního commitu a v případě, že od poslední kontroly nedošlo k změně, testování není spuštěno. V konfiguraci zadání je možné pro studenty uvést termín odevzdání řešení. Chvíli po termínu odevzdání je proveden test všech repozitářů, které neprošly úspěšnými testy a je jim nastaven finální stav. Finální stav označuje repozitář jako úspěšný nebo neúspěšný. Repozitáře jsou před spuštěním testovací pipeline vloženy do fronty na CI systému *Jenkins*. Nedochází tak k vytěžování systému zahlcením prováděním desítek testování studentský repozitářů. Konfigurací více *Jenkins* agentů je možné docílit kontroly více repozitářů současně.

4.3.3 Konfigurovatelné kroky vyhodnocení

Konfigurace vyhodnocování studentský repozitářů je řešena definicí pipeline v CI systému *Jenkins*. Toto řešení dovoluje flexibilní přístup k podporovaným technologiím, ve kterých mohou být řešena zadání. V rámci konfigurace *Jenkins* agentů je možné uvést více verzí jazyků (*C#*, *Java*, ...) a pomocných nástrojů pro spouštění studentských prací (*Maven*, *Gradle*, ...). Konfigurace jednotlivých kroků vyhodnocení úloh je také usnadněna a je možné definovat kroky pro sestavění projektu, spuštění jednotkových testů, nebo kroků pro analýzu projektu jako takové. Konfigurace pipeline se nacházejí na vzdáleném repozitáři a je možné jednotlivé konfigurace používat mezi více zadáními.

4.3.4 Zaznamenávání výsledků a chyb

Aplikace si po otestování studentského řešení ukládá výsledky posledního běhu kontroly řešení. Uchovány jsou pipeline logy ze systému *Jenkins*. Tyto logy obsahují podrobné informace o průběhu jednotlivých kroků definovaných v pipeline, včetně výstupů testů, chyb a varování, které mohou nastat během testování řešení studenta. Kromě toho je také uchováván výsledný stav běhu pipeline. Tento stav poskytuje informaci o tom, zda celý proces vyhodnocení proběhl úspěšně nebo došlo k nějakým chybám či selháním.

Dále jsou zaznamenávány rozdílů mezi repozitářem zadání úkolu a studentského řešení. To vyučujícím dovoluje snadnou kontrolu provedených změn.

4.3.5 Generování přehledných reportů

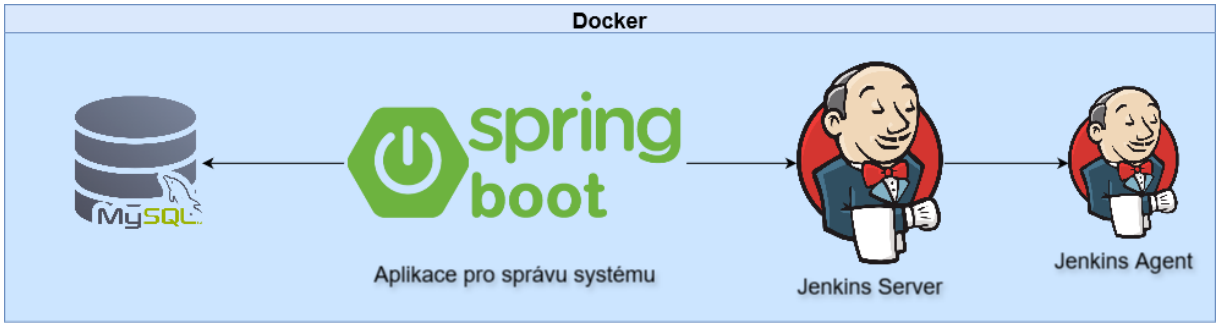
V rámci aplikace je dostupné grafické rozhraní v podobě webové stránky. Webové rozhraní obsahuje obrazovky, které poskytují informace o všech sledovaných předmětech, studentech a zadáních. Jsou dostupné zobrazení, které obsahuje přehled stavu všech řešení studentů pro jedno zadání. Dále je dostupná matice, která má kompaktnější zobrazení pro všechna zadání a studenty v rámci jednoho předmětu. Následně je možné si zobrazit detailní výpis změn všech změn, které student provedl oproti zadaní společně s logem z posledního uloženého běhu pipeliny ze systému *Jenkins*. V rámci tohoto logu jsou dostupné informace o případných chybách, které nastaly v průběhu vyhodnocování řešení.

4.3.6 Aplikace běží na pozadí systému

Všechny komponenty dodaného systému jsou spuštěny jako služby na pozadí, což zajišťuje jejich neustálý běh a dostupnost pro uživatele. Specificky aplikace pro správu a *Jenkins* poskytují uživatelům přístup ke grafickému rozhraní prostřednictvím webových stránek, které umožňují snadnou interakci s funkcionalitami systému.

4.4 Návrh a struktura systému

Řešení systému pro automatickou kontrolu studentských prací se skládá ze 4 hlavních komponent. Celý systém je řízen centrální aplikací, která poskytuje uživatelské rozhraní přes webovou stránku. Tato aplikace ukládá svá data v relační databázi *MySQL* a spouští kontrolu studentských prací prostřednictvím CI nástroje *Jenkins*. Jenkins spouští jednotlivé pipeline na *Jenkins Agentovi*. Celý tento systém běží v rámci *Docker* kontejnerů. V této kapitole budou popsány jednotlivé komponenty, včetně souboru *Docker compose*.



Obrázek 3 – Struktura komponent systému

4.4.1 Docker Compose

Docker je populární platforma pro kontejnerizaci, která vývojářům umožňuje sdružovat aplikace a jejich závislosti do izolovaných kontejnerů. Tyto kontejnery jsou lehké, a přenosné což umožňuje konzistentní nasazení v různých prostředích. *Docker* poskytuje standardizovaný způsob vytváření a správy těchto kontejnerů, což usnadňuje bezproblémové vytváření a spouštění aplikací v různých prostředích. Díky nástroji *Docker* mohou vývojáři zefektivnit své vývojové procesy, zlepšit spolupráci mezi týmy a dosáhnout větší škálovatelnosti a efektivity při nasazování aplikací. Hlavní výhodou *Docker* kontejnerů oproti virtuálním strojů je, že *Docker* kontejnery sdílí jádro operačního systému s hostitelským počítačem. Oproti tomu virtuální stroje jsou dodávány včetně operačních systémů. [28]

