FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA INFORMATIKY
A VÝPOČETNÍ TECHNIKY

Diplomová práce

# Technologická sada pro výuku předmětů WEB a OKS

David Kůta

PLZEŇ

2024

FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA INFORMATIKY
A VÝPOČETNÍ TECHNIKY

# Diplomová práce

# Technologická sada pro výuku předmětů WEB a OKS

Bc. David Kůta

**Vedoucí práce**
doc. Ing. Pavel Herout, Ph.D.

**PLZEŇ** **2024**

**Citace v seznamu literatury:**
KŮTA, David. *Technologická sada pro výuku předmětů WEB a OKS*. Plzeň, 2024. Diplomová práce. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky. Vedoucí práce doc. Ing. Pavel Herout, Ph.D.

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2023/2024

# ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

| | |
|---|---|
| Jméno a příjmení: | **Bc. David KŮTA** |
| Osobní číslo: | **A22N0053P** |
| Studijní program: | **N0613A140040 Softwarové a informační systémy** |
| Téma práce: | **Technologická sada pro výuku předmětů WEB a OKS** |
| Zadávající katedra: | **Katedra informatiky a výpočetní techniky** |

## Zásady pro vypracování

1. Seznamte se s hlavními službami, které poskytují verzovací systémy typu Git a jejich možnostmi CI/CD procesu. Dále se seznamte s nástroji pro kontejnerizaci a pro provoz kontejnerů.
2. Podle požadavků na studentské artefakty dodané vedoucím práce navrhněte organizační strukturu úložiště a příslušné aktivity (pipelines, actions, processes, ...) pro zpracování a publikaci jednotlivých artefaktů.
3. Realizujte kompletní technologickou sadu, která bude zpracovávat artefakty odevzdané studenty.
4. Ověřte funkčnost celého systému na ukázkových artefaktech postupně vytvářené jednoduché webové aplikace.
5. Celý postup odevzdávání popište v detailní uživatelské dokumentaci a zhodnoťte dosažené výsledky.

Rozsah diplomové práce: **doporuč. 50 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

dodá vedoucí diplomové práce

Vedoucí diplomové práce: **Doc. Ing. Pavel Herout, Ph.D.**
Katedra informatiky a výpočetní techniky

Datum zadání diplomové práce: **8. září 2023**
Termín odevzdání diplomové práce: **16. května 2024**

L.S.

**Doc. Ing. Miloš Železný, Ph.D.**
děkan

**Doc. Ing. Přemysl Brada, MSc., Ph.D.**
vedoucí katedry

V Plzni dne 11. října 2023

# Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného akademického titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Západočeská univerzita v Plzni má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Plzni dne 14. května 2024

. . . . . .
David Kůta

V textu jsou použity názvy produktů, technologií, služeb, aplikací, společností apod., které mohou být ochrannými známkami nebo registrovanými ochrannými známkami příslušných vlastníků.

# Abstrakt

Diplomová práce se zabývá tvorbou technologického stacku pro výuku předmětů KIV/OKS a KIV/WEB na Západočeské univerzitě v Plzni. Cílem práce bylo vytvoření systému, který umožní studentům efektivně zpracovávat dílčí úlohy semestrální práce a pedagogům zjednoduší kontroly. Byla provedena analýza současně využívaných systémů a technologií v oblastech CI/CD, verzování a kontejnerizace, ze kterých byly následně zvoleny GitLab, Docker a Kubernetes.

V praktické části byl realizován technologický stack, jehož součástí bylo nastavení platformy GitLab, které obsahovalo například také návrh struktury repozitářů. Dále byla vytvořena konfigurace několika CI/CD pipelines, které zpracovávají, validují samostatné úlohy studentů a publikují jejich výsledky na testovací prostředí v Kubernetes Clusteru. Následně byly vytvořeny pomocné nástroje pro práci se systémem a kontejnerové prostředí pro vývoj semestrální práce na jednotné platformě.

# Abstract

The thesis deals with the creation of a technology stack for teaching KIV/OKS and KIV/WEB at the University of West Bohemia in Pilsen. The aim of the thesis was to create a system that will allow students to efficiently process partial tasks of assignments and simplify the controls for teachers. An analysis of currently used systems and technologies in the areas of CI/CD, versioning and containerization was performed, from which GitLab, Docker and Kubernetes were subsequently selected.

In the practical part, a technology stack was implemented, which included the setup of the GitLab platform, which included, for example, the design of the repository structure. Furthermore, the configuration of several CI/CD pipelines was created to process and validate the students' independent jobs and publish their results to the test environment in the Kubernetes Cluster. Subsequently, the help tools for working with the system and the container environment for developing assignments on a unified environment were created.

## Klíčová slova

CI/CD • GitLab • Docker • Kontejnerizace • Technologický Stack • Kubernetes

# Poděkování

Tímto bych chtěl poděkovat doc. Ing. Pavlu Heroutovi, Ph.D. za příkladné vedení této diplomové práce, cenné rady a odborný dohled. Rád bych také poděkoval své rodině a všem blízkým, kteří mě při vytváření této práce podporovali. Nakonec bych rád poděkoval firmě RTsoft s.r.o za poskytnutí zpětné vazby a cenných rad, které do této diplomové práce vnesly pohled z praxe.

*David Kůta*

# Obsah

# Úvod 1

Od akademického roku 2024/25 dochází vzhledem k náběhu nově akreditovaného bakalářského oboru k postupné změně ve složení struktury vyučovaných předmětů. Bylo rozhodnuto, že dojde ke spolupráci na samostatných pracích studentů v předmětech KIV/OKS a KIV/OKS. Spolupráce bude realizována tím způsobem, že studenti budou připravovat svoji webovou aplikaci v předmětu KIV/WEB víceméně stejným způsobem, jako dosud. Ovšem v předmětu KIV/OKS nastane zásadní změna a to, že v samostatné práci budou studenti průběžně testovat svoji v KIV/WEB současně vznikající aplikaci. Z tohoto důvodu již nebude možné pro validaci studentských úloh používat validátor provozovaný na CIV. Pro dosažení co největší možné jednotnosti vyhodnocovacího postupu (náhrada validátoru) bude nutné, aby studenti měli k dispozici jednotný způsob odevzdávání postupně vytvářených částí (artefaktů) jak své aplikace pro KIV/WEB, tak i jejich testů pro KIV/OKS a zároveň jednotnou vývojovou platformu. Odevzdávání prací bude probíhat přes zvolený jednotný verzovací systém. V něm též proběhnou automaticky základní verifikační aktivity odevzdaného artefaktu. Následně bude artefakt a případně i výsledky jeho validace nasazeny či publikovány na testovací prostředí dostupné přes webové rozhraní.

Celý systém odevzdávání studentských úloh by se měl co nejvíce přiblížit dobré praxi ze softwarového průmyslu. Tedy, použije se platforma pro Continuous Integration/Continuous Deployment (CI/CD), různé možnosti kontejnerizace, jako například Docker a jejich efektivní provoz, jako například Kubernetes. Cílem je vytvořit komplexní technologický stack pro výše popsanou kombinaci předmětů, který využije přednosti kontejnerizace a nastavení CI/CD procesu pro validaci jednotlivých úloh. Zároveň by měl technologický stack obsahovat vývojové nástroje, které studentům značně zmírní úsilí pro nastavení prostředí pro zpracování semestrální práce. K tomu se využije Docker například formou připravených docker images, které budou obsahovat potřebné nástroje pro realizaci jednotlivých samostatných úloh.

# CI/CD 2

CI/CD je proces spojující kontinuální integraci (CI), kontinuální dodávku (CD) a kontinuální nasazení. Centrálním prvkem tohoto procesu je tzv. pipeline, která popisuje posloupnost kroků nebo akcí, které jsou vykonány po spuštění definované události. Tou je obvykle nahrání změny do systému správy verzí. Ten poté notifikuje CI/CD platformu, která spustí pipeline a provede požadované akce[Hat24].

## 2.1 Kontinuální integrace

Důvodem pro CI je častá integrace změn kódu do hlavní větve verzovacího systému, používaného při vývoji SW. Tento proces zahrnuje podrobení nových úprav přísným validačním kritériím, včetně sestavení a provádění různých druhů automatizovaných testů. Tento proces slouží k preventivnímu řešení potenciálních integračních překážek, které mohou vzniknout v důsledku odkládání slučování změn až do určeného období vydání. Hlavním prvkem filozofie kontinuální integrace je robustní automatizace testovacích mechanismů, které slouží k zajištění integrity aplikace při každém postupném začleňování nových revizí do hlavní větve verzovacího systému[Atl24b].

## 2.2 Kontinuální dodávka

Kontinuální dodávka rozšiřuje principy kontinuální integrace o automatické nasazení změn kódu po sestavení do testovacího nebo produkčního prostředí. To zahrnuje automatizované procesy testování a uvolňování, které umožňují nasazení aplikace za pomoci nějakého interakce. To znamená, že tato činnost není zcela automatizována (při začlenění do hlavní větve se změna neobjeví na produkci). Frekvenci vydávání lze přizpůsobit potřebám metodice vývoje software [Git22].

## 2.3 Kontinuální nasazení

Kontinuální nasazení představuje další rozšíření nad rámec kontinuálního dodávky. V rámci tohoto režimu je každá změna, která úspěšně projde všemi fázemi pipeline, okamžitě a bez lidského zásahu zpřístupněna koncovým uživatelům. Přijetí kontinuálního nasazení usnadňuje rychlé uzavření smyčky zpětné vazby se zákazníky a snižuje zátěž vývojového týmu tím, že eliminuje potřebu určených „dnů vydání". Tento přístup umožňuje vývojářům soustředit se na vývoj softwaru s tím rozdílem, že software musí být kvalitně pokryt různými druhy testů, aby se zaručilo, že se zákazníkům zpřístupní funkční produkt [IBM24b].

## 2.4 Pipeline

Pipeline definuje činnosti z oblasti kontinuální integrace, jako je například sestavení a spuštění testů, a zároveň může obsahovat akce z kontinuální dodávky nebo nasazení, jako například sestavení spustitelných binárních souborů a nasazení na cílový server. Většina platforem pro CI/CD využívá koncept pipeline jako kód, kde je konfigurace pipeline uložena ve stejném repozitáři jako zdrojový kód software, pro který je pipeline vytvořena. Tento přístup umožňuje členům týmu upravovat, verzovat a kontrolovat konfiguraci jako běžný kód. Konfigurační soubor, obvykle ve formátu YAML, dodržující specifikace platformy, definuje jednotlivé kroky, prostředí pro sestavení aplikace a další parametry. Během provádění pipeline mohou být postupně vytvářeny artefakty, které popisují výsledky jednotlivých částí pipeline. Jedná se například o spustitelné binární soubory, vygenerované logy spuštěných testů, či logy samotných jobů [HD20].

## 2.5 Platformy pro CI/CD

Tyto platformy zprostředkovávají možnosti CI/CD procesu a zároveň mohou integrovat i další DevOps nástroje (např. výkonnostní monitoring, správa kontejnerů. Po nahrání změny do systému správy verzí je odeslána notifikace do CI/CD platformy, která následně provede požadované kroky z definice pipeline. Populární CI/CD platformy[Ort20] budou dále popsány.

### 2.5.1 GitLab CI/CD

Jak je z názvu zřejmé, tato platforma je integrována do služby GitLab. Společně s CI/CD nástroji jsou k dispozici i nástroje pro plánování. Například funkce Auto DevOps automatizuje vytváření, testování, nasazování a monitorování aplikací. Mezi

klíčové funkce patří integrace s poskytovateli cloudových služeb, integrace s Kubernetes clusterem, ChatOps (spouštění CI úloh pomocí chatu) a přehledy reportů vznihlých z jednotkového testování [Git22].

### 2.5.2 Azure DevOps

Poskytuje cloudově hostované pipeline pro Windows, Linux and macOS. Díky využití Azure Pipeline a Azure Artifacts je možné zefektivnit sestavení a nasazení. Dostupná integrace s GitHubem umožňuje vývoj aplikací a nabízí spolehlivá sestavení napříč platformami. Mezi klíčové funkce patří integrace s Kubernetes clusterem, self-hosted CI/CD model a přizpůsobitelné nástroje díky integraci s Azure Board, což zajistí nativní podporu pro využití SCRUM, Agile a Kanban procesů [Azu24].

### 2.5.3 GitHub Actions

Jak je z názvu patrné, tato platforma je integrována do služby GitHub. Pomocí nástroje Actions je možné automatizovat vytváření, testování a nasazování kódu přímo ze služby GitHub. Dále je také možné integrovat nástroje pro nasazení webových služeb, sestavení kontejnerů nebo správa open-source projektů. Společně s Packages (služba GitHubu pro správu balíčků) zjednodušuje správu balíčků, včetně distribuce, aktualizace verzí a řešení závislostí. Mezi klíčové funkce patří podpora Windows, Linux, macOS, ARM, and containers, matrix workflows, vestavěné úložiště tajných informací (např. pro hesla či tokeny) a možnost vlastního hostování runnerů [Git24b].

### 2.5.4 Jenkins

Jenkins, open-source automatizační server proslulý svým rozsáhlým ekosystémem zásuvných modulů, je průkopníkem v oblasti nástrojů pro kontinuální integraci a kontinuální nasazení (CI/CD). Jenkins je nabízen jako self-hosted a nenabízí možnost softwaru jako služby (SaaS). Umožňuje distribuovat pracovní úlohy na více strojích, čímž zvyšuje provozní efektivitu a urychluje realizaci projektů. Jedná se o server pro kontinuální integraci a zároveň sofistikovanou platformu pro kontinuální dodávání. Vyznačuje se samostatnou architekturou založenou na jazyce Java, která zajišťuje nasazení. Klíčové vlastnosti zahrnují rozšiřitelnost, kterou umožňuje robustní rámec zásuvných modulů, jenž otevírá možnosti všestranného přizpůsobení [Jen24].

### 2.5.5 CircleCI

CircleCI je přední nástroj CI/CD pro DevOps, s certifikacemi FedRAMP a SOC2 Type II. Tato platforma je navržena s ohledem na škálovatelnost a flexibilitu, nabízí integrace s mnoha DevOps nástroji a poskytuje přizpůsobitelné rozhraní, rychlé

CI runnery, díky využití mezipaměti pro jednotlivé kroky pipeline. Dále je možné nasadit nástroje kontinuální integrace do Cloudu s možností vlastního hostování runnerů [Dha22].

# 3 Hlavní služby poskytované verzovacími systémy typu GIT

V předchozí kapitole bylo popsáno několik platforem, za jejichž pomoci lze CI/CD provozovat. Cílem této kapitoly je krátce představit hlavní představitele služeb, které poskytují verzovací systémy typu GIT a zároveň možnosti CI/CD procesu, které tyto služby v základu podporují a zároveň jaké jsou možnosti zapisovat pipeline. V současném vývoji softwaru jsou robustní systémy pro správu verzí nepostradatelné. Systém GIT, který je známý svou efektivitou a možností větvit verze kódu, slouží jako základní kámen moderních vývojových postupů. Dle [Bad23] bylo zjištěno, že v současnosti se vyskytují tři hlavní služby, GitLab, GitHub a Bitbucket, které budou v následujících sekcích popsány.

## 3.1 GitLab

GitLab je všestranná webová platforma provozovaná firmou GitLab Inc., která nabízí kromě CI/CD platformy, která bude krátce popsána v další kapitole, nástroje pro podporu všech fází životního cyklu DevOps. Konkrétně se jedná o nástroje pro plánování, spravování kódu za pomoci branching modelu, ověřování nových úprav, správa balíčků, zabezpečení za pomoci například statického aplikačního bezpečnostního testování (SAST), skenování kontejnerů, vydávání nových verzí, konfigurační management, monitoring a správa, která dovoluje promítnout role organizace do role v systému. Díky těmto vlastnostem je možné využívat platformu jako all-in-one řešení pro efektivní vývoj software [Git21]. GitLab je možné využívat přes oficiální doménu gitlab.com, dále je možné provozovat self-managed instanci a v poslední řadě lze využít oficiální poskytované SaaS (Software as a Service) služby. GitLab je možné využívat zdarma ve verzi Free a dále za poplatek ve verzích Premium a Ultimate, které nabídnou více možností než verze Free.

## 3.1.1 Možnosti GitLab CI/CD

### 3.1.1.1 GitLab Runner

GitLab Runner je aplikace, která spouští joby, které jsou deklarované v souboru pipeline. Je možnost využívat dedikované runnery, pokud je využívána instance na GitLab.com. Dále je možnost využívat vlastní instance runnerů, které můžou být nainstalované na libovolné infrastruktuře, pro kterou máme možnost její správy. Je k dispozici několik „Executors“ spouštěčů, které reprezentují platformy, na kterých bude job spouštěn. Jedná se například o SSH, Shell, VirtualBox, Parallels, Docker, Kubernetes a custom executor [Git22].