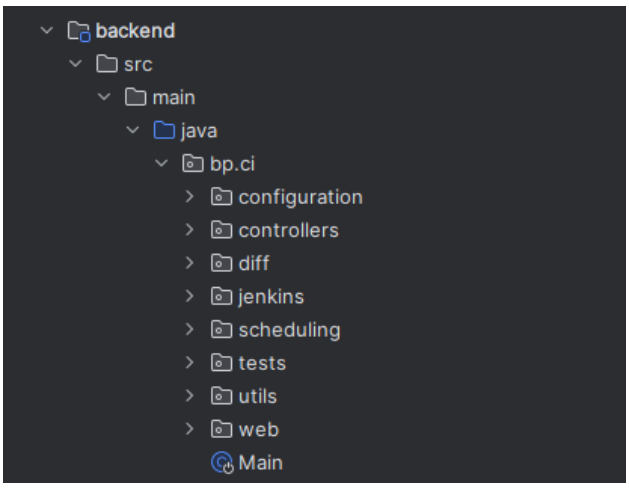
Docker compose je nástroj, který zjednodušuje proces definování a správy systémů, které jsou založeny a běží prostřednictvím *Docker* kontejnerů. Umožňuje vývojářům pomocí souboru *YAML* definovat jednotlivé kontejnery označované jako služby (*services*), sítě, kterými jsou kontejnery propojeny a permanentní úložiště pro kontejnery označované jako svazky (*volume*) potřebné pro jejich aplikaci. Pomocí jediného příkazu lze vytvořit, spustit a spravovat všechny kontejnery definované v konfiguraci. [29]

Hlavním motivací pro využití platformy Docker pro řešení tohoto systémů byla jednoduchá správa a přenosnost jednotlivých komponent systému. Pro distribuci systému stačí konfigurační soubor pro *Docker compose* a image aplikace pro správu systému. Image pro ostatní komponenty systému jsou dostupné prostřednictvím *Docker Hubu* a není je tedy nutné dodávat samostatně při distribuci systému. Nedílnou výhodou je také snadné rozšíření systému o další kontejnery, zejména pro *Jenkins Agenty*.

4.4.2 Aplikace pro správu systému

Pro správu a řízení kontroly studentských repozitářů pro zadané úkoly byla v rámci systému zvolena samostatná komponenta v podobě aplikace využívající *Spring Boot* framework. Nejdůležitějším úkolem této aplikace je správa pipeline v rámci CI nástroje *Jenkins*, sledování studentských repozitářů a hlídání změn, spouštění pipeline v případě detekce změn a zobrazení výsledků pro provedené testování prostřednictvím webové rozhraní včetně změn, které byly studenti v řešení provedeny.

Aplikace je strukturovaná do jednotlivých modulů a balíčků, které se zaměřují na jednotlivé funkcionality a požadavky aplikace. Na obrázku níže je zobrazena struktura balíčků v *backend* modulu. Pro balíčky obsahující nejdůležitější části aplikace bude následovat jejich popis.



Obrázek *4* – Struktura balíčků v modulu backend

Prvním balíčkem je balíček *configuration* pro konfiguraci předmětů, zadání a studentů. Tento balíček se stará o načtení konfigurace ze vzdáleného repozitáře, jejího načtení do paměti aplikace a vytvoření záznamů v databázi pro jednotlivé předměty a studenty. Aplikace využívá databázi pro ukládání dat o výsledcích testování studentských prací. Aplikace tak pracuje částečně s konfigurací, která je uložená v rámci paměti aplikace a výsledcích a datech z databáze, které doplňují tuto konfiguraci.

Balíček *tests* obsahuje službu pro správu a spouštění pipeline v nástroji *Jenkins*. Třída této služby obsahuje metody, které provedou inicializaci pipeline pro všechny studenty, včetně záznamů pro evidenci jejich výsledků v databázi. Dále jsou zde přístupné metody pro spouštění jednotlivých pipeline a zaznamenávání jejich výsledků. Aplikace nespouští kontroly nad repozitáři, které neobsahují žádné změny oproti předchozí kontrole, nebo již prošly úspěšnou kontrolou.

Podstatným balíčkem pro fungování celého systému je balíček *scheduling* obsahující službu pro plánování kontrol studentských prací. Služba v tomto balíčku provádí automatizované plánování kontrol na základě *cron* výrazů. Plánové jsou průběžné kontroly a finální kontrola všech repozitářů krátce po termínu odevzdání úkolu.

Dále aplikace obsahuje balíček *diff* pro vytváření rozdílů mezi zadáním a studentskými pracemi. Tento balíček se stará o vytváření rozdílů mezi všemi soubory zadání a studentů. Služba pro vytváření rozdílů si stahuje repozitáře na lokální úložiště a nad staženými soubory vytváří jednotlivé rozdíly. Rozdíly jsou vytvářeny pouze pokud došlo ke změně SHA-1 hash studentského repozitáře a případné rozdíly jsou zaznamenány v databázi pro každé řešení. Není zde ovšem podpora evidence průběžných změn v repozitáři, ale jen změn posledních.

V neposlední řadě aplikace obsahuje *frontend* modul pro webové rozhraní aplikace. V tomto modulu jsou vybudované webové stránky v technologii *ReactJS* s knihovnou komponent *MUI*. Toto rozhraní především umožňuje graficky vizualizovat stav kontrol pro jednotlivé předměty, zadání v rámci předmětu a poté i výsledky kontrol pro každého studenta. Dále existuje obrazovka, která dovoluje zobrazení aktuálního nastavení aplikace včetně načtené konfigurace ze vzdáleného repozitáře.

4.4.2.1 Konfigurace aplikace

Protože aplikace využívá *Spring Boot* framework, můžeme využívat konfigurační parametry poskytované tímto frameworkem. Tyto parametry nejsou v rámci této práce pospané.

Následovat bude přehled a popis jednotlivých konfiguračních parametrů, které jsou specifické pro fungování aplikace.

Pro vytvoření spojení do databáze se používají běžné parametry pro aplikace využívající *Spring* framework. Tyto parametry jsou označeny prefixem *spring.datasource* a jejich výčet s popisem je v následující tabulce. Prefix u parametrů v tabulce není uveden.

Tabulka *3* – Parametry konfigurace databázového spojení

Parametr	Popis
url	Parametr pro definici JDBC připojovacího řetězce do databáze.
username	Pomocí uživatelského jména uvedeného v tomto parametru se aplikace připojí do databáze.
password	Pomocí hesla uvedeného v tomto parametru se aplikace připojí do databáze.

Konfigurační parametry pro nastavení vzdáleného repozitáře s uloženými konfiguracemi předmětů, zadání a studentů jsou označovány s prefixem *bp.ci.management-repository*. Následuje tabulka všech parametrů, kde je tento prefix vynechán, ale ve skutečné konfiguraci musí být přítomen.