### 3.1.1.2 CI/CD Pipeline

Pipeline je v systému GitLab definována v YAML souboru, který se může nacházet v repozitáři, pro který je pipeline spouštěna, nebo v jiném repozitáři. Soubor s deklarací pipeline lze imporovat do projektu a následně ji spouštět. Pipeline lze parametrizovat pomocí proměnných, které mohou být deklarovány na vícero úrovních [Git22].

Následuje ukázka 3.1 YAML souboru pro definici pipeline, která se spustí s využitím docker executora.

Zdrojový kód 3.1: Ukázka syntaxe pipeline v GitLab CI/CD

```yaml
# Define global variable
variables:
  PROJECT_NAME: "Sample Project"

stages:
  - build

build_job:
  stage: build
  image: python:3.9
  before_script:
    - echo "Preparing build environment..."
  script:
    - echo "Building the project named $PROJECT_NAME..."
    - echo "Compiling code..."
  after_script:
    - echo "Build complete. Cleaning up..."
  rules:
    - if: '$CI_COMMIT_BRANCH == "main"'
      when: always
```

Vysvětlivky k příkladu 3.1:

- `variables` — Proměnné, kterými lze řídit běh pipeline.
- `stages` — Skupina vzájemně souvisejících jobů.
- `jobs` — Jedná se o bloky, které jsou uvozené jejich názvem a pokračují deklarací (tj. `stage`, `script`, `before_script`, ...) provádí příkazy, které jsou ve vnitřní sekci `script` definované.
- `rules` — Pravidla, které definují, kdy se job přidá do běhu pipeline (v tomto konkrétním určují, že se job zařadí do běhu pipeline pouze pokud je pipeline spouštěna nad větví `main`).
- `image` — Docker image, ve které se vykonávají příkazy jobů.

## 3.2 GitHub

GitHub je rozsáhlá webová platforma provozovaná společností GitHub Inc., která nabízí nástroje pro podporu všech fází životního cyklu DevOps podobně jako GitLab. Mezi hlavní funkce patří správa repozitářů, využívání branching modelu, revize kódu skrze pull requests a code reviews, správa balíčků a zabezpečení aplikací pomocí nástrojů jako je dependency scanning nebo automatické aktualizace zranitelností v balíčkách. Dále také umožňuje konfigurační management, monitoring a CI/CD s využitím GitHub Actions. Platforma je přístupná jak přes webové rozhraní na adrese `github.com`, tak i jako self-hosted řešení s GitHub Enterprise, nebo jako cloudová služba GitHub Enterprise Cloud. GitHub nabízí několik tarifů, včetně bezplatného plánu pro veřejné repozitáře, sudentského plánu a dále placené alternativy s názvem Team a Enterprise s rozšířenými funkcemi pro týmy a organizace [Kha24].

### 3.2.1 Možnosti GitHub Actions

GitHub Actions je robustní nástroj pro automatizaci, testování a nasazení kódu přímo v rámci GitHub repozitářů. GitHub Actions umožňuje vytvářet složité pracovní postupy (workflows) definované v YAML souboru umístěném v repozitáři. Tyto workflow mohou být spouštěny automaticky při událostech jako je push, pull request nebo dokonce naplánované události. Uživatelé mohou využívat jak hostované runnery od GitHubu, tak vlastní runnery instalované na libovolné infrastruktuře. Nabízená běhová prostředí zahrnují Linux, Windows, MacOS a Docker. V rámci pracovních postupů (workflows) je možné definovat proměnné, skupiny úkolů (jobs), které obsahují skripty pro spouštění a pravidla určující, kdy a jak se dané úkoly spustí.

Následuje příklad YAML souboru 3.2 pro GitHub Actions, který provádí stejné činnosti, jako předchozí příklad [Git24a].

Zdrojový kód 3.2: Ukázka syntaxe pipeline v GitHub Actions

```yaml
name: CI Workflow

on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main

jobs:
  build_job:
    runs-on: ubuntu-latest
    container: python:3.9

    steps:
    - name: Checkout code
      uses: actions/checkout@v2

    - name: Preparing build environment
      run: echo "Preparing build environment..."

    - name: Building the project
      run: |
        echo "Building the project named ${{ env.PROJECT_NAME }}..."
        echo "Compiling code..."

    - name: Build complete
      run: echo "Build complete. Cleaning up..."

env:
  PROJECT_NAME: "Sample Project"
```

Vysvětlivky k příkladu 3.2:

- `runs-on` – Specifikuje, na jakém operačním systému budou úlohy spouštěny. Zde se používá ubuntu-latest.
- `container` – Definuje Docker obraz, který se má použít pro job. Tento atribut se používá k definování obrazu Pythonu pro úkol sestavení.
- `env` – Globální proměnné pro celý workflow, zde specifikuje název projektu, který se používá ve skriptech.

## 3.3 Bitbucket

Bitbucket je komplexní webová platforma provozovaná společností Atlassian, která poskytuje robustní nástroje pro podporu všech fází životního cyklu DevOps, analogicky k platformám jako GitHub nebo GitLab. Bitbucket se specializuje na správu repozitářů, využívání modelu větvení, revizi kódu prostřednictvím pull requests a code reviews, a také na správu balíčků. Zabezpečení je zajišťováno pomocí nástrojů pro skenování závislostí a automatických aktualizací zranitelností. Platforma rovněž umožňuje konfigurační management, monitoring a implementaci CI/CD prostřednictvím Bitbucket Pipelines. Bitbucket je dostupný jak přes webové rozhraní na adrese `bitbucket.org`, tak v self-hosted variantě pod názvem Bitbucket Server, nebo jako cloudová služba Bitbucket Cloud. Bitbucket nabízí různé tarify, včetně bezplatného plánu pro malé týmy, studentského plánu, a také placených verzí s názvem Standard a Premium, které poskytují rozšířené možnosti pro větší týmy a organizace [Atl24a].

### 3.3.1 Možnosti Bitbucket Pipelines

Bitbucket Pipelines je integrovaný nástroj pro CI/CD, který umožňuje automatizaci, testování a nasazení kódu přímo v rámci Bitbucket repozitářů. Tento nástroj podporuje vytváření složitých pracovních postupů definovaných v YAML souboru, který je umístěn přímo v repozitáři. Workflow mohou být spouštěny na základě různých událostí, jako jsou push, pull request, nebo naplánované události. Uživatelé mohou využívat jak cloudové běhové prostředí poskytované Bitbucketem, tak vlastní infrastrukturu. Podporovaná prostředí zahrnují Linux, Windows a Docker. V rámci pracovních postupů je možné definovat proměnné, úkoly, které obsahují skripty pro spouštění, a pravidla, která určují podmínky pro spuštění jednotlivých úkolů. [Atl21] Dále následuje ilustrativní příklad YAML souboru 3.3 pro Bitbucket Pipelines, který odpovídá funkčnosti popsané v příkladu pro GitHub Actions a GitLab CI/CD.

Zdrojový kód 3.3: Ukázka syntaxe pipeline v GitHub Actions

```yaml
image: atlassian/default-image:4

pipelines:
  default:
    - step:
        name: Build Job
        image: python:3.9
        script:
          - echo "Preparing build environment..."
          - echo "Building the project named $PROJECT_NAME..."
          - echo "Compiling code..."
```

```
12            - echo "Build complete. Cleaning up..."
13          caches:
14            - pip
15  definitions:
16    caches:
17      pip: ~/.cache/pip
18      node: ~/.npm
19
20  options:
21    docker: true
```

Vysvětlivky k příkladu 3.3:

- `image` – Nastavuje výchozí Docker image pro všechny kroky, pokud není pro konkrétní krok určen jiný obraz.
- `pipelines` – Definuje činnosti, které se provedou v Bitbucket Pipelines, kde default značí pracovní postup, který se spustí při každé události push nebo pull na hlavní větvi.
- `step` – Každý krok v rámci pracovního postupu obsahuje jméno, použitý obraz Dockeru, a skripty, které se mají spustit.
- `caches` – Definuje cache, která se má použít pro určité závislosti, zde pro Python a Node.js, aby se zrychlilo následné stahování a instalace.
- `options` – Umožňuje globální nastavení, zde specificky povolení použití Dockeru pro běh jednotlivých kroků.

# Kontejnerizace 4

Kontejnerizace je „lightweight“ alternativa k plné virtualizaci počítače, která spočívá v zapouzdření aplikace do kontejneru s vlastním běhovým prostředím. Tato technologie umožňuje vývojářům snadno vyvíjet, nasazovat, spravovat a monitorovat aplikace, jelikož se kontejnery vyznačují svou snadnou přenositelností, konzistencí a zároveň zajišťují spolehlivý běh softwaru při spouštění na libovolně různorodém běhovém prostředím. Kontejnerizace má původ v konceptu lodních kontejnerů, které standardizovaly přepravu nákladu, a umožňuje standardizovat dodávání softwaru, přičemž aplikace lze zabalit spolu s jejich závislostmi. Ve srovnání s tradičními virtuálními počítači kontejnerizace poskytuje díky sdílení jádra hostitelského operačního systému a izolací aplikačních procesů od prostředí, efektivitu i rychlost. Stala se klíčovou součástí postupů DevOps pro svou schopnost zefektivnit vývojové cykly a zlepšit škálovatelnost a bezpečnost [IBM24a].

## 4.1 Rozdíl oproti virtuálnímu stroji

Kontejnerizace nabízí přístup k manipulaci s aplikací tím, že umožňuje spouštět aplikace v izolovaných uživatelských prostorech zvaných kontejnery. Na rozdíl od virtuálních počítačů (VM), které vyžadují plnohodnotný operační systém a simulují hardware, kontejnery sdílejí pouze jádro hostitelského systému a izolují pouze aplikaci a její závislosti. Tento rozdíl vede k tomu, že kontejnery jsou ve srovnání s virtuálními počítači výrazně odlehčené, rychleji se spouštějí a jsou méně náročné na zdroje. Zatímco virtuální počítače jsou vynikající pro provoz více různých operačních systémů nebo pro prostředí, kde je nutná úplná izolace operačního systému, kontejnery vynikají v prostředích s vysokou mírou využití přostředků, kde je prioritou maximální efektivita serveru a rychlé nasazení. Snížená režie a vyšší efektivita kontejnerů (která je zajištěna například sdílením jednotlivých vrstev docker image), poskytuje praktické výhody v například v micro-service architekturách a cloudových aplikacích, kde je lze dynamicky spravovat a škálovat [Mic23].

## 4.2 Nástroje pro kontejnerizaci

### 4.2.1 Docker

Docker způsobil revoluci v softwarovém průmyslu tím, že zpřístupnil kontejnerizaci vývojářům softwaru a IT provozu. Poskytuje komplexní ekosystém pro správu kontejnerů, který se skládá z Docker Engine, běhového a balíčkovacího nástroje. Centrální prvek funčnosti je Docker Daemon, což je služba, která je zodpovědná za orchestraci celého životního cyklu kontejneru od sestavení po spuštění. Pro sdílení kontejnerů je k dispozici oficiální platforma s názvem Docker Hub. Docker zjednodušuje proces vytváření, nasazování a správy kontejnerů, čímž výrazně zvyšuje produktivitu vývojářů a efektivitu provozu od mikroslužeb až po tradiční aplikace a usnadňuje bezproblémovou integraci a procesy nepřetržitého nasazování. Rozsáhlý ekosystém Docker, včetně Docker Hub, umožňuje uživatelům sdílet a distribuovat kontejnerové aplikace v různých týmech a prostředích, čímž může podporovat spolupráci [Hed20].

#### 4.2.1.1 Dockerfile

Pro definici docker kontejneru se využívá Dockerfile. Jedná se o textové instrukce, předem definovaných příkazů, které společně sestaví docker image. Výhoda Dockerfile je, že je definován pomocí jednotlivých vrstev, které mohou být napříč kontejnery sdíleny, čímž je docíleno ještě větší efektivity z hlediska vytíženosti prostředků [Doc24b].

Na následujícím příkladu 4.1 bude ukázána tvorba jednoduché docker image, která si jako základ zvolí ubuntu, nainstaluje aktualizace a Python a vypíše „Hello World!“.

Zdrojový kód 4.1: Ukázkový Dockerfile

```
1 FROM ubuntu:18.04
2 LABEL maintainer="name@example.com"
3
4 RUN apt-get update && apt-get install -y python3
5
6 CMD ["python3", "-c", "print('Hello, World!')"]
```

### 4.2.2 Podman

Jako alternativa k Dockeru řeší jeho provozní a bezpečnostní problémy, především nutnost a zároveň závislost na jediném daemonovi. Podman pracuje bez centrálního démona technologií daemonless, čímž zlepšuje zabezpečení systému tím, že

snižuje potenciální styčnou plochu útoku a následnou eskalaci systémových oprávnění prostřednictvím procesu démona. Podman navíc zlepšuje správu kontejnerů tím, že umožňuje uživatelům spouštět kontejnery a pody (skupiny kontejnerů) přímo jako běžný uživatel, aniž by vyžadoval práva roota. Pod Daemonless architekturou si také lze představit, že každý kontejner lze spravovat nezávisle, což zvyšuje stabilitu a spolehlivost systému. Podman bývá vhodný v prostředích, kde je nejdůležitější bezpečnost a kontrola uživatelských prostředků. Další výhodou je, že pro definici kontejnerů využívá stejné příkazy jako Dockerfile [Wil22].

## 4.3 Nástroje pro provoz kontejnerů

V předchozích sekcích byly popsány nástroje pro kontejnerizaci. Následuje proto popis, jakým způsobem mohou být kontejnery spouštěny, spravovány a monitorovány.

### 4.3.1 Docker Compose

Tento nástroj je určený ke zjednodušení procesu definice a správy Docker kontejnerů. Pomocí nástroje Docker Compose (v současnosti dostupný ve verzi 2) mohou být deklarovány konfigurace v YAML souboru a spuštěním jediného příkazu se spustí všechny požadované kontejnery. Mezi kontejnery lze vytvářet virtuální sítě, čímž se docílí toho, že na sebe například aplikace a databáze „uvidí“, což znamená, že spolu budou moci komunikovat. Tato možnost je užitečná zejména pro vývojové, testovací a staging prostředí, kde jsou důležité konzistence a reprodukovatelnost nastavení. Docker Compose podporuje různé funkce, včetně vytváření obrazů, škálování služeb, přidělování prostředků a efektivní správy prostředí orientovaných na služby. [Doc24a]

### 4.3.2 Kubernetes

Kubernetes je robustní systém pro správu a orchestraci kontejnerových aplikací v tzv. k8s (Kubernetes) clusteru. Je navržen tak, aby řešil složitost správy rozsáhlých skupin kontejnerů a zároveň podporoval automatické nasazení, škálování a provoz aplikačních kontejnerů. Kubernetes vyniká v prostředích, kde je vyžadována vysoká dostupnost, škálovatelnost a obnova po havárii. Poskytuje nástroje pro průběžné aktualizace, monitorování a škálování služeb bez výpadků, což je zajištěno možností nastavit strategii nasazení [Kub23].

Následuje krátký příklad ukázek 4.2 a 4.3 deklarace Kubernetes infrastruktury pro spuštění serveru Apache v k8s.

Zdrojový kód 4.2: Ukázka vytvoření Kubernetes deploymentu

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: apache-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: apache
  template:
    metadata:
      labels:
        app: apache
    spec:
      containers:
      - name: apache
        image: httpd:2.4
        ports:
        - containerPort: 80
```

Zdrojový kód 4.3: Ukázka vytvoření kubernetes service

```yaml
apiVersion: v1
kind: Service
metadata:
  name: apache-service
spec:
  type: NodePort
  selector:
    app: apache
  ports:
  - port: 80
    nodePort: 30080
    protocol: TCP
    targetPort: 80
```

#### 4.3.2.1 Vysvětlení předchozích Kubernetes příkladů

- Deployment zajišťuje, že požadovaný počet replik deklarované aplikace (v tomto případě Apache server) bude běžet a udržován aktuální. Deployment automatizuje aktualizace a rollbacky aplikace, což je zásadní pro správu kontejnerizovaných aplikací ve výrobě.
  - `apiVersion` – Specifikuje verzi Kubernetes API, kterou tento soubor využívá.

  - `kind` – Typ objektu, který se vytváří, v tomto případě `Deployment`.
  - `metadata` – Název deploymentu, zde `apache-deployment`.
  - `spec` – specifikace
  - `replicas` – Počet replik aplikace, které mají být udržovány.
  - `selector` – Definuje, jak Deployment identifikuje pody[1], které má spravovat. V tomto případě vybírá pody, které mají label `app: apache`.
  - `template` – Šablona pro vytváření nových podů. Zahrnuje metadata a specifikace podu:
  - `containers` – Specifikace kontejnerů v podu. Zde definujeme jeden kontejner s názvem `apache`, který používá docker image `httpd:2.4` a má přístupný port 80.

- Service v Kubernetes poskytuje stabilní síťovou adresu, na které jsou kontejnery dostupné. To umožňuje komunikaci s aplikací nezávisle na tom, na jakých konkrétních nodes nebo podech aplikace běží.

  - `apiVersion` – Verze Kubernetes API, kterou tento soubor využívá.
  - `kind` – Typ objektu, v tomto případě `Service`.
  - `metadata` – Obsahuje název service, zde `apache-service`.
  - `spec` – specifikace
  - `type` – Typ service, NodePort umožňuje přístup z vnější sítě přímo na porty nodes.
  - `selector` – Definuje, které pody obsluhuje tento service. V tomto případě jsou to pody s labelem `app: apache`.
  - `ports` – Seznam portů, které service vystavuje.
  - `port` – Port, na kterém je service dostupná v rámci clusteru.
  - `targetPort` – Port na podu, na který service směřuje traffic.
  - `nodePort` – Externí port, na který je možné se připojit z vnějšku clusteru.

### 4.3.3 Openshift

Red Hat OpenShift přestavuje rozšíření Kubernetes a poskytuje integrované vývojové a hostingové prostředí pro vytváření a nasazování Docker kontejnerů a správu prostředí Kubernetes. Jedná se o rozšíření typu: nástroje pro vývojáře a technologie

[1] instance kontejneru, která běží v kuberentes clusteru

S2I (source-to-image), která automatizuje vytváření obrazů kontejnerů ze zdrojového kódu. OpenShift navíc nabízí robustní bezpečnostní funkce, zjednodušenou provozní správu a rozsáhlý ekosystém aplikačních služeb. Například díky těmto vlastnostem, může být OpenShift zvolen podniky, které chtějí zavést kontejnerizaci při zachování přísných standardů správy a dodržování předpisů [Hat20].

# Požadavky na studentské artefakty — 5

V akademickém roce 2024/25 dochází k přesunu předmětu KIV/OKS z letního semestru 2. ročníku bakalářského studia do zimního semestru téhož ročníku. To znamená, že bude vyučován současně s předmětem KIV/WEB. Díky těmto okolnostem vzešla od garanta předmětu KIV/OKS snaha propojit semestrální práce na těchto předmětech. V důsledku této iniciativy budou studenti pracovat na svých webových aplikacích v obou těchto předmětech, kdy v rámci předmětu KIV/WEB budou webovou aplikaci vyvíjet a v rámci předmětu KIV/OKS budou ověřovat její kvalitu vytvářením automatizovaných testů.

Doposud tomu bylo tak, že v předmětu KIV/OKS všichni testovali stejnou webovou aplikaci dodanou garantem předmětu. Změna charakteru semestrálních prací bude velice razantní, takže v jejím důsledku bylo nutné zcela přepracovat semestrální práci pro předmět KIV/OKS.

Následuje krátké představení jednotlivých úloh, které budou studenti odevzdávat. Tyto úlohy budou studenti odevzdávat v průběhu celého semestru, to znamená, že termíny odevzdání budou rovnoměrně rozděleny po celém semestru. Popis bude rozdělen do několika sekcí. Cíl, kde je specifikováno, čeho mají prací na úloze dosáhnout. Aktivity, které definují, jaké činnosti budou studenti během zpracování daného úlohy provádět. Následují výstupy, které definují, co bude výsledkem zpracování dané úlohy a nakonec a hodnocení, které specifikuje, jak bude úloha hodnocena. V popisu bude použit pojem „virtuální stroj“, který zastupuje úložiště, kam budou dané úlohy nahrávány a kde bude probíhat automatická validace.

Všechny výstupy studentských úloh (dále „artefakty“) budou každému studentovi nahrány na osobní dashboard virtuálního stroje. Tento dashboard bude jednotným místem kontroly.

## 5.1 Popis úloh

### 5.1.1 01_UC: První kroky k WEB-APP

- **Cíl** – Studenti začínají práci na předmětu tím, že specifikují základní Use Cases (UC) pro svou webovou aplikaci, přičemž vycházejí ze své předchozí samostatné práce z předmětu KIV/DB1.
- **Aktivity** – Studenti modifikují ukázkový HTML soubor, kde definují název, zkratku a popis svého budoucího projektu a specifikují počet vytvořených UC. Tyto informace jsou následně ověřovány vyučujícími KIV/OKS a KIV/WEB což zajišťuje, že projekt splňuje minimální kritéria projektu.
- **Výstupy** – XML a HTML soubory s Use Cases, které jsou automaticky transformovány a nahrány na virtuální stroj pro snadnou navigaci.
- **Hodnocení** – Automatizované kontroly zahrnují validaci XML souborů a manuální kontroly se zaměřují na obsah UC dokumentů.

### 5.1.2 02_DB: Databázové základy

- **Cíl** – Vytvoření a naplnění databázového schématu pro potřeby webové aplikace.
- **Aktivity** – Studenti píší SQL skripty pro vytvoření a naplnění databázových tabulek, respektive tyto převezmou ze své dřívější práce z KIV/DB1 a pouze je upraví. Součástí je i implementace Robot Framework testů (bude využita Database library) pro ověření základních funkcí CRUD operací, dle již existujících UC.
- **Výstupy** – SQL skripty a Robot Framework testy, které jsou spuštěné a ověřené v prostředí virtuálního stroje.
- **Hodnocení** – Automatizované skripty ověřují správnost vytvoření tabulek a naplnění dat, zatímco manuální kontrola se soustředí na logy z testů.

### 5.1.3 03_RQM: Specifikace požadavků

- **Cíl** – Definování a hierarchizace požadavků na software (RQM) v test management systému SquashTM podle UC. To má za cíl, maximalizaci pokrytí veškerých aspektů aplikace a rozvržení hierarchické struktury RQM.
- **Aktivity** – Psaní RQM v nástroji SquashTM, kde každý klíčový UC musí být pokryt alespoň jedním RQM.

- **Výstupy** – Exportované JSON soubory s RQM, které jsou automaticky konvertovány na HTML pro vizuální zobrazení pokrytí UC.
- **Hodnocení** – Kontroly se zaměřují na pokrytí UC v závislosti na RQM a na celkovou úplnost dokumentace RQM.

### 5.1.4 04_TC: Vazby testů na požadavky

- **Cíl** – Přiřazení testovacích případů (Test Cases – TC) k RQM a zajištění, že priorita TC odpovídá prioritě RQM.maximalizace pokrytí veškerých aspektů aplikace. Případná úprava hierarchické struktury TC i zpětně RQM.
- **Aktivity** – Psaní TC ve SquashTM, které jsou strukturovaně přiřazeny k příslušným RQM.
- **Výstupy** – Exportované JSON soubory s TC, které jsou automaticky konvertovány na HTML.
- **Hodnocení** – Automatizované i manuální kontroly zajišťují, že každý RQM je adekvátně pokryt příslušným TC.

### 5.1.5 05_RF-A: Základy uživatelského rozhraní

- **Cíl** – Implementace a testování základních statických HTML stránek pro klíčové UC.
- **Aktivity** – Psaní automatizovaných testů na statické prvky webové stránky v nástroji Robot Framework s využitím Browser Library (RFBL).
- **Výstupy** – Statické HTML stránky (toto je část samostatné práce z KIV/WEB) a strukturované RF testy.
- **Hodnocení** – Kontroly se soustředí na funkčnost a pokrytí UC funkcionálními testy.

### 5.1.6 06_UC_RQM_TC: Finalizace a detaily

- **Cíl** – Upravení a doplnění předchozích UC, RQM a TC o alternativní toky a další detaily, které vycházejí z reálných potřeb vytvářené aplikace.
- **Aktivity** – Aktualizace dokumentů UC a přidání alternativních toků do UC. Reflektování změn a doplnění v příslušných RQM a TC.
- **Výstupy** – XML a JSON soubory s UC, RQM a TC, které jsou automaticky zpracovávané, stejně jako v předchozích částech.

- **Hodnocení** – Sledování úplnosti a adekvátního pokrytí UC, RQM a TC s důrazem na klíčové aspekty všech tří.

### 5.1.7 07_RF: Dynamické testování funkcionalit

- **Cíl** – Implementace a testování klíčových funkcionalit aplikace pomocí dynamických RFBL testů a zároveň úprava testů statických stránek pro dynamickou webovou aplikaci.
- **Aktivity** – Psaní RFBL testů pro UC, které jsou dynamické a zaměřené na funkčnost aplikace.
- **Výstupy** – Hierarchická struktura RF testů, která je spouštěna na virtuálním stroji.
- **Hodnocení** – Kontroly zahrnují jak automatizované, tak manuální ověření funkčnosti všech testů.

### 5.1.8 08_LOG: Integrace a logování

- **Cíl** – Začlenění logovacích mechanismů a principů do aplikace pro sledování operací a chování.
- **Aktivity** – Implementace logování v aplikaci za pomoci knihovny Monolog, včetně nastavení logovacích úrovní a správy log souborů.
- **Výstupy** – Implementované logování v aplikaci, které zvyšuje trasovatelnost případných problémů.
- **Hodnocení** – Kontroly se zaměřují na správnou implementaci logování a na obsah log souborů.

### 5.1.9 09_RF: Rozšířené testování

- **Cíl** – Implementace komplexních end-to-end a negativních testů pro plné pokrytí funkcionalit aplikace.
- **Aktivity** – Psaní rozšířených RFBL testů, které testují aplikaci od začátku do konce a simuluje uživatelské chyby.
- **Výstupy** – Kompletní sada RF testů, které jsou spouštěné a analyzované.
- **Hodnocení** – Kontroly zahrnují jak automatizované, tak manuální ověření funkčnosti všech testů.

### 5.1.10 10_STAT: Refaktoring a optimalizace

- **Cíl** – Zlepšení kvality a údržby kódu na základě výsledků statické analýzy.
- **Aktivity** – Refaktoring kódu pro zvýšení čitelnosti, údržby a efektivity podle standardů definovaných nástrojem PhpStan.
- **Výstupy** – Zdrojový kód aplikace, který je kontinuálně testován a ověřován.
- **Hodnocení** – Automatizované testy a statická analýza zajišťují, že kód splňuje požadované standardy.

# Volba vhodného technologického stacku

# 6

V této kapitole bude diskutován výběr klíčových technologií a nástrojů, které jsou nezbytné pro efektivní vývoj, zpracování a nasazení semestrálních prací, které budou zpracovávány studenty. Zvažování různých alternativ vychází z popisu nástrojů, který byl uveden v několika předchozích kapitolách. Jedná se zejména o kritéria typu výkonnosti, bezpečnosti, komunitní podpory a celkových nákladů na vlastnictví.

## 6.1 Volba úložiště a CI/CD platformy

Výběr vhodného úložiště kódu a CI/CD platformy je klíčový pro efektivní zpracování semestrálních prací a prakticky pro hladký průběh celého semestru z pohledu předmětu KIV/OKS. Jelikož je v plánu systém dlouhodobě využívat, spravovat a rozšiřovat, je důležité zahrnout mezi kritéria pro výběr i zkušenosti s daným produktem na univerzitě.

### 6.1.1 GitLab

GitLab, poskytuje robustní platformu pro správu kódu, CI/CD a monitoring. Dlouholeté zkušenosti s jeho provozem na katedře informatiky a výpočetní techniky (KIV), i na Centru informatizace a výpočetní techniky (CIV) zajišťují efektivní integraci do životního cyklu předmětu.

Platforma nabízí automatizaci na každé úrovni vývojového procesu, což vede k rychlejšímu vývoji a snazší správě projektů. Zároveň nabízí integrovanou CI/CD platformu GitLab CI/CD a možnost integrace s Kubernetes clusterem, což značně zjednoduší nastavení celého systému. Na KIV je vyučován předmět KIV/CICD, který pro svou výuku právě GitLab CI/CD využívá. To zajistí větší možnosti rozšiřitelnosti do budoucna, jelikož studenti získají povědomí o této technologii přímo při studiu. Zároveň je po diskusi s garantem předmetu KIV/OKS, systém zařazen

i z pedagogických důvodů, jelikož studenti budou od počátku studia seznámeni s verzovacím systémem, který je hojně v praxi využíván.

Pro účely kombinace předmětů KIV/OKS a KIV/WEB bude tato služba spuštěna a monitorována prostřednictvím CIV na URL adrese `gitlab-vyuka.kiv.zcu.cz` s univerzitním SSO pomocí ORION[1] konta, které se využívá i na studijním systému IS/STAG[2].

## 6.2 Volba nástroje pro kontejnerizaci

Pro realizaci technologického stacku byl zvolen nástroj Docker, což je široce používaná platforma pro kontejnerizaci aplikací, která umožňuje jednoduché balení aplikací do kontejnerů. Tyto kontejnery jsou pak izolované, přenosné a konzistentní napříč různými prostředími, což výrazně zjednodušuje práci studentů v semestru. Díky izolaci a přenositelnosti kontejnerů bude výrazně zjednodušena původní „inicializace“ studentů v předmětu, kdy odpadnou problémy s verzemi potřebného software a zároveň závislost na operačním systému. Jediná závislost, která vznikne, je Docker. Dále bude díky této technologii výrazně zjednodušené nasazení a škálování a spravování studentských aplikací.

## 6.3 Volba prostředí pro provoz kontejnerů

### 6.3.1 Kubernetes

Kubernetes byl zvolen jako platforma pro orchestraci kontejnerů, nejen kvůli jeho schopnosti škálování a vysoké dostupnosti, ale také kvůli jeho rozsáhlým možnostem monitoringu. Tento aspekt je zásadní, protože do systému budou nasazovány nedokonalé a možná i výkonostně problematické semestrální práce studentů, které je nutné efektivně monitorovat a spravovat. Monitoring v Kubernetes umožňuje sledování výkonu, zatížení a zdraví aplikací v reálném čase [Ren15], což je klíčové pro zajištění stabilního a bezproblémového chodu semestru.

Pro účely kombinace předmětů KIV/OKS a KIV/WEB byl díky CIV zřízen i Kubernetes cluster.

[1] Konto, které slouží k jednotnému přihlášení napříč všemi službami na ZČU.
[2] Systém pro správu studijní agenty na ZČU.

# Návrh organizační struktury úložiště

# 7

V této kapitole bude diskutován návrh struktury, která bude využívána pro výuku předmětů KIV/OKS a KIV/WEB. Cílem tohoto návrhu je zjednodušit správu veškerého obsahu, který budou studenti v těchto dvou předmětech vytvářet. Dále byl při návrhu kladen nárok na ucelení veškerých nástrojů, které jsou vytvořeny pro chod této kombinace předmětů. Pro návrh bude využita možnost členění repozitářů do skupin, kterou GitLab nabízí.

## 7.1 Popis skupin

Jelikož je plánováno, že GitLab, na kterém budou semestrální práce z již zmíněné kombinace předmětů uloženy, bude v budoucnu využíván i v jiných předmětech, byla vytvořena skupina s názvem OKS-WEB, která vymezuje prostor výhradně na již zmíněné předměty. Dále následují skupiny označené číslem, které představuje ročník (a je pojmenována dle roku, ve kterém byl předmět vyučován). V této skupině jsou jednotlivé repositáře studentů vždy pojmenované dle jejich přihlašovacího jména do celouniverzitního systému Orion. Následuje skupina `images`, ve které se nachází veškeré předpřipravené docker images od validačních skriptů po image pro běh aplikace, či testů. Dále skupina `management-tools` sdružující nástroje pro hladký běh předmětu, jako webovou aplikaci pro garanta předmětu (není součástí této diplomové práce) a repozitář, který provádí inicializaci studentů (bude dále blíže popsáno). Nakonec následují pouze repozitáře sloužící jako template projekty pro definici pipeline, projekt, Kubernetes deklarace a konfigurační repozitář pro Kubernetes cluster agenta.

Následuje krátký komentovaný příklad organizační struktury skupin a repozitářů.

- `OKS-WEB` – Hlavní skupina, která zapouzdřuje aktivity a práce předmětu do jedné izolované skupiny

- `2023` – Představuje podskupinu obsahující studenty, kteří plnili semestrální práci v roce 2023 (testovací ročník).
  - `orionlogin` – Představuje repozitář studenta, který obsahuje semestrální práce.
- `images` – Obsahuje docker images, které tvoří technologický stack
- `management-tools` – Obsahuje nástroje a aplikace pro správu předmětu.
- `k8s-definiton` – Obsahuje veškeré deklarace Kubernetes prostředí pro studentské aplikace a Dockerfile, který je nasazován do Kubernetes. Tato extrakce do jiného repozitáře proběhla zejména z bezpečnostních důvodů, kvůli izolaci studentů od deklarací prostředí a zároveň kvůli minimalizaci styčné plochy při úpravách (upraví se pouze na jednom místě)
- `oks-web-kca` – Obsahuje konfiguraci agenta, který zprostředkovává integraci GitLabu a Kubernetes clusteru
- `pipeline-template` – Představuje šablonu pro pipeline, která je stejná pro všechny studenty. Díky extrakci repozitáře s šablonou pipeline mimo repozitáře studentů je docíleno větší bezpečnosti, díky oddělení studentů od definice pipeline a zároveň snazší editace pipeline při úpravě. Změna se ihned projeví u všech studentů.
- `project-template` – Obsahuje template projektu, ze kterého jsou poté tvořeny pomocí technologie `fork` studentské repozitáře.

## 7.2 Popis struktury úložiště pro studenta

Úložiště pro studenta, tj. repozitář slouží k jedinému účelu. Student provádí VCS[1] operace `commit a push` s nastaveným upstreamem na tento repozitář. Tím si zálohuje své výsledky dosažené při semestrální práci a dále si příslušným mechanismem (vytvoří tag) spustí validace odevzdávaných úloh a nasazení stavu, který je momentálně v `main` větvi. Pro svou jednoduchost a přehlednost byla zvolena možnost, kdy budou veškeré části semestrální práce uloženy v jednom repozitáři, který bude mít ale každý student vlastní. Bylo tak učiněno z důvodu, kdy se razantně zjednoduší validace úloh a následná propagace sestavených artefaktů do Kubernetes clusteru. Dále se zvýší přehlednost z pohledu studenta i vyučujícího, jelikož veškerá práce studenta bude uzavřena v jednom celku (repozitáři). Repozitář tedy bude členěn dle úlohy a dle technologie, která je na danou úlohu využita. V každém adresáři v repositáři existuje `README.md` soubor, který přesně popisuje k jaké úloze se daný adresář vztahuje a co je cílem konkrétní úlohy semestrální práce.

[1] Version Control System

Následuje ukázka adresářové struktury repozitáře studenta, kdy se názvy adresářů shodují s identifikátory úloh definovaných v sekci 5.1. Jak je na rozpisu dále vidět, tak sloučení 7. a 9. úlohy proběhlo proto, že student vytváří ucelenou sadu testů pro jednu webovou aplikaci.

- `01_UC` – 1. úloha: XML soubory pro definici případů užití
- `02_DB` – 2. úloha: Databázové Robot Framework testy
- `03_RQM` – 3. úloha: Definice požadavků v SquashTM
- `04_TC` – 4. úloha: Definice testovacích případů a provázání s požadavky v SquashTM (pouze export)
- `05_RF-A` – 5. úloha: Robot Framework testy statických HTML stránek
- `06_UC_RQM_TC` – 6. úloha: Zpřesnění provázání a definice případů užití, požadavků a testovacích případů
- `RF` – 7. a 9. úloha – Robot Framework testy dynamické PHP aplikace
- `WEB_APP` – Obsahuje webovou aplikaci vytvářenou v předmětu KIV/WEB a současně 8. úlohu na integraci logování do webové aplikace
- `WEB_HTML` – Umístění statických HTML stránek z 5. úlohy.

# 8 Realizace technologického stacku

Nejprve je třeba definovat, co se přesně skrývá pod pojmem „technologický stack“. V kontextu předmětů KIV/OKS a KIV/WEB se jedná o soubor všech technologií potřebných pro hladký průběh semestru, zejména pro zpracování semestrálních prací. Další informace o úložišti, tj. GitLab, zde nebudou zmiňovány, jelikož úložiště bylo podrobně popsáno v předchozích kapitolách.

Tato kapitola popisuje, jaké technologie byly zahrnuty do technologického stacku a jak byly implementovány s využitím Dockeru, což umožnilo efektivně zvládat výzvy spojené s různými požadavky na běhové prostředí, které právě výuka předmětů na začátku semestru obsahuje. Při návrhu tohoto stacku byl kladen velký důraz na rozšiřitelnost a znovupoužitelnost. Docker kontejnery zde hrají klíčovou roli díky svým výhodám, jako jsou izolace prostředí a snadná konfigurace [Vas15]. Tato flexibilita umožňuje studentům, aby se soustředili pouze na zpracování a vytváření studentských úloh, aniž by museli trávit nadměrné množství času řešením problémů s kompatibilitou a instalací softwaru.

Díky využití Docker kontejnerů se technologický stack stává nejen robustnější, ale i přizpůsobivější na změny v technologických trendech, čímž je podporován inovativní a flexibilní přístup ze strany vyučujících a jejich tendence vyučovat nejnovější technologie. Výsledkem je, že pokud bude tendence inovovat, aktualizuje se jen požadovaná docker image a ne celý systém, což umožní vytvářet rychleji zpracované aktualizace v menších přírůstcích. Technologický stack se dělí na několik dílčích celků, které budou dále představeny. Jedná se zejména o kontejnerizované nástroje, které ulehčí studentům práci na semestrální práci z předmětu KIV/WEB, dále o kontejnerizované nástroje, které budou plnit stejný účel v předmětu KIV/OKS. Následně byla připravena infrastruktura pro validační skripty, produkční prostředí pro nasazení studentských artefaktů ze zpracovaných úloh. Nakonec byla pro výuku těchto předmětů zprovozněna databáze.

# 8.1 Využití programu make

Pro realizaci a snadnou dostupnost technologického stacku byla zvolena na základě článku [Fow90] a zkušeností z praxe získaných z několika firem zabývající se vývojem webových aplikací technologie make, která bude využita k usnadnění vykonávání příkazů.

## 8.1.1 Úvod k programu make

Tato technologie primárně sloužila k automatizaci zdrojových kódů do binárních souborů, což mohou být například spustitelné EXE soubory. Postup činností definuje soubor pojmenovaný Makefile, který při sestavení cíle (spustitelný .exe soubor) sleduje topologické seřazení dílčích podcílů zapsaných typickou syntaxí [Pat18].

Na ukázce kódu 8.1 je pro úplnost demonstrován ilustrativní příklad překladu a sestavení jednoduchého programu napsaném v jazyce C. Nástroj make tímto způsobem v technologickém stacku pro předměty KIV/OKS a KIV/WEB nebyl využit, jedná se pouze o demonstrativní příklad typické funkčnosti a použití.

Zdrojový kód 8.1: Klasické použití Makefile