Tabulka *4* – Parametry konfiguračního repozitáře

Parametr	Popis
repositoryUrl	Parametr slouží pro nastavení URL k serveru, kde se nacházejí konfigurace.
mainConfigurationPath	Parametr slouží pro specifikaci, kde na tomto serveru se nachází soubor obsahující cesty k ostatním konfiguračním souborům.
authorizationToken	Parametr slouží pro autentizace přístupu ke vzdálenému repozitáři a není povinný v případě, že vzdálený repozitář je veřejný.

checkIntervalCron	Parametr je ve formátu zápisu *cron* výrazu a slouží pro nastavení frekvence kontrol konfiguračního repozitáře. Výchozí hodnotnou tohoto parametru je kontrola repozitáře každou 6 hodinu.

Konfigurační parametry, které aplikace využívá pro připojení na *Jenkins* server začínají prefixem *bp.ci.jenkins*. Následuje tabulka s popisem těchto parametrů bez uvedeného prefixu.

Tabulka *5* – Parametry připojení na Jenkins server

Parametr	Popis
serverUrl	Tento parametr slouží pro definici URL webového rozhraní Jenkinsu a používá se pro komunikaci skrz REST API.
username	Pomocí uživatelského jména uvedeného v tomto parametru se aplikace přihlašuje.
token	Token slouží pro autentizaci daného uživatele. Token musí být nastavený s dostatečnými právy.

Pro konfiguraci vytváření rozdílů mezi řešením studentů a zadáním existuje jeden parametr, který je označen s prefixem *bp.ci.assignment-difference*. Následuje jeho popis opět bez uvedeného prefixu.

Tabulka *6* – Parametr pro rozdíly repozitářů studentů

Parametr	Popis
sourceDirectory	Parametr slouží pro definici umístění, kam si aplikace stahuje repozitáře se zadáním úkolů. Doporučené je umístění v dočasné složce operačního systému.

4.4.2.2 Konfigurace docker kontejneru

Kontejner pro řídící aplikaci používá vlastní image, který není dostupný z Docker Hub repozitáře. Image je možné vybudovat spuštěním Maven pluginu ve zdrojových kódech aplikace. Kontejner je spuštěný se jménem *bp-cp-management*. Pomocí svazků je ke kontejneru připojená složka obsahující konfiguraci pro aplikaci samotnou a její aplikační logy. K tomu se

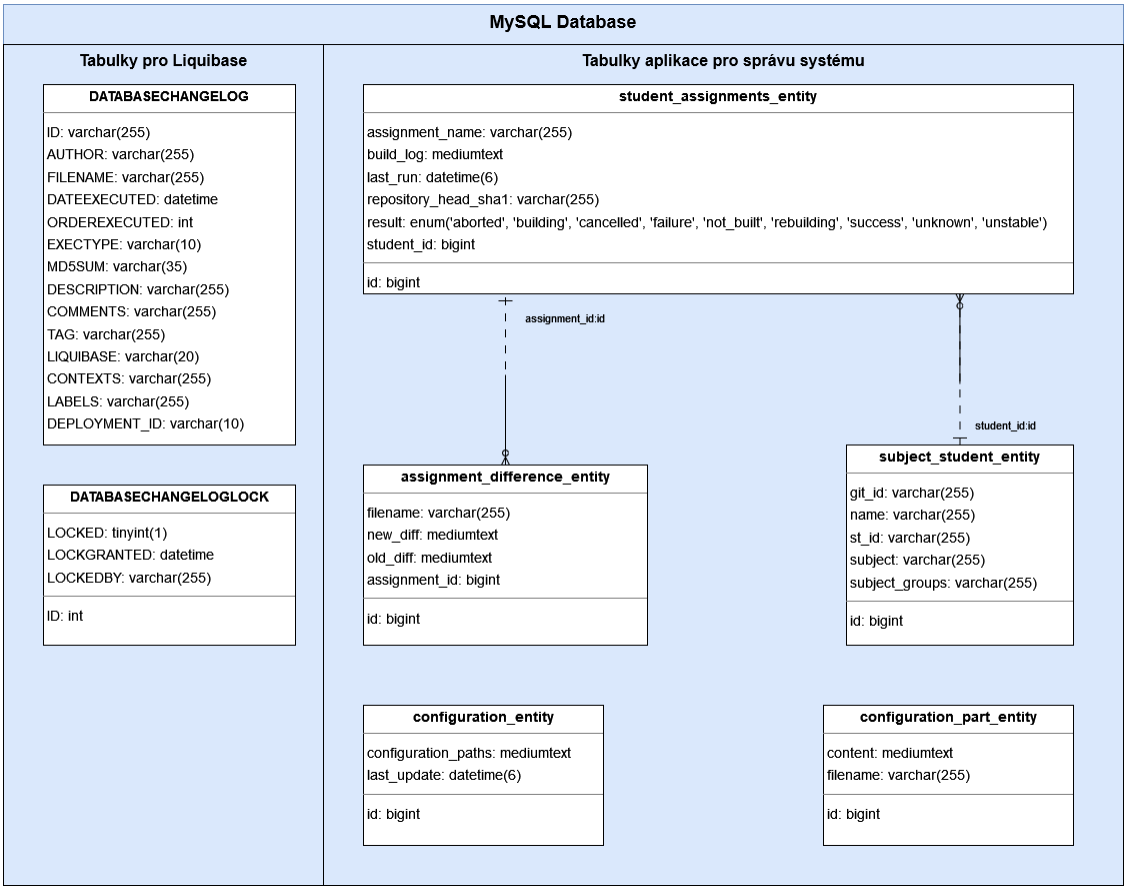
váže proměnná prostředí, která obsahuje cestu ke konfiguračním souborům uvnitř kontejneru. Konfigurace je rozložena do 2 souborů, kde jeden soubor obsahuje obecné parametry a ve druhém souboru se nachází hesla a autorizační tokeny. Vystavěný je také port pro přístup na webové rozhraní aplikace. Níže je uvedena konfigurace kontejneru.

```
management:
  container_name: 'bp-ci-management'
  image: 'pinkas/bp-ci-management:latest'
  volumes:
    - ./management/config:/opt/management/config
    - ./management/logs:/opt/logs/
  environment:
    - 'spring.config.location=/opt/management/config/application
      .properties,/opt/management/config/secret.properties'
  ports:
    - '8888:8080'
```