```
1 CC=gcc
2 CFLAGS=-Wall
3 TARGET=prog
4 all: $(TARGET)
5 $(TARGET): main.c
6   $(CC) $(CFLAGS) -o $(TARGET) main.c
7 clean:
8   rm -f $(TARGET)
```

## 8.1.2 Využití technologie make v technologickém stacku

Jak již bylo zmíněno, tak technologie make byla využita pouze jako nástroj ke zjednodušení a automatizaci rutinních příkazů.

Hlavní výhodou je, že veškeré příkazy, které jsou k určité činnosti potřeba, budou popsané na jednom přístupném místě. Odpadne tedy například kopírování příkazů do příkazové řádky, jejich hledání (jelikož příkazy většinou nejsou všechny dostupné na jednom místě, ale například mohou být různě „rozdrobené“ napříč zadáními jednotlivých semestrálních úloh).

Další výhodou je, že je využita nejjednodušší syntaxe pro vykonání jednotlivých příkazů. V řeči technologie make je definován pro každý příkaz jeho cíl, který je splněn po vykonání požadovaného příkazu.

Spouštěné příkazy lze i jednoduchou syntaxí parametrizovat, což bude ukázáno dále na příkladu.

Následuje příklad 8.2 definice souboru `Makefile`, který obsahuje definované cíle pro spuštění docker kontejnerů definovaných v docker-compose souboru a dále cíl pro vypnutí docker kontejnerů z docker-compose souboru.

Zdrojový kód 8.2: Využití technologie make v technologickém stacku

```
1 DIR_DOCKER=docker
2 up:
3   cd "${DIR_DOCKER}" && docker-compose up
4 down:
5   cd "${DIR_DOCKER}" && docker-compose down
```

Na začátku `Makefile` souboru lze vidět definice konstanty, která zde určuje adresář, ve kterém jsou konfigurační soubory dockeru. Následně cíl `up`, který je spuštěn voláním `make up` a splněn v moment, kdy se úspěšně automaticky provede příkaz pod ním, tj `cd "${DIR_DOCKER} "&& docker-compose up`. Identicky pro cíl `down`. I na tomto jednoduchém příkazu je vidět, že je významně ulehčeno zapínání a vypínání docker kontejneru, zejména pro uživatele, kteří s dockerem mají minimální zkušenosti a chtějí ho pouze zapnout a vypnout. V obdobném principu jsou deklarovány i další příkazy, které definují technologický stack.

## 8.2 Technologický stack pro KIV/WEB

Předmět KIV/WEB cílí na základy webového vývoje. Je v něm vyučován jazyk PHP a semestrální práce z tohoto předmětu představuje naprogramování jednoduché PHP webové aplikace. Studenti využívají různé správce balíčků. Zejména composer a NPM. Composer je technologie pro správu PHP balíčků a knihoven a NPM je správce balíčků pro JavaScript. Doposud se pro vývoj semestrální práce využívala instalace všech potřebných softwarů na lokální stroje studentů, což způsobovalo problémy s nastavením prostředí. Zároveň každý vyvíjel na jiné verzi PHP s jinou verzí composeru, databáze a NPM, což znamenalo, že řešení stejných problémů u různých studentů mohlo být odlišné. Nově tedy budou všechny zde zmíněné technologie implementovány s využitím dockeru. Lokálně budou tedy studenti vyvíjet na stejném prostředí, na které se jejich práce budou i následně automaticky nasazovat, což bude popsáno v další kapitole. Tímto se opět vyřeší množství problémů. Docker pro KIV/WEB tedy sestává docker image pro server `apache` s instalací PHP, docker image pro databázi a webového nástroje phpMyAdmin pro administraci lokální databáze, a nakonec kontejner s nástrojem NPM.

### 8.2.1 Docker image pro Apache s PHP

Kvůli snadnému rozběhání vývojového prostředí (které je technologicky specifikováno vedením předmětu KIV/WEB) byl nadefinován minimalistický docker image (cca 61 MB) `php-apache`, který obsahuje webový server Apache s nastaveným PHP ve verzi 8.3 a zároveň instalovaným (pomocí multi stage buildu[1]) nástrojem composer ve verzi 2.6.6. Pokud existuje ve webové aplikaci soubor `composer.json`, tak image automaticky při startu stáhne potřebné aplikační závislosti (composer balíčky). Manipulace s touto docker image je zajištěna pomocí docker-compose souboru, což bude popsáno dále.

### 8.2.2 Docker image pro databázi s phpMyAdmin

Vedení předmětu KIV/WEB plánuje zachovat databázi MariaDB v kombinaci s webovým rozhranním pro administraci phpMyAdmin. Pro databázi nebyl vytvořen dedikovaný docker image, protože byl použit oficiální image z veřejného repozitáře Docker Hub, který byl pouze deklarován v docker-compose souboru. Pro phpMyAdmin byl díky existenci oficiálního docker image zvolen stejný přístup.

### 8.2.3 Docker image pro Node kontejner

Z požadavků vedení předmětu KIV/WEB plyne, že studenti mohou využít pro získávání javascript balíčků NPM. Proto byl využit rovněž oficiální docker image pro technologii Node, který se při startu celého docker-compose stacku pouze spustí a provede instalaci NPM balíčků, pokud existuje soubor package.json, jinak se image vypne, kvůli úspoře prostředků. Pokud by nastala situace vyžadující spuštění instalace při vývoji, například u přidání nového nebo odebrání starého balíčku, je pro tento příklad připravena make akce `make sh`, pomocí které může uživatel manipulovat s NPM dle libosti.

### 8.2.4 Manipulace s technologickým stackem pro KIV/WEB

Veškeré nástroje, které jsou relevantní pro předmět KIV/WEB jsou umístěny do docker-compose souboru, který je umístěný adresáři pro webovou aplikaci v podadresáři `docker`. Následuje příklad docker-compose souboru 8.3, ve kterém bude ukázána jeho struktura a využité docker images.

[1] Technika sestavení docker image, kde je použito za účelem efektivního využití prostředků použito vícekrát klíčové slovo `FROM`

Zdrojový kód 8.3: docker-compose soubor pro KIV/WEB

```
version: '3.8'
name: oks-web-docker
services:
  php-apache:
    container_name: OKS_WEB_apache
    image: registry.gitlab-vyuka.kiv.zcu.cz/oks-web/images/
       php-apache/php83:latest
  ...
    depends_on:
      - database

  database:
    image: mariadb:11.3.2
    container_name: OKS_WEB_mariadb
  ...

  phpmyadmin:
    container_name: OKS_WEB_phpmyadmin
    image: phpmyadmin:5.2.1
  ...
    depends_on:
      - database

  node:
    image: registry.gitlab-vyuka.kiv.zcu.cz/oks-web/images/
       node/node:latest
    container_name: OKS_WEB_node
  ...
    depends_on:
      - php-apache
```

Jak je vidět na ukázce 8.3, tak docker images, které byly v rámci této diplomové práce předpřipraveny, jsou ve výchozím stavu nastavené na tag `latest`, čímž je docíleno automatické aktualizace (zejména ze strany studentů je toto žádoucí chování). Na místech, kde se v ukázce vyskytují tečky se vyskytují další konfigurace, které avšak pro pochopení nejsou podstatné. Ostatní docker images, které čerpají z oficiálních úložišť jsou zafixovány na nejnovější verzi v době zpracování této diplomové práce.

#### 8.2.4.1 Manipulace s KIV/WEB dockerem

Manipulace s dockerem pro KIV/WEB je zajištěna nástrojem `make`. `Makefile` je umístěný v adresáři s webovou aplikací, přičemž jsou v něm definovány akce, které lze s technologickým stackem provádět. Tyto akce se vykonávají vždy v rámci tech-

nologického stacku, tj. v dockeru. Následuje přehled akcí, které provádí jednotlivé cíle `Makefile`:

- `make up`: Spustí docker-compose stack (aplikace je dostupná na localhost:80, databáze na localhost:3306, phpMyAdmin na localhost:8081).
- `make down`: Vypne docker-compose stack.
- `make down-rmi`: Vypne docker-compose stack a smaže docker images a jejich data.
- `make restart`: Restartuje docker-compose stack.
- `make install`: Spustí v dockeru příkaz composer install, na základě `composer.json` souboru definovaného v adresáři s webovou aplikací, v příkazové řádce docker kontejneru s PHP.
- `make phpstan-min`: Spustí v dockeru statickou analýzu pro splnění minimálních kritérií pro KIV/OKS (viz dále).
- `make phpstan-dop`: Spustí v dockeru statickou analýzu pro splnění doporučených kritérií pro KIV/OKS.
- `make phpstan-max`: Spustí v dockeru statickou analýzu pro splnění maximálních kritérií (v KIV/OKS se nehodnotí).
- `make sh`: Spustí shell v kontejneru s PHP.
- `make node-install`: Spustí příkaz `NPM install` na základě `package.json` souboru definovaného v adresáři s webovou aplikací, v příkazové řádce docker kontejneru obsahující NPM.
- `make node-sh`: Spustí `shell` v kontejneru obsahující NPM

## 8.3 Technologický stack pro KIV/OKS

Jak již bylo řečeno, pro předmět KIV/OKS byly garantem předmětu vytvořeny zcela nové požadavky na dílčí úlohy semestrální práce, proto bylo třeba realizovat úplně nový technologický stack, pomocí kterého budou studenti semestrální práci zpracovávat.

Při jeho návrhu byly vzaty v úvahu veškeré problémy, které v průběhu výuky předmětu KIV/OKS v minulosti pravidelně nastávali. Jednalo se zejména o problémy s nastavením prostředí při instalaci správného software pro tvorbu semestrálních prací, dále například problémy s nainstalovanými různými verzemi software.

Po diskusi s garantem předmětu byly všechny tyto požadavky zapracovány do nového technologického stacku. Díky již dříve popisovaným výhodám kontejnerizace byl opět veškerý technologický stack dockerizován a společně s využití technologie `make` byl pro každou část technologického stacku vytvořen záznam v globálním `Makefile`, který se nachází v kořenovém adresáři studentova repozitáře (úložiště).

Následuje krátký popis veškerých částí technologického stacku.

### 8.3.1 Transformace případů užití

Pro úlohy `01_UC` a `06_UC_RQM_TC` je zapotřebí zajistit konverzi XML souborů, pomocí kterých jsou definovány případy užití, na publikovatelné HTML soubory. Pro tuto konverzi je využívána XSLT transformace. Na žádost garanta předmětu KIV/OKS byla vybrána Java knihovna `saxon`, která byla posléze dockerizována tak, aby bylo možné transformaci provést jak v pipeline, tak lokálně na studentských strojích. V podstatě kdekoli, kde lze spustit docker kontejner.

Následuje komentovaný příklad docker image , která zajišťuje požadovanou konverzi.

Zdrojový kód 8.4: Dockerfile pro transformaci UC