4.4.3 Databáze

MySQL je jednou z nejrozšířenějších relační databází na světě. Pro ukládání dat byla zvolena z důvodu její jednoduchosti a snadnému integraci s celým systémem. Databáze je využívána pouze aplikací pro správu systému. V databázi jsou ukládá data o studentech, předmětech a zadání. Model databáze je lépe vidět na obrázku níže. Hlavní entitou je *subject_student_entity* ve které jsou evidované informace o studentovi, předmětu v rámci, kterého je evidován, skupiny, do které je přiřazen a jeho identifikátorem pro jeho *Git* repozitáře. Na tuto entitu navazuje entita *student_assignment_entity* která obsahuje data o jednotlivých zadání pro daného studenta. Podstatnými položkami pro evidenci zadání je jeho název, zda proběhla úspěšná kontrola studentova řešení, případně stav poslední kontroly v opačném případě a záznam kontroly z nástroje *Jenkins.* SHA-1 hash studentského repozitáře je také uložena. V návazné entitě *assignment_difference_entity* jsou evidovány rozdíly souborů oproti souborům z repozitáře zadání. Databáze pak obsahuje nezávislé entity pro ukládání konfigurace, a to *configuration_entity* a *configuration_part_entity*. Tyto entity obsahují časovou známku posledního načtení konfigurace, odkud jsou konfigurace načítány a následně názvy a obsah těchto konfigurací. Databáze nakonec obsahuje tabulky *DATABASECHANGELOG* a *DATABASECHANGELOGLOCK*.

Obě tyto tabulky jsou vytvořené pro knihovnu Liquibase. Liquibase slouží jako verzovací systém pro databáze a umožňuje snadnou a bezpečnou aktualizaci modelu databáze. [30]



Obrázek *5* – Databázový model řídící aplikace

4.4.3.1 Konfigurace docker kontejneru

Kontejner pro *MySQL* databázi má nastavený image pro verzi 8.0. Kontejner samotný je pojmenovaný *bp-ci-mysql* a v rámci sítě je dostupný pod jménem *mysql*. To dovoluje připojení aplikace pro správu systému skrz tento alias místo IP adresy kontejneru, nebo hostitelského serveru. Pro možnost připojení do databáze pomocí databázového klienta, byl vystaven port 3306. Dále má kontejner nastavené proměnné prostředí, které slouží pro nastavení přihlašovacích údajů a vytvoření výchozí databáze hlavní aplikace. Níže je uvedená konfigurace pro kontejner *MySQL* databáze. [31]

```
mysql:
  image: 'mysql:8.0'
  container_name: 'bp-ci-mysql'
  hostname: 'mysql'
  environment:
    - 'MYSQL_DATABASE=ci'
    - 'MYSQL_ROOT_PASSWORD=password'
    - 'MYSQL_USER=user'
    - 'MYSQL_PASSWORD=password'
  ports:
    - '3306:3306'
```

4.4.4 Jenkins Server

Hlavními požadavky na systém je automatické testování studentských repozitářů společně s konfigurací jednotlivých kroků. Pro řešení těchto požadavků bylo zvoleno využití CI nástroje, konkrétně nástroje *Jenkins. Jenkins* je popsán v předcházejících kapitolách. Hlavním důvodem pro využití existujícího CI nástroje k testování studentských řešení je jeho schopnost snadné konfigurace prováděných kroků a podpora této konfigurace včetně rozšíření ve výchozím nastavení. Mezi hlavní rozhodovací faktory pro výběr nástroje *Jenkins* patřila široká nabídka pluginů, možnost instalace vlastní instance serveru, včetně podpory instalace prostřednictvím platformy *Docker* a dostupnost všech funkcionalit zdarma, bez potřeby placené edice. Dalším faktorem pro výběr byla podpora aplikačních knihoven, které umožňují komunikaci mezi řídící aplikací a *Jenkins* serverem prostřednictvím REST API.

4.4.4.1 Konfigurace docker kontejneru

Kontejner pro nástroj *Jenkins* využívá image *jenkins* z repozitáře *jenkins* s verzí *lts-alpine-jdk21*. Kontejner je pojmenovaný *bp-ci-jenkins* a v rámci sítě je dostupný pod názvem *jenkins.* Díky tomu se může aplikace pro správu systému připojit použitím tohoto názvu bez znalosti IP adresy kontejneru nebo hostitelského počítače. Kontejner běží v privilegovaném režimu s uživatelem *root*. Vystavený port 8080 umožňuje přístup do webového rozhraní a port 50000 slouží pro komunikaci s *Jenkins Agenty*. Dále je nastavený svazek pro složku obsahující konfiguraci *Jenkins* serveru, který je využitý pro počáteční konfiguraci serveru během instalace celého systému. Dále kontejner využívá soubor *docker.sock* z hostitelského počítače pro správu docker kontejnerů umožňující spouštění nových kontejnerů při testování studentských řešení. Níže je uvedená konfigurace ze souboru docker compose.

```
jenkins:
  image: 'jenkins/jenkins:lts-alpine-jdk21'
  container_name: 'bp-ci-jenkins'
  hostname: 'jenkins'
  privileged: true
  user: 'root'
  ports:
    - '8080:8080'
    - '50000:50000'
  volumes:
    - ./jenkins/jenkins_configuration:/var/jenkins_home
    - /var/run/docker.sock:/var/run/docker.sock
```

4.4.5 Jenkins Agent

*Jenkins a*gent je server, který je používán *Jenkins* serverem pro spouštění a vykonávání pipeline. Spustit pipeline lze i na samotném *Jenkins* serveru, ale tento přístup není obecně doporučený a potenciálně nebezpečný. Hlavním rizikem je spouštění neznámého kódu ze studentského repozitáře, který by mohl způsobit poškození souborového systému nebo jiné důležité součásti systému. Agenti dovolují izolaci od hlavní části systému a v případě poškození jej lze obnovit, jelikož neobsahuje žádné důležitá data. Další výhodou je možnost rozšíření o další agenty, které mohou provádět pipeline paralelně.

4.4.5.1 Konfigurace docker kontejneru

Kontejner pro *Jenkins Agent* používá image *ssh-agent* z repozitáře *jenkins* s verzí *debian-jdk21*. Tento agent má v základu podporu pro Javu 21 a Jenkins server komunikuje s tímto agentem prostřednictvím SSH připojení. Kontejner je opět spuštěný v privilegovaném režimu s uživatelem *root*. Kontejner nese název *bp-ci-jenkins_agent* a v rámci sítě je dostupný jako *jenkins_agent*. Pro SSH komunikaci je vystavěný port 22. Nakonec má kontejner nastavenou proměnnou prostředí, která obsahuje SSH klíč pro komunikaci mezi Jenkins serverem a agentem.

```
jenkins_agent:
  image: 'jenkins/ssh-agent:debian-jdk21'
  privileged: true
  user: 'root'
  container_name: 'bp-ci-jenkins_agent'
  hostname: 'jenkins_agent'
  ports:
    - '2222:22'
  environment:
    - 'JENKINS_AGENT_SSH_PUBKEY=ssh-rsa <SSH klíč>'
```

4.5 Obsluha systému

V této kapitole bude popsán systém pro uživatele a jak jej mohou využít.