```
1  FROM openjdk:11.0.10-jdk-slim
2  WORKDIR /saxon
3  COPY saxon/* .
4  RUN chmod +x transform.sh
5  RUN chgrp -R 0 /saxon && \
6      chmod -R g+rwX /saxon
7
8  ENV PATH="/saxon:${PATH}"
9
10 COPY convertUc /usr/local/bin/
11 RUN chmod a+x /usr/local/bin/convertUc
12
13 ENTRYPOINT ["convertUc", "/saxon/XML", "/saxon/HTML"
```

Na ukázce 8.4 je vidět docker image, která přiloží potřebné soubory k provedení konverze (nachází se v adresáři `saxon`). Následně přiřadí příslušná práva, aby byl skript spustitelný a zároveň byl v proměnné prostředí `PATH`, což zajistí možnost jeho spouštění z libovolného adresáře a nakonec nastavení `ENTRYPOINT`, které zajistí snadné použití image na lokálním prostředí. Při tomto typu spouštění stačí k `image` připojit `volumes` s adresáři pro XML a HTML na příslušná místa definovaná sekci `ENTRYPOINT`.

### 8.3.2 Databázové testy

Pro databázové testy z úlohy `02_DB` byl dle požadavků garanta předmětu KIV/OKS zvolen Robot Framework s využitím jeho rozšíření Database Library. Tato technologie byla také kontejnerizována. Na následující ukázce 8.5 tvorby docker image je vidět instalace pip balíčků `pymysql, robotframework, robotframework-databaselibrary` vycházející z minimalistické docker image `python:3.12-slim`, ze které vychází i ostatní docker image v celém technologickém stacku, které vychází z image pro Python.

Zdrojový kód 8.5: Dockerfile pro databázové testy

```
FROM python:3.12-slim
RUN pip install pymysql \
    robotframework \
    robotframework-databaselibrary

WORKDIR /tests

ENTRYPOINT ["robot"]
```

### 8.3.3 Generátor pokrytí pro požadavky, testovací případy a případy užití

Garantem předmětu KIV/OKS byl dodán Python program (ve skutečnosti je to kombinace dvou programů, kdy jeden připraví data pro konverzi a druhý konverzi provede, ale pro jednoduchost bude v textu zmiňován jako jeden program), který generuje HTML soubor pokrytí ze souborů pocházejících z exportů ze systému SquashTM a již existujících UC. Výstup programu na generování pokrytí je tedy 2D tabulka zachycující závislost požadavků a případů užití, dále požadavků a testovacích případů, a nakonec případů užití a testovacích případů.

Testovací případy a požadavky jsou reprezentovány v programu SquashTM. To, jaký druh pokrytí se vygeneruje, závisí na volbě parametrů příkazové řádky pro generátor pokrytí. Manipulace s generátorem pokrytí byla opět zjednodušena pomocí nástroje `make`. Jedním příkazem se tedy spustí generátor pokrytí v příslušné docker image, čímž budou opět využity veškeré výhody dockeru.

Tvorba docker image pro generátor pokrytí je na stejné bázi, jako tvorba docker image v předchozím příkladu, kdy se navíc přidají pouze `.pyz` balíčky obsahující generátor pokrytí.

Pro efektivní správu a aktualizace generátoru pokrytí byla v repozitáři s generátorem pokrytí vytvořena pipeline, která vytvoří pyz balíčky a následně sestaví docker image pro generátor pokrytí.

Spoustění generátoru pokrytí ze strany uživatele probíhá z kořenového adresáře repozitáře studenta, kde lze zadat následující příkazy:

- `make rqm_03 STUDENT_URL="https://www.example.com" RQM_FILENAME="Vzor_RQM.json"`
  - `STUDENT_URL` – URL adresa, na níž se vyskytují HTML UC a může být i lokální (Pro správnou funkcionalitu by to měla být adresa, na které lze zadat `/01_uc` a zobrazí se seznam vygenerovaných HTML UC).
  - `RQM_FILENAME` – Název `.json` souboru obsahující export požadavků z programu SquashTM, který se vyskytuje v adresáři 03_RQM
  - Výsledek – Bude vytvořen adresář `reports`, ve kterém bude vytvořen soubor `uc-rqm.html` obsahující pokrytí případů užití a požadavků.
- `make tc_04 STUDENT_URL="https://www.example.com" RQM_FILENAME="Vzor_RQM.json" TC_FILENAME="Vzor_TC.json"`
  - `STUDENT_URL` – URL adresa, na níž se vyskytují HTML UC (stejně jako v předchozím případě).
  - `RQM_FILENAME` – Název `.json` souboru obsahující export požadavků z programu SquashTM, který se vyskytuje v adresáři 03_RQM.
  - `TC_FILENAME` – Název `.json` souboru obsahující export testovacích případů z programu SquashTM, který se vyskytuje v adresáři `04_TC`.
  - Výsledek – Bude vytvořen adresář `reports`, ve kterém budou vytvořeny soubory `rqm-tc.html` a `uc-tc.html` obsahující pokrytí požadavků v závislosti na testovacích případech a případů užití v závislosti na testovacích případech.
- `make tc_04-robot STUDENT_URL="https://www.example.com" RQM_FILENAME="Vzor_RQM.json" TC_FILENAME="Vzor_TC.json" RF_OUTPUT_FILEPATH="05_RF-A/reports/output.xml"`
  - `STUDENT_URL` – URL adresa, na níž se vyskytují HTML UC (stejné, jako v předchozím případě).
  - `RQM_FILENAME` – Název `.json` souboru obsahující export požadavků z programu SquashTM, který se vyskytuje v adresáři 03_RQM.

  - `TC_FILENAME` – Název `.json` souboru obsahující export testovacích případů z programu SquashTM, který se vyskytuje v adresáři `04_TC`.
  - `RF_OUTPUT_FILEPATH` — Cesta k souboru `output.xml`, který je výstupem běhu Robot Framework testů.
  - Výsledek – Bude vytvořen adresář `reports`, ve kterém budou vytvořeny soubory `rqm-tc.html` a `uc-tc.html` obsahující pokrytí požadavků v závislosti na testovacích případech a případů užití v závislosti na testovacích případech, který je navíc doplněn o skutečné výsledky testů.

- Stejné varianty existují i pro 6. úlohu.

### 8.3.4 Robot Framework UI testy

Pro běh tohoto typu testů byla využita oficiální podporovaná docker image, která je uložena na veřejném úložišti Docker Hub. Její využití z uživatelského hlediska je opět zjednodušeno použitím technologie `make`, kdy se například testy pro 5. úlohu spouští takto:

- `make rf-a_05`
  - Spustí všechny testy.
- `make rf-a_05 TEST="TS_A/TS_A_01/TS_A_01_01.robot"`
  - Spustí jeden testovací soubor. Případně lze spustit i více testovacích souborů, pokud by se zadala cesta na celý podadresář.
- Stejným způsobem je realizováno i spouštění testů pro 7. a 9. úlohu.

### 8.3.5 SquashTM

Garantem předmětu KIV/OKS byl pro výuku vybrán test management systém SquashTM ve verzi 6.0.1. Pro používání byla opět využita dostupná oficiální docker image. Pro manipulaci byly opět vytvořeny příslušné sekce v `Makefile` v kořenovém adresáři studentského repozitáře. Následuje krátká ukázka manipulace:

- `make squash` – Zapnutí kontejneru obsahující SquashTM (je dostupný na localhost:8080.
- `make squash-stop` – Vypnutí kontejneru obsahující SquashTM.

## 8.4 Validační skripty

Studenti odevzdávají (commitují) postupně vytvářené úlohy do osobního repozitáře, a tyto je třeba nějak úlohy validovat. Termín „validovat“ zde znamená automatickou kontrolu splnění základních náležitostí odevzdané úlohy (např. jméno odevzdaného souboru apod.). Skutečné ověření smysluplnosti úlohy musí následně provést garant předmětu ručně pro každou úlohu. Pro tuto funkcionalitu byla vytvořena architektura a systém pro začlenění validačních skriptů životního cyklu vytvářené úlohy (bude popsán v kapitole týkající se aktivit pro zpracování jednotlivých artefaktů). Validační skripty jsou realizovány v jazyce Python, kdy se jedná o jednoduché skripty, které se spouští s parametry příkazové řádky a validují stav stav úlohy před i po provedení akce v pipeline. To znamená, že pokud je cíl validace validní, skript skončí s návratovou hodnotou `0`. Pokud validní není, skončí s nenulovou návratovou hodnotou a vypíše, nalezený problém. Toto chování zajistí i korektní výsledek jobu v pipeline, který je určen právě návratovou hodnotou programu.

### 8.4.1 Struktura repozitáře pro validační skripty

Jako součást této práce byla navržena architektura a struktura repozitáře, který reprezentuje uložení, testování (ověření pomocí pytest[2] testů) a sestavení docker image pro validační skripty. Následuje popis struktury pro validační skripty:

- `00_docker` – Adresář pro definici docker image pro validační skripty.
  - `Dockerfile` – Dockerfile stejného typu, jako například u generátoru pokrytí. Akce v Dockerfile sestávají z kopírování připraveného adresáře (příprava pomocí skriptu `prepare_files.sh`), dále instalace pip balíčků potřebných pro validační skripty a následně umístění validačních skriptů do `path`, aby bylo možné spustit skripty na libovolném místě na souborovém systému (což je výhodné v pipeline).
  - `prepare_files.sh` – skript, který z adresářů úloh vykopíruje validační skripty a sloučí u všech úloh soubory `requirements.txt` (pokud existují) za účelem jednoduchého sestavení docker image.
- XX_ZKRATKA_ÚLOHY – Adresář obsahující validační skripty pro danou úlohu (ve skutečnosti je vytvořen stejný počet adresářů, kolik existuje úloh, které potřebují validovat validačními skripty)
  - `tests` – Testy pro validační skripty (ověřují testy jako celek a to tak, že je skript spuštěn s různými parametry na různých datasetech, přičemž se zkoumá korektní reakce).

[2] Python knihovna pro unit testy

    * `resources` – Obsahuje datasety pro negativní a pozitivní testy.
    * `test_positive.py` – Pytest testy pro happy day scenario [3].
    * `test_negative.py` – Pytest negativní testy (testy alternativních toků).
- skripty `check_<nazev>.py` – validační skripty.
- `requirements.txt` – Soubor obsahující popsané závislosti validačních skriptů na pip balíčky.

### 8.4.2 Struktura validačního skriptu

Validační skript začíná načtením parametrů příkazové řádky a stupňuje úrovně validace od obecného ke konkrétnímu. To znamená, kontroly začínají zjištěním existence požadovaného souboru, či jeho neexistence a případně pokračují se zkoumáním obsahu těchto souborů. Při návrhu tedy byla snaha o to, aby závažnější chyby byly odhaleny dříve než ty méně závažné. Lze tedy říct, že struktura skriptu je následující:

1. Načtení parametrů příkazové řádky skriptu.
2. Validace parametrů příkazové řádky.
3. Požadované kontroly seřazené od velmi závažných k méně závažným.

## 8.5 Databáze

Pro běh kombinace předmětů KIV/OKS a KIV/WEB byla zřízena centrální databáze MariaDB, která obsahuje stejnou verzi, jako databáze pro KIV/WEB, kterou studenti budou používat pro lokální vývoj. Na tomto databázovém serveru budou mít všichni studenti předmětů KIV/WEB a KIV/OKS vytvořené konto, které bude sloužit pro výukové účely. Jméno a heslo bude stejné, jako jejich orionlogin. Toto vytvoření přihlašovacích údajů proběhne při inicializaci studenta, které bude popsáno dále. Tato databáze bude sloužit převážně k zobrazení stavu artefaktů, které budou zpřístupněny pomocí dashboardu, který bude popsán dále. K databázi bude z bezpečnostních důvodů povolen přístup pouze ze sítě univerzity, což znamená, že pro vzdálený přístup pak bude nutné použít VPN.

[3] Situace, kdy validace proběhne úspěšně.

## 8.6 Produkční prostředí studentských aplikací

Jak již bylo naznačeno, studenti budou mít výsledky svých semestrálních prací zpřístupněny pomocí dashboardu, který bude publikován v Kubernetes clusteru. Ten byl zřízen organizací CIV. přičemž ho bude tato organizace i monitorovat. Z důvodu lepší trasovatelnosti problémů a monitoringu bude mít každý student dedikovaný Kubernetes namespace, ve kterém bude nasazen personalizovaný dashboard studenta. Název namespace bude jednoznačně přiřaditelný ke studentovi, který v něm provozuje svou aplikaci, jelikož je v něm zakomponován rok, ve kterém student studoval a dále orionlogin studenta. Tento namespace vznikne při inicializaci pracovního prostředí studenta, který bude popsán dále. Tento namespace bude vytvořen tak, že manipulace s ním bude umožněna pouze z repozitáře, ke kterému je namespace přiřazen. V následujících ukázkách bude vystupovat termín „pod“, který v Kubernetes světě reprezentuje nejmenší a nejjednodušší nasaditelnou jednotku (kolekci jednoho nebo více kontejnerů), která může být vytvořena a spravována v Kubernetes.

Takto definované produkční prostředí, společně s databází, která byla popsána výše, bude sloužit k reprezentaci výsledků úloh, které budou studenti odevzdávat do GitLabu.

### 8.6.1 Struktura objektů v Kubernetes namespace pro studentské aplikace

V namespace, který má každý student dedikovaný, se nachází několik kubernetes resources, které budou následně popsány. Jako první je Deployment, který je nastaven tak, aby spravoval životní cyklus kontejnerizované aplikace, tj. dashboardu [Gup19]. Následuje popis deklarace deploymentu.

Zdrojový kód 8.6: Kubernetes Deployment resource pro aplikace studentů (soubor `app.yaml`)

```
1 apiVersion: apps/v1
2 kind: Deployment
3 metadata:
4   name: student-app  # Pojmenovani deployment objektu.
5   labels:
6     app: student-app  # Slouzi k identifikaci vsech zdroju,
        ktere jsou relevantni k tomuto deploymentu.
7 spec:
8   replicas: 1  # Deklaruje, ze bude vzdy bezet pouze jedna
      replika.
9   selector:
```

```
10      matchLabels:
11        app: student-app  # Urcuje, ktere pody spadaji pod
              spravu tohoto Deploymentu. Jedna se tedy o~pody,
              ktere maji label app: student-app.
12    strategy:
13      type: Recreate  # Nejjednodussi typ nasazeni, kdy pri
          provedeni operace pro nasazeni se bezici replika
          ukonci, smaze a v zapeti je nahrazena novou.
14    template:
15      metadata:
16        labels:
17          app: student-app  # Oznaceni podu.
18      spec:
19        imagePullSecrets:
20          - name: student-registry  # Definice secret objektu,
              ktery slouzi pro komunikaci se studentovo
              container registry (objekt secret je nastaven pri
              inicializaci).
21        containers:
22          - name: student-application  # Definuje kontejner,
              ktery bezi v~podu.
23            image: ${CI_REGISTRY}/${CI_PROJECT_PATH}/app:${
                APP_TAG}  # Specifikuje nazev image, ktera bezi
                v~podu.
24                 # ${CI_REGISTRY}: URL container registry.
25                 # ${CI_PROJECT_PATH}: Cesta k~projektu.
26                 # ${APP_TAG}: docker image tag.
27            imagePullPolicy: Always  # Nastaveni, aby se pri
                kazdem startu podu stahla ta nejnovejsi docker
                image.
28            ports:
29              - containerPort: 80
```

Pro obalení deploymentu je vytvořena Service, která slouží jako abstrakce, která definuje sadu podů a politiky, podle kterých k nim lze přistupovat. Následující service 8.7 je navržena tak, aby umožňovala komunikaci v rámci clusteru i z externích zdrojů [Gup19].

Zdrojový kód 8.7: Kubernetes Service resource pro aplikace studentů (soubor `service.yaml`)