4.5.1 Konfigurace

Pro konfiguraci předmětů, zadání a studentů se používají soubory YAML, které jsou dostupné prostřednictvím vzdáleného repozitáře. Konfigurační parametry pro nastavení tohoto repozitáře jsou popsány v předešlé kapitole. Následovat budou krátké ukázky možné podoby těchto souborů.

Hlavní konfigurační soubor obsahuje cesty k částem konfigurací obsahují skutečnou definici konfigurace. Soubory jsou při načtení aplikací sloučeny do jedné konfigurace. Soubory jsou načteny ve stejném pořadí, ve které jsou uvedeny v hlavním konfiguračním souboru. Pro duplicitní záznamy bude použit poslední načtený.

```
configurationPaths:
  - 'master/configuration/assignments/BCSH2.yaml'
  - 'master/configuration/assignments/BCSH3.yaml'
  - 'master/configuration/subjects/BCSH2.yaml'
  - 'master/configuration/subjects/BCSH3.yaml'
  - 'master/configuration/SEM_A.yaml'
```

Zadání jsou definována jako pole v konfiguračním souboru.

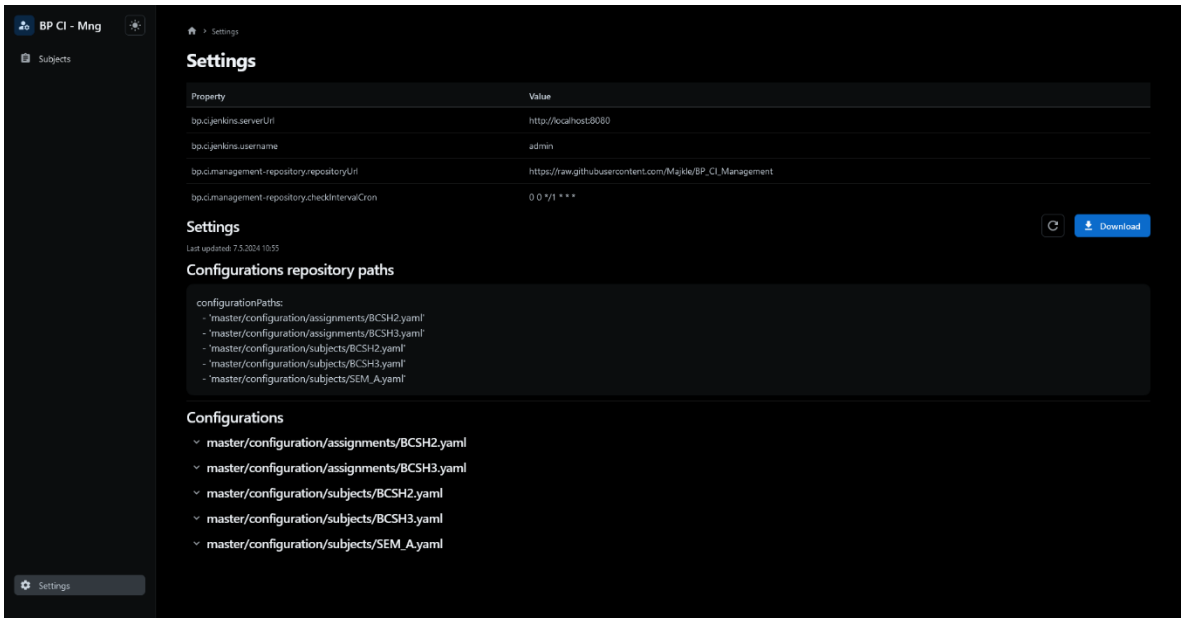
```
assignments:
  - subject: 'BCSH2'
    name: 'Úkol 1'
    deadline: '2024-03-02T00:00Z'
    checkIntervalCron: '* * */10 * * *'
    sourceRepositoryUrl:
        'https://gitserver.com/assignment/BCSH2/exercise-01/'
    configPath: 'master/jenkins/config/defaultConfig.xml'
    jenkinsfilePath: 'master/jenkins/jenkinsfile/Maven.Jenkinsfile'
```

Předměty jsou taktéž definována formou pole, a navíc obsahují definici všech studentů a skupin ve kterých jsou přiřazené.

```
subjects:
  - abbreviation: 'BCSH2'
    repositoryUrlTemplate 'https://github.com/%STUDENT-GIT-ID%
        /%ASSIGNMENT-NAME%.git'
    students:
      - stId: 'st12345'
        name: 'Karel Lekr'
        gitId: 'karellekr'
        groups:
          - 'ut15'
          - 'st16'
      - stId: 'st54321'
        name: 'Daniel Rik'
        gitId: 'danielrik'
        groups: 'st16'
      - stId: 'st13579'
        name: 'Aleš Fernand'
        gitId: 'alesfernand'
        groups: 'st16'
```