```
1 apiVersion: v1 # Specifikace Kubernetes API. v1
2 kind: Service # Deklarace, ze nasledujici Kubernetes objekt
    je typu Service
3 metadata:
4   name: student-app # Nastaveni nazvu service
5 spec: # specifikace Service
6   selector:
```

```
7     app: student-app # Tento selector zahrne veskere pody,
          ktere maji stejny label a bude k~nim smerovat sitovy
          provoz. V~tomto pripade zahrne vsechny pody, ktere
          maji label app: student-app
8   type: ClusterIP # Specifikace typ Service. ClusterIP
       prirazuje IP v~ramci clusteru tak, ze ostatni objekty v~
       ramci clusteru k~ni mohou pristupovat (nelze k~teto IP
       pristupovat externe)
9   ports: # konfigurace otevrenych portu
10    - port: 80 # Port, na kterem service naslouchа
11      protocol: TCP # Protokol pouzity pro port
12      targetPort: 80 # Specifikuje port podu, ke kterym
           service forwarduje pozadavky
13      name: http # Nazev konfigurace portu
```

Správné přesměrování na aplikaci konkrétního studenta bude docíleno Kubernetes technologií Ingress, která spravuje externí přístup k službám definovaných k clusteru [Gup19]. Následuje popis nastavení, které směruje požadavky z definované adresy, přičemž pro identifikaci jednotných studentských dashboardů se použije prefix za lomítkem s hodnotou `orionlogin`.

Zdrojový kód 8.8: Kubernetes Ingress resource pro aplikace studentů (soubor `ingress.yaml`)

```
1  apiVersion: networking.k8s.io/v1 # Specifikace verze
      Kubernetes API
2  kind: Ingress # Typ objektu v~Kubernetes
3  metadata:
4    name: student-app # Specifikace nazvu pro Ingress objekt
5  spec:
6    ingressClassName: nginx # Vyuziti ingress controlleru "
       nginx"
7    rules: # Definice pravidel, podle kterych se bude
       presmerovavat provoz
8      - host: oks-web.vyuka.kiv.zcu.cz # Definuje, ze se toto
         pravidlo tyka adresy oks-web.vyuka.kiv.zcu.cz
9        http:
10         paths:
11           - path: /${CI_PROJECT_NAME} # Toto pravidlo se
                spusti, pokud bude v~url adrese za lomitkem naze
                 v~repozitari
12             pathType: Prefix
13             backend:
14               service:
15                 name: student-app # Provoz, na ktery se
                      uplatni definice pravidel vyse je smerovan
                       k~service s~identifikatorem student-app
16                 port:
```

```
17                    number: 80 # Specifikace otevreneho portu
                          service
```

Následuje pouze soubor `kustomization.yaml`, který definuje, které deklarace se mají použít pomocí nástroje Kustomize [JR23]. Následně definuje, které soubory popsané výše budou použity společně. Dále určuje, které resources budou při změně konfigurací případně upraveny. Jedna z několika výhod technologie Kustomize je, že jednotlivé Kubernetes resources jsou rozděleny pro větší přehlednost do několika souborů.

Zdrojový kód 8.9: Kubernetes Kustomization soubor pro aplikace studentů (soubor `kustomization.yaml`)

```
1 resources:
2   - app.yaml
3   - service.yaml
4   - ingress.yaml
```

## 8.7 Integrace Kubernetes a GitLab

Pro zajištění komunikace spravovatelné a řízené komunikace mezi GitLabem a Kubernetes clusterem byla využita možnost GitLabu instalovat Kubernetes Cluster Agenta [Git24c]. Tento agent slouží zejména k zprostředkování komunikace GitLab CI/CD pipeline a Kubernetes clusteru, která je využita při inicializaci studentů a nasazování studentských aplikací. Jeho výhodou je, že za pomoci konfigurace agenta lze přímo omezit manipulaci s Kubernetes clusterem. Na rozdíl od přímého přístupu k API Kubernetes clusteru, agent funguje na základě pull-based modelu. Ten je charakterizován tím, že zvyšuje bezpečnost komunikace tak, že umožňuje clusteru zachovat firewall, který blokuje příchozí spojení a zároveň umožňuje pouze spojení s GitLabem.

### 8.7.1 Instalace Kubernetes Cluster Agenta

Instalace je detailně popsána v dokumentaci GitLabu [Git24c]. Jedná se zkráceně o vytvoření repozitáře na GitLabu, který obsahuje konfiguraci. Dále se pomocí uživatelského rozhraní GitLabu provede registrace agenta s GitLabem. Tato registrace vygeneruje Helm [4] chart, který po při vykonání v Kubernetes clusteru provede deploy agenta do Kubernetes clusteru.

[4] Nástroj, který zjednodušuje instalaci a spravování Kubernetes aplikací. Lze si ho představit jako `apt/yum/homebrew` pro Kubernetes

### 8.7.2 Konfigurace Cluster Agenta

Konfigurace je zajištěna pomocí souboru, který se nachází v repozitáři pro agenta. Tímto souborem lze vymezit, které skupiny, či repozitáře mohou s agentem komunikovat, případně jak [Ahu23]. Pro instanci GitLabu byla zvolena konfigurace, kdy skupina management tools má plná práva a skupina studentů má práva pouze do svých namespace (tj. nemá práva vytvářet namespace). Tím je zajištěno, že studenti budou omezeni pouze na svůj namespace.

# 9 Aktivity pro zpracování a publikování jednotlivých artefaktů

V předchozích kapitolách byly popsány dílčí úlohy semestrální práce z předmětu KIV/OKS a KIV/WEB, dále byl popsán technologický stack, který k těmto úlohám doplní realizační prostředky. Nakonec zbývá popsat, jakým způsobem budou tyto úlohy zpracovávány, jak budou odevzdávány a zkrátka jak probíhá celý proces odevzdávání jednotlivých úloh. Pro úplné zahájení práce na začátku semestru je třeba provést automatickou inicializaci studentských repozitářů tak, aby se pouze přihlásili a mohli rovnou zpracovávat úlohy. Dále je třeba stanovit, že role jednotlivých uživatelů skupiny OKS-WEB budou kopírovat již přednastavené role v systému GitLab a tj. vyučující a správci systému budou v roli Maintainer a větší a studenti, kteří budou zpracovávat semestrální práce bude přidělena role Developer (byl upraven tak, aby mohl publikovat změny do hlavní větve `main`). Následně je třeba navrhnout proces odevzdávání těchto úloh a strukturu pipeline, která bude tyto úlohy zpracovávat (tj. vytvářet z nich artefakty) a publikovat jejich výsledky (artefakty) prostřednictvím již dříve zmiňovaného dashboardu. Těmto bodům bude věnována následující kapitola.

## 9.1 Nastavení prostředí pro studenty – inicializace

Pro nastavení studentů byly zvažovány různé alternativy, které zohledňovaly různou míru samostatnosti. Jedna z nich byla, že bude vytvořen základní projekt, který budou studenti dle instrukcí „forkovat“, čímž by si procvičili i tento úkon. Druhá varianta byla ta, kdy by se veškeré akce provedly automatizovaně na základě pokynu

garanta předmětu KIV/OKS. Druhá varianta je technicky náročnější, avšak díky ní bude eliminováno mnoho rizikových faktorů a snížena práce cvičících pro odstranění případných chyb způsobených díky volnosti studentů. I když bude tato činnost provozována pouze jednou za semestr (tj. na začátku běhu předmětu KIV/OKS), je největší snahou tuto posloupnost akcí automatizovat. Na základě diskuse s garantem předmětu KIV/OKS byla nakonec zvolena.

Inicializace se tedy skládá z několika částí, které jsou standardně vykonávány na základě stisku tlačítka z webové aplikace garanta předmětu KIV/OKS [Hin24] (mohou být vykonány i bez webové aplikace). Tato aplikace byla vypracována součástí jiné diplomové práce a slouží k monitorování průběhu vypracování semestrálních prací studentů a efektivní správy průběhu předmětu KIV/OKS. Tento stisk tlačítka kombinuje volání API systému GitLab z webové aplikace garanta KIV/OKS a činnost repozitáře Kubernetes Namespace Creator dostupného na GitLabu (`oks-web/management-tools/kubernetes-namespace-creator`), který je součástí této diplomové práce. Již zmíněný stisk tlačítka pro inicializaci studentů (studenta) tedy vykoná následující činnosti (aplikace uchovává API token, díky kterému může vykonávat následující činnosti):

1. Vykoná volání API systému GitLab, které provede `fork` repozitáře z `template` projektu pro nového studenta
    - Právě vytvářený repozitář bude privátní, tj. uvidí ho pouze příslušný student a garant předmětu.
    - Do repozitáře se přiřadí příslušný student (pokud ještě nebyl registrovaný, tak se mu na jeho orionlogin vytvoří v systému GitLab účet) s rolí Developer.
2. Vykonání volání API GitLabu pro spuštění pipeline v Kubernetes Namespace Creator, která provede inicializaci pro studenta.
3. Vytvoření přístupu do sdílené databáze a vytvoření databáze pro dashboard studenta (databázi bude využívat aplikace studentů z předmětu KIV/WEB).
    - Tato funkcionalita byla realizována jednoduchým Python scriptem, který využívá balíček `mysql-connector-python` a pomocí SQL příkazů vykoná požadované události.
4. Vytvoření deploy tokenu v repozitáři studenta pro komunikaci s privátním container registry v úložišti studenta.
    - Tato funkcionalita je rovněž realizována pomocí jednoduchého Python skriptu, který využívá API systému GitLab pro vygenerování deploy tokenu (jsou určeny k automatizaci funkcí týkajících se nasazení). Tento

token bude sloužit k vytvoření Kubernetes Secret Resource [1], která bude používána k získání přístupu z podu v příslušném namespace v Kubernetes clusteru do neveřejného container registry u studentských repozitářů. Tato akce je potřeba pro provádění opakované aktualizace studentských dashboardů s výsledky semestrálních prací.

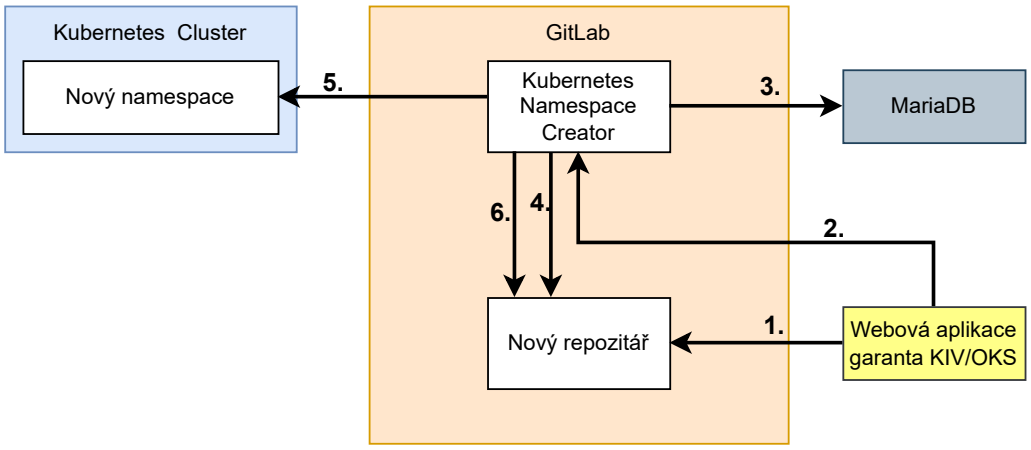
5. Vytvoření Kubernetes namespace s příslušnými oprávněními a Kubernetes Secret Resource.

   - Toto je realizováno stejnými YAML skripty pro definici Kubernetes resources, které byly popsány dříve. Jedná se tedy o skript pro vytvoření namespace, který odpovídá cestě, na které je projekt dostupný (tj. `skupina-rok-orionlogin` a RBAC[2] resource, která namespace omezí tak, aby byl přístupný pouze z projektu, ke kterému náleží [Ahu23].
   - Vytvoření Kubernetes Secret Resource v novém namespace s návem student-registry obsahující dříve zmiňovaný deploy token.

6. Spuštění pipeline pro vytvoření dashboardu v cílovém, nově tvořeném repozitáři studenta.

   - Tato funkcionalita je docílena spuštěním inicializační pipeline (spustí pouze aktivity, které sestaví a nasadí dashboard) v studentském repozitáři přes API systému GitLab.

Celkově lze tedy tento proces vyjádřit zjednodušeně diagramem na následujícím obrázku 9.1 (čísla v seznamu výše se mapují na čísla u jednotlivých šipek v diagramu).:




Obrázek 9.1: Diagram počáteční systémové inicializace studenta

[1] Slouží k uchovávání citlivých údajů.
[2] Role-based autentizace (vytvoří v namespace roli s přístupem `edit` pro repozitář studenta).

Dále je možné inicializaci provést i bez součinosti webové aplikace, kdy bude pořadí operací stejné s tím rozdílem, repozitář pro studenta musí být manuálně vytvořen a student do repozitáře manuálně přiřazen. Dále je nutné manuálně spustit pipeline v Kubernetes Namespace Creator se správně nastavenými parametry, která provede zbývající kroky.

## 9.2 Návrh procesu odevzdávání studentských úloh

Jak již bylo nastíněno, tak odevzdání úloh bude probíhat prostřednictvím systému GitLab, kdy studenti nahrají (provedou operaci `push`) do jejich personalizovaného repozitáře. Nejprve je nutné zvážit, kdy lze práci považovat za odevzdanou. Jak již bylo definováno v kapitole 5.1, tak studenti budou tyto samostatné úlohy odevzdávat v průběhu celého semestru. Pro definici odevzdané úlohy byly tedy zvažovány dvě možnosti. Jako první byla ta, že by studenti nemuseli provádět pro odevzdání úlohy žádnou akci a prostě by provedli push do hlavní větve. Tato možnost byla nakonec zavrhnuta proto, že studenti, kteří budou tyto úlohy odevzdávat, pravděpodobně budou mít malé zkušenosti s verzovacími systémy. Potom by časté spouštění pipeline pro hodnocení (každou operací `git push` do hlavní větve) mohlo vést k velmi vysoké vytíženosti GitLab Runneru.

Další možností se jeví ta, kdy studenti úlohu odevzdají tak, že vytvoří tag. To způsobí jednak to, že studenti provedou „akci pro odevzdání“, čímž svým způsobem završí práci, a druhak to, že stav odevzdaných semestrálních úloh bude vždy jednoznačně zpětně dohledatelný podle vytvořených tagů. Druhá možnost se během testování a diskusí s garantem předmětu KIV/OKS jevila jako velmi vhodná, zejména díky zpětné trasovatelnosti a oddělení odevzdání od běžného zálohování práce na úlohách semestrální práce. Proto byla také zvolena.

### 9.2.1 Návaznost pojmenování tagu na volbu odevzdané úlohy

Jelikož bylo cílem vymyslet takové pojmenování, aby bylo odevzdání co nejjednoznačnější, byla zvolena konvence pojmenování tagu podle úlohy. Jelikož má každá úloha vytvořené kódové označení dle pořadí a zkratky (například `01_UC`, `02_DB`, `...`) byla zavedena konvence, kdy tag musí začínat právě tímto kódovým označením, aby se spustila validace a odevzdání úlohy.

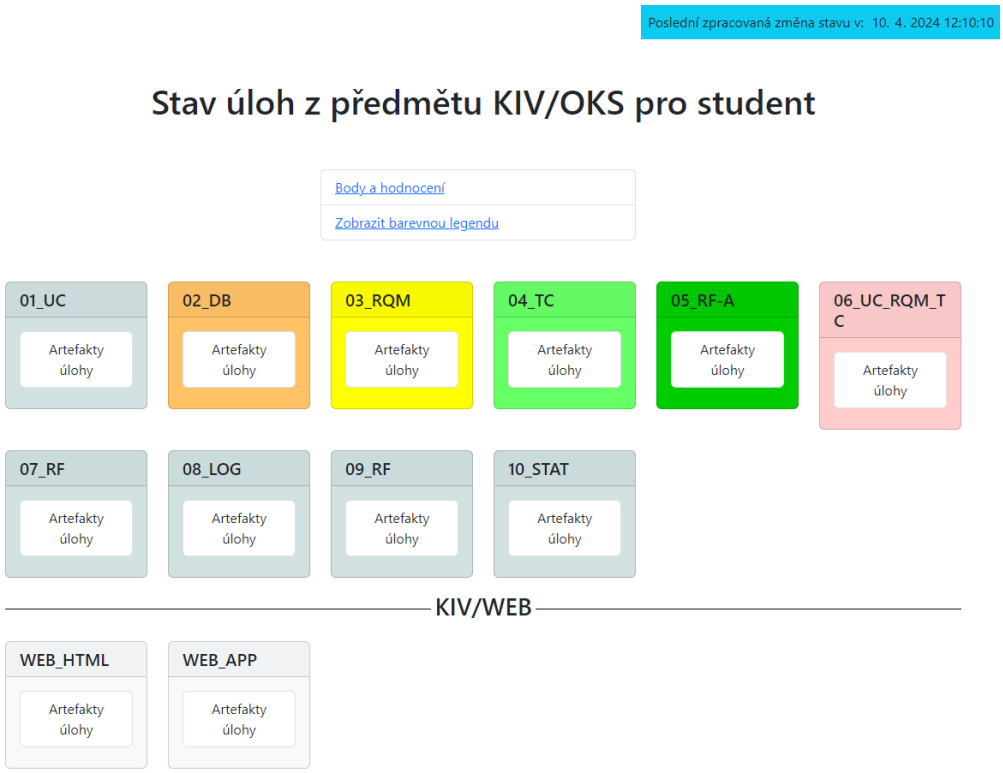
Odevzdání je nastaveno na prefix právě kvůli situaci, kdy odevzdání může selhat (validační skripty odhalí v odevzdání chybu) a díky tomu bude nutné vytvořit druhé odevzdání, tj. nový tag ve tvaru v čele s kódem úlohy končícím libovolným

textem, například `01_UC_oprava` nebo `01_UC_oprava_1`. Toto je nezbytné kvůli funkci nutnosti unikátnosti jména tagu v Gitu [CM23].

## 9.3 Dashboard – návaznost na bodové hodnocení

Jak již bylo několikrát zmiňováno, tak výsledky studentských úloh (artefakty) jsou nasazovány do definovaného Kubernetes prostředí na ukázce 8.6, ve kterém se zobrazí dashboard. Tento dashboard lze libovolně parametrizovat na libovolného studenta, který studuje KIV/OKS a zároveň je v jeho připravené verzi umístěn v docker image pro PHP verze 8.3. Výslednou parametrizaci vykoná při nasazení dashboardu pipeline job pro sestavení aplikace obsahující dashboard a veškeré artefakty, které student kdy vytvořil (získané z GitLabu). Tato parametrizace a nastavení image pro dashboard je zařízena pomocí `Dockerfile`, která je umístěna v repozitáři `oks-web/k8s-definition` v adresáři build.

Na následujícím obrázku 9.2 je zobrazen dashboard studenta s orionlogin student, na kterém jsou obarveny karty jednotlivých úkolů tak, aby odpovídaly všem stavům, které mohou v rámci procesu odevzdávání nastat.




Obrázek 9.2: Obrázek Dashboardu

Dashboard je vytvořen jako jednoduchá HTML stránka s částmi v Javascriptu a skládá se z několika částí. Z informativního rámečku pro zobrazení času poslední zpracované změny stavu hodnocení, dále z odkazů, kdy první otevře nové okno s cílem do studentské části aplikace garanta KIV/OKS, kde se student přihlásí pod svým `orionlogin` a uvidí detailní výsledky bodování své semestrální práce. Druhý odkaz zobrazí barevnou legendu. Nakonec jsou zde karty, které jsou rozděleny do dvou kategorií. Karty pro KIV/OKS se obarvují a jsou v sekci KIV/OKS. Karty pro KIV/WEB se neobarvují a jsou v sekci KIV/WEB. V každé kartě je tlačítko „Artefakty úlohy“, které po kliknutí zpřístupní veškeré artefakty, které vznikly transformací pro danou úlohu.

### 9.3.1 Návaznost na hodnocení

Dashboard je provázaný s hodnocením tak,že webová aplikace garanta KIV/OKS čerpá data z provádění hodnotících pipeline vzniklých při odevzdání úlohy u jednotlivých studentů. Na základě výsledků jobů určí stav úlohy, který se promítne do hodnocení úlohy (to už ale není součástí dashboardu a je přístupné přes přihlášení na „orionlogin“ v sekci body a hodnocení).

Dashboard tedy zobrazuje stavy (v následujícím popisu jsou seřazeny tak, jako na obrázku výše):

- Šedá – Hodnotící pipeline se zatím nespustila.
- Oranžová – Automatická kontrola v pipeline neprošla. Úlohu je třeba opravit a odevzdat znovu
- Žlutá – Automatická kontrola v pipeline prošla, ale úloha byla odevzdána v pozdním termínu.
- Světle zelená – Automatická kontrola v pipeline prošla, ale čeká se na manuální přidělení bodů vyučujícím.
- Zelená – OK, body byly přiděleny.
- Světle červená – Garantem nalezený a studentem dosud neodstraněný PROBLÉM v této úloze.

### 9.3.2 Získávání stavu jednotlivých úloh

Informace o stavech jednotlivých úloh se získávají z API aplikace pro garanta předmětu KIV/OKS. Aplikace má vytvořen GET endpoint `api/v1/evaluations/orionlogin`, kdy `orionlogin` představuje libovolný orionlogin studenta, pro kterého se získávají data o hodnocení. Následuje ukázka 9.1, na které je odpověď výše zmiňovaného endpointu ve tvaru JSON:

Zdrojový kód 9.1: Ukázka odpovědi z aplikace garanta KIV/OKS