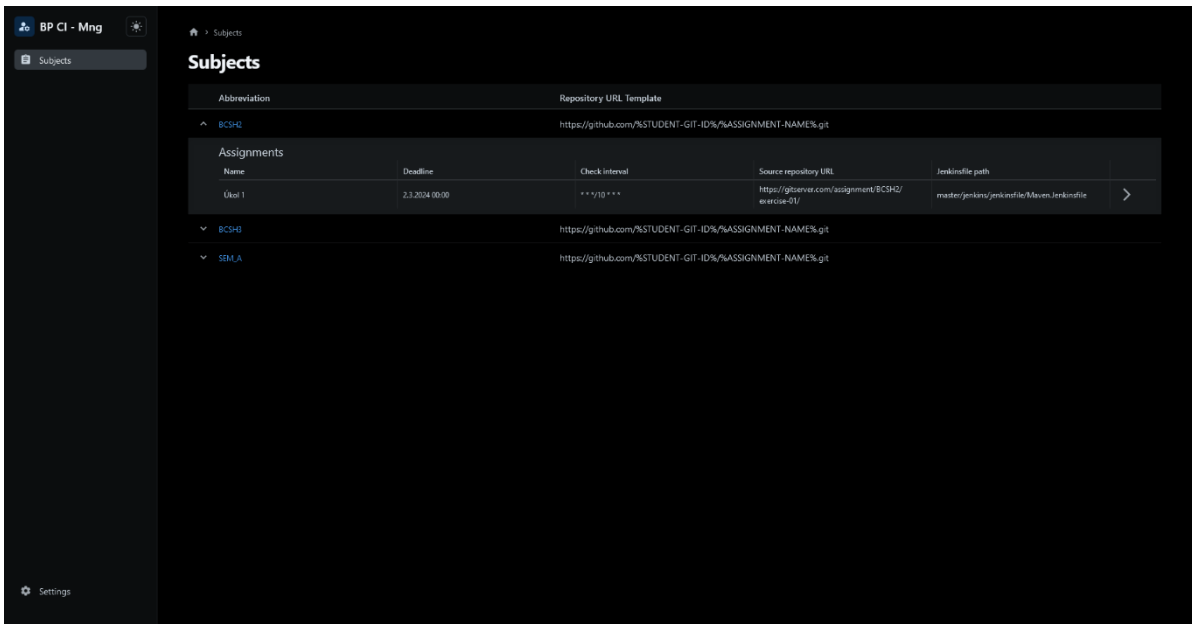
4.5.2 Grafické rozhraní

Po načtení konfigurace spuštění aplikace je možné otevřít webové rozhraní aplikace. Webové rozhraní je dostupné na portu, který byl definovaný v rámci konfigurace docker kontejneru v předešlé kapitole. Kontrolu načtené konfigurace je možné provést v záložce *Settings*. Na této obrazovce jsou dostupné všechny konfigurační soubory, čas, kdy byly načteny a obecná konfigurace samotné aplikace.



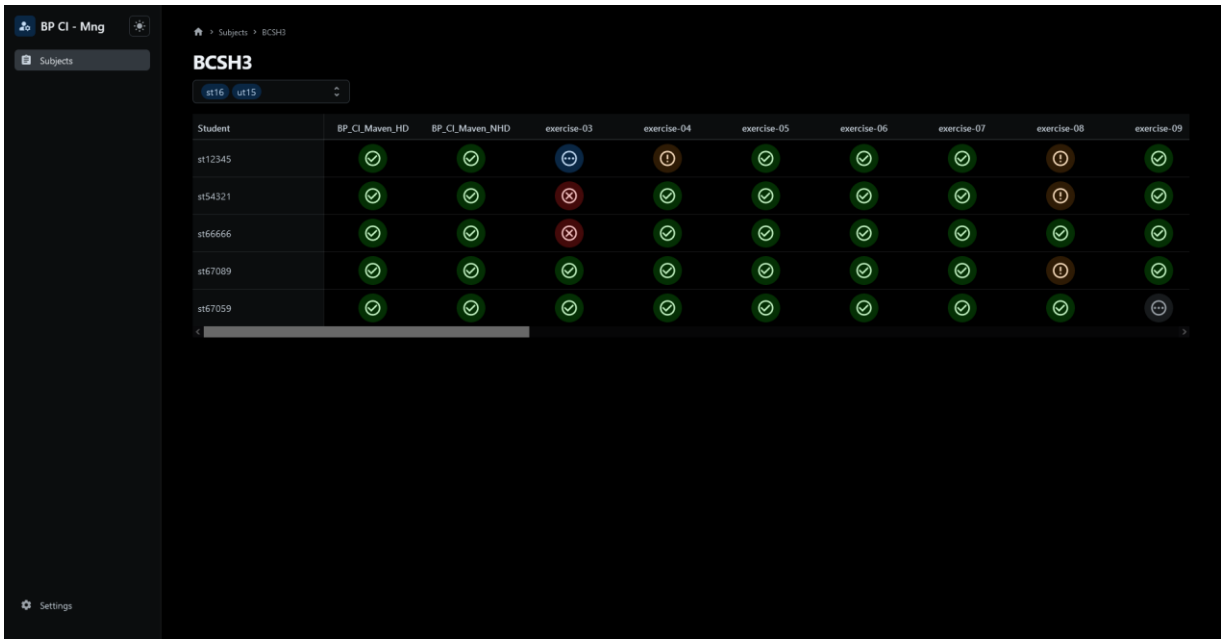
Obrázek 6 – Obrazovka nastavení aplikace

Na záložce *Subjects* je dostupný výpis všech předmětů a lze si zobrazit zadání, která byla pro daný předmět vytvořena. Pro každé zadání je na této obrazovce také vidět název, termín odevzdání a další informace.



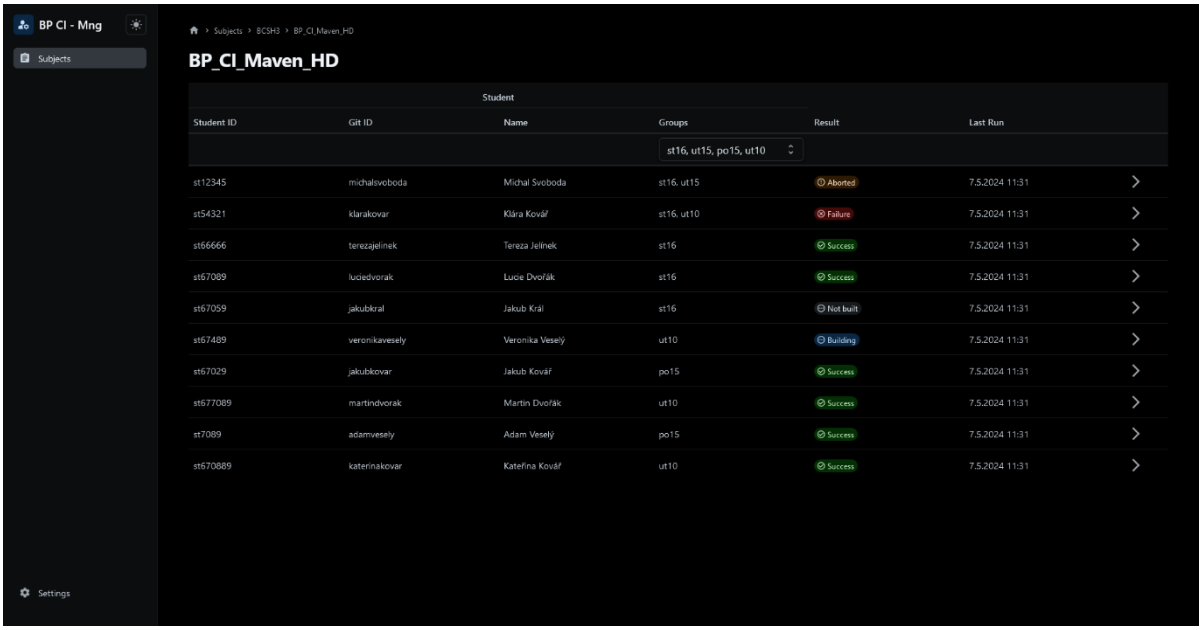
Obrázek 7 – Obrazovka s přehledem všech předmětů

Pro každý předmět lze zobrazit matici obsahující informace o poslední kontrole všech prací tohoto předmětu. Tato matice obsahuje všech studenty a stav kontrol pro všechny zadání.