```
{
    "tasks": [
        {
            "task_name": "01_UC",
            "task_short_name": "01",
            "html_id": "card-01-uc",
            "status": "evaluation_accepted",
        },
  ...
            ],
    "updated_at": "2024-04-08T04:03:29+02:00"
}
```

Z dat na příkladu 9.1 lze tedy získat informace o stavu jednotlivých úloh (`tasks`) a poslední zpracované změny stavu hodnocení, které je zpracované aplikací garanta KIV/OKS (`updated_at`).

## 9.4 Spouštěné aktivity při odevzdání úlohy

V předchozích kapitolách a sekcích byly popsány veškeré komponenty odevzdávacího procesu, až na jednu, a tou jsou aktivity, kterými se úlohy validují, převádí na artefakty a publikují na studentský dashboard.

### 9.4.1 Typy spouštěných aktivit

Tyto aktivity lze rozdělit na různé typy, a to:

- Validace před akcí (volitelná) – validace odevzdávaných studentských úloh
- Akce – provedení akce, v podstatě cíl každé úlohy (například spuštění testů, spuštění konverze případů užití, vytvoření tabulek pokrytí, …)
- Validace po akci (volitelná) – Provedení určité validace po akci (například kontrola počtu proběhlých testů v RF)
- Vytvoření package – Uchování stavu odevzdané úlohy tj. vytvoření package (artefakt), v package repository příslušného repozitáře která byla spouštěčem pipeline. Proces je takto realizován proto, že s každým „odevzdáním“, tj. spuštění pipeline se musí nasadit do Kubernetes clusteru kompletní Dashboard studenta obsahující veškeré zpracované artefakty.
- Publikování s dashboardem

- Sestavení docker image pro Kubernetes cluster – Při vykonávání jobu `build` se stáhne z repozitáře `k8s-definition` Dockerfile pro Kubernetes Cluster. Následně se stáhnou, pokud existují, i vytvořené artefakty úloh samostatné práce, které se nakopírují do cílových adresářů, ze kterých pak budou přístupné pomocí dashboardu.
- Publikování docker image do Kubernetes Clusteru – Při vykonání jobu `deploy` se stáhne z repozitáře `k8s-definition` deklarace kubernetes prostředí, která se vhodně parametrizuje pomocí nástroje `envsubst` a poté vykoná prostřednictvím Kubernetes Cluster Agenta. Tímto je zajištěno nasazení dashboardu se zpracovanými artefakty.

### 9.4.2 Členění aktivit v pipeline

Jelikož budou aktivity prováděny formou CI/CD pipeline, bylo třeba navrhnout přehledné členění pipeline tak, aby byla na první pohled jasná vazba job – úloha. Toto bylo docíleno tím, že související joby jsou uspořádané do stages, které jsou pojmenované podle kódových označeních jednotlivých úloh tak, jako jsou pojmenované karty na dashboardu a stejně tak musí být pojmenovány tagy, které budou studenti tvořit, aby spustili proces odevzdání úlohy. Soubor s pipeline je tedy dostupný v repozitáři `pipeline_template` v jediném souboru, který obsahuje, a to `jobs.yml`. Tento soubor se pomocí klíčového slova `include` v souboru `.gitlab-ci.yml`, v každém repozitáři studenta, nastaví, jako zdroj konfigurace pro CI/CD pipeline. Tím je zajištěno, že každý student bude mít vždy stejnou konfiguraci pipeline. Pokud by došlo z nějakého důvodu k úpravě, pak by byla provedena na jednom místě a nebylo by třeba manuální aktualizace všech repozitářů.

#### 9.4.2.1 Popis aktivit v pipeline

Následuje krátký soupis veškerých aktivit (`stages`, ve kterých jsou vepsané jednotlivé `jobs`), které jsou vztaženy k určité úloze. Aktivity validačních skriptů nebudou uváděny, jelikož se jedná pouze o jeden job, který je zařazen buď před, nebo po akční část v úloze tzn. že zkontroluje obsah adresáře, či vygenerovaný artefakt. Dále nebude uváděn job pro vytvoření packages (samotných artefaktů), jelikož by byl identický u každé úlohy. Nakonec také nebudou uváděny joby pro sestavení a nasazení docker image s dashboardem a artefakty, jelikož jsou prováděny při odevzdání každé úlohy a již byly popsány výše.

- `01_UC`
  - Konverze případů užití z formátu XML na formát HTML (využití docker image z technologického stacku).

  - Artefakty – konvertované HTML soubory s případy užití.

- `02_DB`
  - Parametrizace souboru při připojení do databáze (adresa a port databáze jsou vytvořeny jako skupinové proměnné, tj. jsou dostupné u každého repozitáře). Jméno, heslo mají všichni studenti nastaven na svůj orion login a název databáze je `db_jejichorionlogin`. Pro studenta s orion-login „herout“ by tedy údaje vypadaly následovně:
    * název databáze – db_herout
    * přihlašovací jméno – herout
    * heslo – herout
  - Plnění vytvořené databáze studentem vytvořenými SQL skripty.
  - Spuštění Robot Framework databázových testů (využití docker image z technologického stacku).
  - Artefakty – Logy proběhlých testů vygenerované Robot Frameworkem.

- `03_RQM`
  - Využití docker image z technologického stacku pro generátor pokrytí pro vygenerování pokrytí pro případy užití v závislosti na požadavcích na software.
  - Artefakt – Vygenerovaný soubor pokrytí.

- `04_TC`
  - Využití docker image z technologického stacku pro generátor pokrytí pro vygenerování pokrytí požadavky v závislosti na testovacích případech a případech užití v závislosti na testovacích případech.
  - Artefakty – Vygenerované soubory pokrytí.

- `05_RF-A`
  - Nejprve nasadí celý dashboard (kvůli statickým HTML stránkám pro plánované spuštění testů).
  - Parametrizace konfiguračního souboru pro testy (nastavení bázové URL, na které bude dostupný dashboard, a která je dostupná v parametrech skupiny na GitLabu).
  - Spuštění automatizovaných testů statických webových stránek napsaných v Robot Framework s využitím Browser Library. Je využita oficiální docker image.

  - Vygenerování souboru pro pokrytí případů užití v závislosti na testovacích případech, které reflektují výsledky testů.
  - Artefakty -– Logy proběhlých testů vygenerované Robot Frameworkem a vygenerovaný soubor pokrytí.

- `06_UC_RQM_TC`
  - Konverze případů užití z XML do HTML a generování všech tří druhů pokrytí (viz dříve).
  - Artefakty – Konvertované případy užití a soubory s tabulkami pokrytí.

- `07_RF`
  - Parametrizace konfiguračního souboru webové aplikace (vytvořena v KIV/WEB) pro připojení k databázi. Funguje na stejně jako u `02_DB`.
  - Nasazení celého dashboardu (kvůli nasazení webové aplikace).
  - Parametrizace konfiguračního souboru pro testy (na stejné bázi jako u `05_RF-A`)
  - Spuštění automatizovaných testů dynamické webové aplikace napsaných v Robot Framework s využitím Browser Library. Je využita oficiální docker image.
  - Vygenerování souboru pro pokrytí případů užití v závislosti na testovacích případech, které reflektují výsledky testů.
  - Artefakty – Logy proběhlých testů vygenerované Robot Frameworkem a vygenerovaný soubor pokrytí.

- `08_LOG`
  - Parametrizace konfiguračního souboru webové aplikace (analogicky jako v `07_RF`)
  - Nasazení celého dashboardu (kvůli nasazení webové aplikace).
  - Spuštění testů z `07_RF` webové aplikace (kvůli simulaci interakce s webovou aplikací).
  - Kontrola, zda se vytvořili požadované logovací soubory.
  - Artefakty – Nejsou.

- `09_RF`
  - Parametrizace konfiguračního souboru webové aplikace (vytvořena v KIV/WEB) pro připojení k databázi. Funguje na stejně jako u `02_DB`.

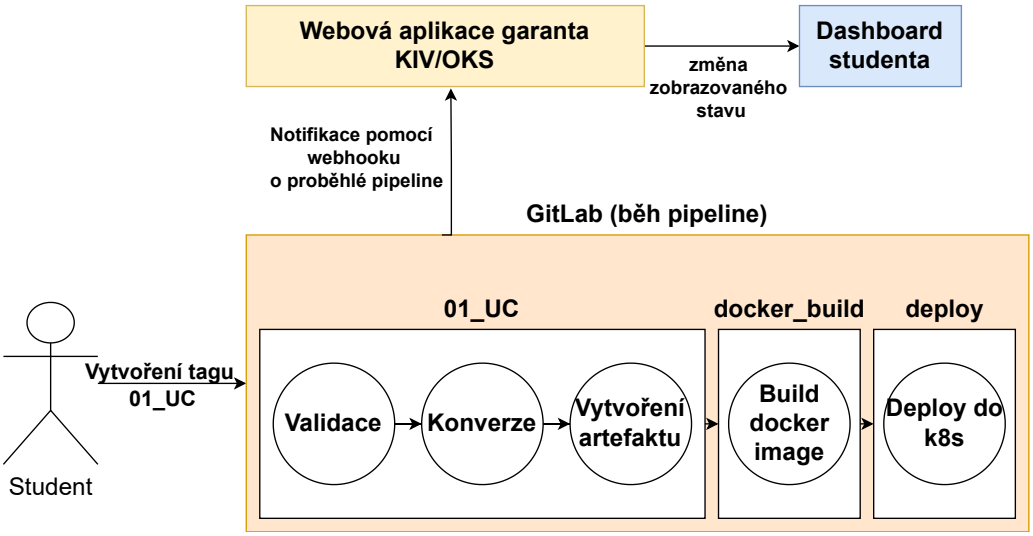
  - Nasazení celého dashboardu (kvůli nasazení webové aplikace).
  - Parametrizace konfiguračního souboru pro testy (na stejné bázi jako u `05_RF-A`)
  - Spuštění automatizovaných testů dynamické webové aplikace napsaných v Robot Framework s využitím Browser Library. Je využita oficiální docker image.
  - Vygenerování souboru pro pokrytí případů užití v závislosti na testovacích případech, které reflektují výsledky testů.
  - Artefakty – Logy proběhlých testů vygenerované Robot Frameworkem a vygenerovaný soubor pokrytí.

- `10_STAT`
  - Pro webovou aplikaci se spustí statická kontrola PHPstan[3], která má různé úrovně komplexity.
  - Spustí se job, který nad kódem aplikace spustí PHPstan úrovně 3.
  - Spustí se job, který nad kódem aplikace spustí PHPstan úrovně 5.

[3]Nástroj pro statickou analýzu pro jazyk PHP, která analyzuje dodržování striktního typování v PHP.

## 9.5 Celistvý pohled na systém

Pokud bychom vzali v potaz typický příklad, jak student použije systém při první úloze, tak poté co si repozitář studenti naklonují (více v uživatelské dokumentaci A.2) a zpracují první úlohu, bude práce s repozitářem vypadat následovně.




Obrázek 9.3: Znázornění celistvého pohledu na systém

Na obrázku 9.3 je vidět akce studenta, vytvoří TAG (přes UI v GitLabu viz uživatelská dokumentace A.4) s názvem `01_UC.` , dále spuštění všech příslušných jobů v pipeline a následnou notifikaci garantské aplikace pomocí skupinového web hooku [4] společně s zobrazením vazby garantská aplikace -> studentský dashboard.

[4] HTTP callback spuštěný událostmi ohledně pipeline ve skupině OKS-WEB.

# Ověření funkčnosti celého systému

# 10

Díky charakteru tohoto systému nebylo možné využít z většiny automatizované testy. Většina práce je totiž nastavování systému GitLab a ostatní systémy jsou pouze uživatelsky využívány, jelikož jejich poskytovatel je CIV. Proto byl celý systém ověřován pouze manuálními scénářovými testy, až na jednu výjimku, a to jsou validační skripty.

## 10.1 Ověřování funkčnosti validačních skriptů

Pro validační skripty byla připravena infrastruktura pro systémové testy, která byla popsána v kapitole 8.4.1. Jedná se o způsob testování, kdy je skript spouštěn tak, jako bude spouštěn v pipeline u jednotlivých studentů, na takových datasetech, aby z většiny reflektovaly stavy semestrálních prací, které mohou studenti odevzdávat. Byl kladen důraz na to, aby validační skripty pokryly nejvíce „nevalidních odevzdání“. Pro zajištění shodnosti prostředí jsou tyto testy spouštěny v pipeline (v repozitáři validačních skriptů) tak, jak budou spouštěny v pipeline u jednotlivých studentů. vždy při jejich úpravě a vydání nové docker image s validačními skripty.

## 10.2 Ověření funkčnosti celého procesu

Ověřování funkčnosti celého systému bylo rozděleno do dvou spolu logicky souvisejících celků. První je Inicializace studentů, kdy byl kladen důraz na správné provedení inicializace, přičemž byl kladen důraz i na pořadí kroků, ve kterých se dané části vytváří. Jako další část byla simulace práce studenta, který vyvíjí a pravidelně odevzdává svou semestrální práci.

Pro simulaci práce studentů byla zřízena hostitelská testovací konta do systému ORION, pomocí kterých byla práce testována. Jednalo se tedy o konta pro pět fiktivních studentů, kdy jejich údaje nebudou v této práci pro větší bezpečnost uvedeny.

## 10.2.1 Ověření inicializace studentů

Ověřování této funkcionality bylo rozděleno do dvou částí. První možností bylo provedení inicializace v součinosti s webovou aplikací garanta KIV/OKS.

### 10.2.1.1 Ověření inicializace v součinosti s webovou aplikací Garanta KIV/OKS

**Akční kroky:**

1. Přihlášení do garantské aplikace
2. Vytvoření skupiny pro ročník v aplikaci (pokud už skupina existuje, je tento krok volitelný).
3. Import fiktivních studentů do této skupiny
4. Provedení inicializace (stisk tlačítka pro inicializaci v sekci „STUDENT“)