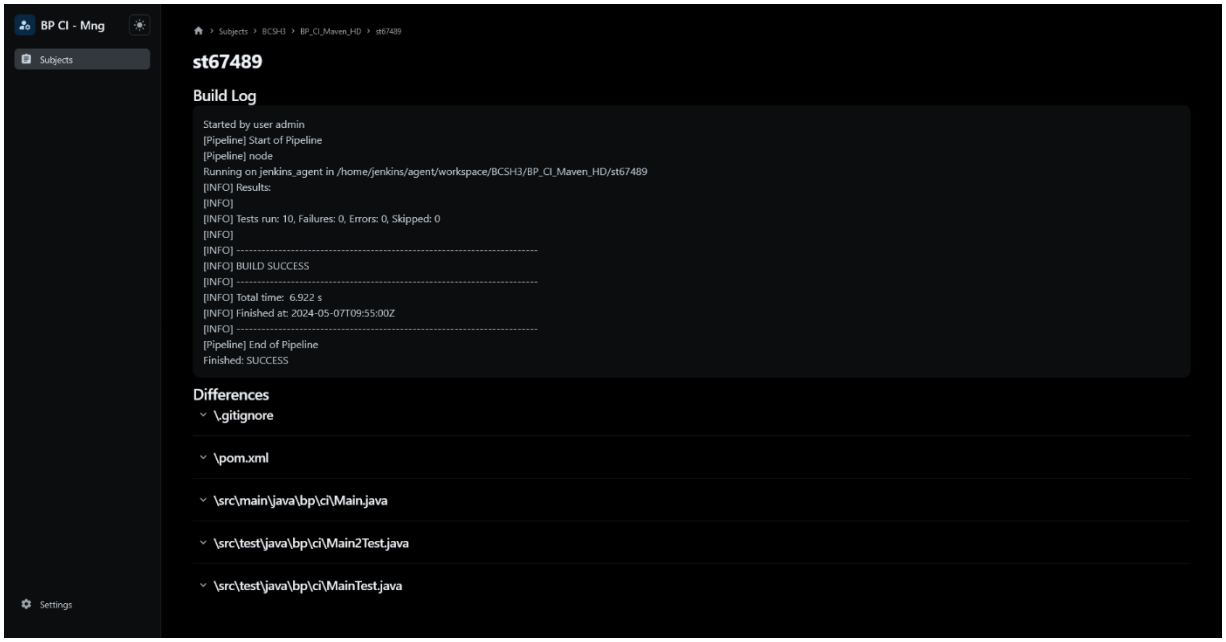
Obrázek 8 – Obrazovka s maticí výsledků testování všech zadání pro předmět

Poté je možné detailnější výpis testování úloh pro zadání. Obrazovka dovolující toto zobrazení obsahuje informace o všech studentech a jejich základních údajích včetně stavu poslední kontroly jejich zadání.



Obrázek 9 – Obrazovka pro zadání a stavu všech studentů

Nakonec existuje obrazovka prostřednictvím, které lze zobrazit detail studentského řešení. Tato obrazovka obsahuje záznam posledního testování řešení studenta. Zároveň jsou zobrazeny změny v souborech, které student provedl. Porovnání je zobrazeno oproti souborům, které se nacházejí v repozitáři zadaná úlohy. *Build Log* byl zkrácen pro účely dokumentace této obrazovky.



Obrázek 10 – Obrazovka pro zobrazení vyhodnocení studentského repozitáře – Build log

Obrázek 11 – Obrazovka pro zobrazení vyhodnocení studentského repozitáře – Differences

ZÁVĚR

Tato bakalářská práce měla za cíl vytvořit systém pro automatizovanou kontrolu školních úloh. V teoretické části mělo být popsáno využití verzovacího systému *Git* a techniky CI ve školním prostředí pro odevzdávání programovacích úloh.

V první kapitole byly obecně popsány verzovací systémy a byla popsána terminologie, která se využívá v kontextu s verzovacími systémy. V následující části této kapitoly byl popis zaměřený na verzovací systém *Git* a jak se liší od ostatních verzovacích systémů

Druhá kapitola byla věnována popisu platforem, které slouží pro správu verzí a jsou založené na systému *Git*. Byla popsána motivace jejich existence a vlastnosti, které tyto platformy mají. Nejznámější platformy z této oblasti byly popsány, jaké nabízejí možnosti a bylo provedeno jejich porovnání mezi sebou. Na konci této kapitoly byla popsána problematiky využití těchto nástrojů ve školním prostředí a jak je lze využít pro efektivnější kontrolu studentských prací.

Třetí kapitola byla zaměřena na popis CI/CD nástrojů. V úvodu bylo provedeno popis těchto nástrojů a proč jsou užitečné. Následně byly popsány termíny pro kontinuální integrace, doručování a nasazení společně s pojmem CI/CD pipeline. Přestaveny byly nejznámější CI/CD nástroje a byly popsány jejich vlastnosti a čím se odlišují od konkurence. Tyto nástroje pak byly mezi sebou porovnány. Nakonec byl popsán způsob užití těchto nástrojů ve školním prostředí pro automatizovanou kontrolu studentských prací.

Poslední čtvrtá kapitola byla věnována popisu praktické části této práce. V první části byl představec cíl implementovaného systému. Byly popsány požadavky, které tento systém vyžaduje. Pro každý požadavek bylo popsáno jeho řešení a přístup ke stavům, které mohou nastat. Byla popsána struktura konfiguračního souboru pro definici předmětů, úloh a studentů. Následoval popis struktury systému samotného a komponent ze kterých se skládá. V počátku byl popsán docker compose a motivace jeho využití v práci. Následoval popis pro jednotlivé komponenty, proč byly vybrány a jakou roli mají v systému. U každé komponenty nechyběl popis pro docker kontejner. Aplikace pro správu systému byla popsána více do detailu a byla popsána vnitřní struktura aplikace. Byly také popsány všechny její konfigurační parametry. U popisu databáze byl poskytnut model databázové aplikace.

V praktické části této práce byl navržený funkční systém pro automatizovanou kontrolu studentských prací. Systém umožňuje konfiguraci předmětů, zadání a studentů prostřednictvím vzdáleného *Git* repozitáře. Pro zadání definovaná v konfiguraci provádí pravidelnou kontrolu

studentských repozitářů a v případě detekce změn provede jejich otestování. Výsledky testování jsou dostupné prostřednictvím webového rozhraní. Systém byl otestován a lze ho nasadit prostřednictvím platformy *Docker*.

POUŽITÁ LITERATURA

[1] CHACON, Scott a Ben STRAUB. *Pro Git*. 2nd ed. Apress, 2014. ISBN 1484200772.

[2] What is version control? *Atlassian* [online]. ©2024 [cit. 2024-05-05]. Dostupné z: https://www.atlassian.com/git/tutorials/what-is-version-control

[3] Git - gitglossary Documentation. *Git-scm.com* [online]. 2024, 23. 02. 2024 [cit. 2024-05-05]. Dostupné z: https://git-scm.com/docs/gitglossary

[4] Best Version Control Software in 2024. *6sense* [online]. ©2024 [cit. 2024-05-05]. Dostupné z: https://6sense.com/tech/version-control

[5] What is Git? *Atlassian* [online]. ©2024 [cit. 2024-05-05]. Dostupné z: https://www.atlassian.com/git/tutorials/what-is-git

[6] EVANS, Julia. Do we think of git commits as diffs, snapshots, and/or histories? *Julia Evans* [online]. [cit. 2024-05-05]. Dostupné z: https://jvns.ca/blog/2024/01/05/do-we-think-of-git-commits-as-diffs--snapshots--or-histories

[7] TOMAYKO, Ryan. Pull Requests 2.0. *The GitHub Blog* [online]. 2010, 31. 08. 2010 [cit. 2024-05-05]. Dostupné z: https://github.blog/2010-08-31-pull-requests-2-0

[8] About issues. *GitHub Docs* [online]. ©2024 [cit. 2024-05-05]. Dostupné z: https://docs.github.com/en/issues/tracking-your-work-with-issues/about-issues

[9] GitLab Pages. *GitLab Docs* [online]. ©2024 [cit. 2024-05-05]. Dostupné z: https://docs.gitlab.com/ee/user/project/pages/

[10] Websites for you and your projects. *GitHub Pages* [online]. ©2024 [cit. 2024-05-05]. Dostupné z: https://pages.github.com

[11] GitLab's AI-assisted Code Suggestions. *GitLab* [online]. ©2024 [cit. 2024-05-05]. Dostupné z: https://about.gitlab.com/solutions/code-suggestions

[12] GitHub Copilot - Your AI pair programmer. *GitHub Copilot · Your AI pair programmer* [online]. ©2024 [cit. 2024-05-05]. Dostupné z: https://github.com/features/copilot

[13] How to build a CI/CD pipeline with GitHub Actions in four simple steps. *The GitHub Blog* [online]. 2022, 02. 02. 2022 [cit. 2024-05-05]. Dostupné z: https://github.blog/2022-02-02-build-ci-cd-pipeline-github-actions-four-steps

[14] KP, Aswin Kumar. Github vs Gitlab vs Bitbucket. *Disbug* [online]. 2022, 21. 12. 2022 [cit. 2024-05-05]. Dostupné z: https://disbug.io/en/blog/github-vs-gitlab-vs-bitbucket

[15] HEYDEN, Nathan Vander. GitHub vs GitLab vs BitBucket: Key Differences & Feature Comparison. *Marker.io* [online]. 2023, 21. 04. 2023 [cit. 2024-05-05]. Dostupné z: https://marker.io/blog/github-vs-gitlab-vs-bitbucket#1-github

[16] Pricing - Plans for every developer. *GitHub* [online]. ©2024 [cit. 2024-05-05]. Dostupné z: https://github.com/pricing

[17] GitLab Pricing. *GitLab* [online]. ©2024 [cit. 2024-05-05]. Dostupné z: https://about.gitlab.com/pricing

[18] Bitbucket - Pricing. *Atlassian* [online]. ©2024 [cit. 2024-05-05]. Dostupné z: https://www.atlassian.com/software/bitbucket/pricing

[19] BELMONT, Jean-Marcel. *Hands-On Continuous Integration and Delivery*. Packt Publishing, 2018. ISBN 9781789133073.

[20] PITTET, Sten. Continuous integration vs. delivery vs. deployment. *Atlassian* [online]. ©2024 [cit. 2024-05-05]. Dostupné z: https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment

[21] What is a CI/CD pipeline? *Red Hat* [online]. 2022, 11. 05. 2022 [cit. 2024-05-05]. Dostupné z: https://www.redhat.com/en/topics/devops/what-cicd-pipeline#ci/cd-pipeline

[22] IBM CLOUD EDUCATION. What Are CI/CD and the CI/CD Pipeline? *IBM Blog* [online]. 2021, 27. 09. 2021 [cit. 2024-05-05]. Dostupné z: https://www.ibm.com/blog/ci-cd-pipeline

[23] Understanding GitHub Actions. *GitHub Docs* [online]. ©2024 [cit. 2024-05-05]. Dostupné z: https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions

[24] Get started with GitLab CI/CD. *GitLab Docs* [online]. ©2024 [cit. 2024-05-05]. Dostupné z: https://docs.gitlab.com/ee/ci

[25] DINGARE, Pranoday Pramod. *CI/CD Pipeline Using Jenkins Unleashed*. Apress, 2022. ISBN 1484275071.

[26] MENNEN, Felix. Feature Parity Comparison: GitLab CI/CD vs. GitHub Actions. *Medium* [online]. 2023, 07. 07. 2023 [cit. 2024-05-05]. Dostupné z: https://thexz3dev.medium.com/feature-parity-comparison-gitlab-ci-cd-vs-github-actions-37401d3e3b1c

[27] What is YAML? *Red Hat* [online]. 2023, 03. 03. 2023 [cit. 2024-05-05]. Dostupné z: https://www.redhat.com/en/topics/automation/what-is-yaml

[28] Docker overview. *Docker Docs* [online]. ©2013-2024 [cit. 2024-05-05]. Dostupné z: https://docs.docker.com/get-started/overview

[29] *Docker Compose overview* [online]. ©2013-2024 [cit. 2024-05-05]. Dostupné z: https://docs.docker.com/compose

[30] Introduction to Liquibase. *Liquibase Documentation* [online]. ©2024 [cit. 2024-05-05]. Dostupné z: https://docs.liquibase.com/concepts/introduction-to-liquibase.html

[31] Mysql. *Docker Hub Container Image Library | App Containerization* [online]. ©2024 [cit. 2024-05-05]. Dostupné z: https://hub.docker.com/_/mysql