**Verifikační kroky:**

1. Ověření viditelnosti repozitáře u jednotlivých fiktivních uživatelů (fiktivní studenti)
2. Ověření existence příslušných databázových kont
3. Ověření existence příslušných Kubernetes namespaces
4. Ověření, že si studenti nemohou navzájem manipulovat se svými repozitáři
    - Je zajištěno jednak viditelností repozitářů – jsou privátní, tak nastavením kubernetes namespace, které jsou přístupné pouze z repozitářů daných studentů.
5. Ověření schopnosti vytvářet Tagy a „pushovat“ nové změny do repozitáře

### 10.2.1.2 Ověření inicializace bez součnosti webové aplikace garanta KIV/OKS

**Akční kroky:**

1. Přihlášení do systému GitLab s právy ve skupině OKS-WEB Maintainer a větší (jsou určeny pouze pro vyučující a správce systému).
2. Manuální `fork` repozitáře `project_template` a přiřazení příslušného fiktivního studenta do právě vytvořeného personalizovaného repozitáře.

3. Spuštění pipeline v repozitáři Kubernetes Namespace Creator s vhodně nastavenými běhovými proměnnými (ID právě vytvořeného projektu, orion-login studenta, cílová „cesta projektu“ tj. obsah předdefinované proměnné `CI_PROJECT_PATH_SLUG` u nově vytvořeného repozitáře a personal access token uživatele, který inicializaci spouští)

**Verifikační kroky:**

1. Ověření viditelnosti repozitáře u jednotlivých fiktivních uživatelů (fiktivní studenti)
2. Ověření existence příslušných databázových kont
3. Ověření existence příslušných Kubernetes namespaces
4. Ověření, že si studenti nemohou navzájem manipulovat se svými repozitáři
    - Je zajištěno jednak viditelností repozitářů – jsou privátní, tak nastavením kubernetes namespace, které jsou přístupné pouze z repozitářů daných studentů.
5. Ověření schopnosti vytvářet Tagy a „pushovat“ nové změny do repozitáře

### 10.2.2 Ověření celého systému během práce fiktivního studenta

Pro účely tohoto druhu scénářových testů byl určen jeden repozitář s názvem `herout` v testovacím ročníku „2023“. Pro tento repozitář bylo vytvořeno řešení veškerých odevzdávaných úloh, které byly odevzdávány v pořadí, ve kterém budou později reálnými studenty odevzdávány.
Jediný scénář, který byl v tomto repozitáři použit je:
**Akční kroky:**

- Vyřešení dané úlohy v daném adresáři dříve specifikovaného repozitáře
- Push změn do repozitáře
- Vytvoření Tagu

**Verifikační kroky:**

- Ověření, zda se pipeline chová dle požadavků k danému úkolu.
- Ověření, zda se artefakt ze zpracovávané úlohy (pokud prošel pipeline) zpřístupnil pomocí dashboardu tj. Ověření jobu v pipeline i dashboardu ve webovém prohlížeči.

Během postupného vytváření jednotlivých samostatných úloh tedy bylo vytvořeno:

- 14 případů užití v rámci `01_UC`
- 16 databázových testů v rámci `01_DB`
- 17 požadavků na software v rámci `03_RQM`
- 49 testovacích případů v rámci `04_TC`
- statické HTML stránky jednoduché webové aplikace49 Robot framework testů statických html stránek v rámci `05_RF-A`
- 53 konkretizovaných požadavků a 85 konkretizovaných testovacích případů a 14 konkretizovaných případů užití v rámci `06_UC_RQM_TC`
- 73 Robot Framework testů v rámci `07_RF`
- nastavené logování v jednoduché webové aplikaci v rámci `08_LOG`
- 85 konkretizovaných Robot Framework testů v rámci `09_RF`
- aplikace pravidel PHPStan statické kontroly na již vytvořenou PHP aplikaci z KIV/WEB

# Závěr 11

Tato diplomová práce se zaměřila na návrh a implementaci efektivního technologického stacku pro výuku předmětů KIV/WEB a KIV/OKS na Západočeské univerzitě v Plzni. Hlavním cílem bylo vytvořit komplexní, uživatelsky přívětivé a technologicky pokročilé prostředí, které by usnadnilo studentům práci na samostatných úlohách a zároveň zlepšilo interakci studentů s moderními vývojovými nástroji a připravilo je na výzvy reálného IT prostředí.

Integrace moderních vývojových nástrojů a procesů do akademického prostředí poskytuje studentům unikátní příležitost naučit se pracovat s nástroji, které jsou v průmyslu běžně používány. Tato zkušenost studentům může poskytnout výhodu při hledání zaměstnání po absolvování univerzity, jelikož již budou obeznámeni s praktickými aspekty a nástroji, které se v praxi běžně využívají.

V průběhu práce byl proveden důkladný výběr a následně i integrace různých technologií. Hlavní prvky technologického stacku zahrnovaly GitLab pro verzování a CI/CD procesy, Docker pro kontejnerizaci aplikací a pomocných nástrojů a Kubernetes pro jejich orchestraci a prostředí pro studentské dashboardy. Tyto nástroje umožnily automatizaci mnoha aspektů vývoje a testování softwaru, což přispěje k efektivitě a snížení časových nároků na správu předmětů. Pro zlepšení uživatelského zážitku byla využita technologie `make`, pro automatizaci rutinních vývojových činností, která značně zjednoduší manipulaci s technologickým stackem a zároveň zrychlí práci na samostatných úlohách.

Ke zvýšení připravenosti studentů na používání technologického stacku byla vytvořena komplexní dokumentace. Tato dokumentace slouží jako kompletní návod k využívání veškerých aspektů technologického stacku během zpracování samostatných úloh.

Ověření realizovaného technologického stacku bylo provedeno prostřednictvím rozsáhlé sady scénářových testů, které pokrývaly různé situace, s jakými se studenti mohou setkat při zpracovávání samostatných úloh. Tyto testy byly navrženy tak, aby ověřily funkčnost celého technologického stacku.

Je plánováno, že tento technologický stack bude plně začleněn do výuky předmětů KIV/OKS a KIV/WEB již v zimním semestru roku 2024.

# Uživatelská dokumentace pro studenty


## A.1 Instalace potřebného software

Pro operační systém Windows, je třeba stáhnout WSL2 (Windows Subsystem for Linux). V případě Linuxu tento krok vynecháte.

- Pokud máte Windows 10 verzi 2004 a vyšší (build 19041 a vyšší)
  - Instalujte podle návodu dostupného na:
    `https://learn.microsoft.com/en-us/windows/wsl/install`
- Pokud máte nižší verzi Windows
  - Instalujte podle návodu dostupného na:
    `https://learn.microsoft.com/en-us/windows/wsl/install-manual`

Pro schopnost práce v systému je třeba mít nainstalován následující software.

### A.1.1 Docker

Docker je třeba stáhnout z oficiálních stránek. Přejdeme na https://docs.docker.com/get-docker/ . Na této adrese je třeba zvolit, pro jaký operační systém je instalace prováděna. Po výběru operačního systému lze následovat instrukce, na které budete odkázáni.

Pokud používáte operační systém Windows, je třeba mít nainstalované WSL2. Je třeba zvolit instalaci dockeru, u které je uvedený WSL2 Backend, který má lepší výkonost. Je vhodné si společně s Dockerem nainstalovat i program Docker Desktop.

### A.1.2 Vývojové IDE

Bude využíván PHPStorm (IDE pro PHP), PyCharm (IDE pro Python), pro které mají všichni studenti dostupné zdarma výukové licence. IDE je možné stáhnout z oficiálních stránek.

- PHPStorm – `https://www.jetbrains.com/phpstorm/`
- PyCharm – `https://www.jetbrains.com/pycharm/`

### A.1.3 Program make

Program `make` je třeba nainstalovat pro zjednodušení vykonávání příkazů, které jsou potřeba pro efektivní práci s technologickým stackem.

- Linux – `https://www.gnu.org/software/make/#download`
- Windows – `https://gnuwin32.sourceforge.net/packages/make.htm`

### A.1.4 GIT

Dále je třeba stáhnout a nainstalovat GIT, jelikož bude hlavním nástrojem pro komunikaci a odevzdávání semestrálních prací.

- `https://git-scm.com/downloads`

## A.2 Nastavení prostředí pro vývoj

1. Přiřadit SSH klíč [1] ke svému GitLab účtu. Pokud SSH klíč nemáte, je třeba ho vygenerovat příkazem

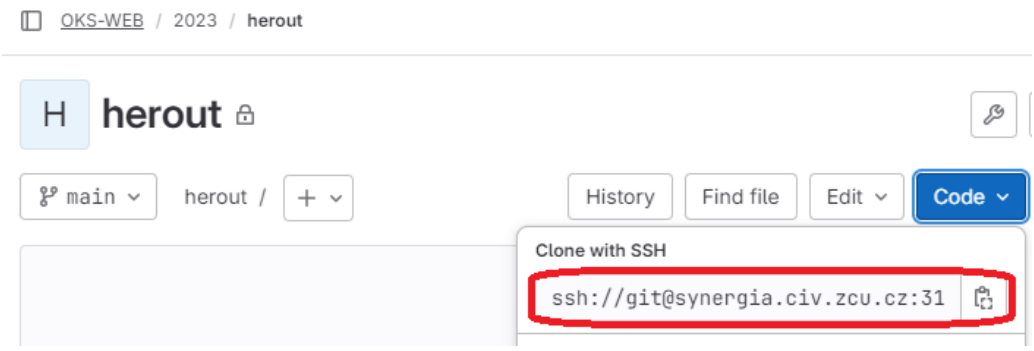
   ```
   ssh-keygen -t ed25519 -C "vasEmail@students.zcu.cz"
   ```

   Pro jednoduchost nechte heslo prázdné. Takto vygenerovaný pár klíčů se uloží do vašeho domovského adresáře v podadresáři `.ssh`

   - Na GitLabu v nastavení účtu (Preferences) zvolit SSH Keys a zde přidat nový SSH klíč (Add new key) a následně vložit veřejný klíč ( s koncovkou `.pub`)

2. Nalézt personalisovaný GitLab repozitář pojmenovaný dle <`orion login`>. Nachází se ve skupině OKS-WEB v podskupině pojmenované dle aktuálního roku.

3. Naklonovat repozitář příkazem `git clone` <`adresa`>. Místo <`adresa`> umístěte do příkazu adresu, která je dostupná v personalizovaném repozitáři v modrém tlačítku „Code“, jak je vidět na obrázku A.1. Po rozbalení rozbalovací nabídky zkopírovat adresu pod textem „Clone with SSH“.




Obrázek A.1: Získání adresy pro `git clone`

[1] Pro větší detaily lze nahlédnout do návodu z oficiálního zdroje na následujícím odkazu `https://docs.gitlab.com/ee/user/ssh.html`

## A.3 Realizace samostatných úloh semestrální práce

### A.3.1 Správná volba adresáře pro úlohu

- Dle zadání vyberte v lokálně naklonovaném vlastním repozitáři adresář, který je pro zpracování úlohy určen
- Je vhodné si výsledky zálohovat do GitLabu, tj. pravidelně provádět operace `git commit` a `git push`
- Můžete libovolně používat větvení (branch), ale pro odevzdání je relevantní pouze hlavní větev `main`

### A.3.2 Technologický stack pro práci na úlohách

Manipulace s technologickým stackem je zpřístupněna přes `Makefile`, který je vždy umístěn na dvou místech.

- Veškeré akce se spouští `make <nazev akce> + případné parametry`
- Pro KIV/OKS – v kořenovém adresáři studentského projektu
  - Obsahuje nutné akce pro zpracování veškerých samostaných úloh.
  - Akce jsou pojmenovány dle kódových označení úloh (s rozdílem, že kvůli funkci `make` jsou prohozené pořadové číslo a typ, který je malými písmeny, proto zde neexistuje cíl např. `01_UC`, ale existuje `uc_01`.
  - Tyto akce využívají lokální stav v adresářích repozitáře a provádí veškeré akce, které jsou potřeba ke zpracování samostatných úloh.

    Pokud se u akce objeví symbol **`?`**, znamená to, že je určena pouze pro ladění a vývoj a po odevzdání je provedena automaticky. Některé cíle je třeba spustit s parametry příkazové řádky. Tyto parametry jsou uvedeny a popsány přímo v `Makefile` a zde nebudou uvedeny.

    Jedná se o akce s názvy:

    * `uc_01` – **`?`** – Provede transformaci UC z XML na HTML.
    * `db_02-tests` – Spuštění databázových testů.
    * `db_02-resetDB` – Uvede DB do stavu vycházejícího ze skriptů v adresáři pro testy.
    * `rqm_03` – **`?`** – Provede konverzi RQM souborů a generuje pokrytí v HTML (s parametry).

    * `tc_04` – **?** – Spustí konverzi testovacích případů (TC) a generuje pokrytí v HTML (s parametry).
    * `tc_04-robot` – **?** – Spustí konverzi TC s integrací výsledků Robot Framework (s parametry).
    * `rf-a_05` – Spustí všechny Robot Framework testy v určeném adresáři (s parametry).
    * `uc_06` – **?** – Spustí konverzi UC pro nový adresář.
    * `rqm_06` – **?** – Provede konverzi RQM pro UC v novém adresáři a generuje pokrytí (s parametry).
    * `tc_06` – **?** –Spustí konverzi TC pro nový adresář a generuje pokrytí (s parametry).
    * `tc_06-robot` – **?** –Spustí konverzi TC s integrací výsledků Robot Framework pro nový adresář (s parametry).
    * `rf` – Spustí Robot Framework testy (s parametry).
    * `squash` – Spustí server SquashTM.
    * `squash-stop` – Zastaví server SquashTM.

- Pro KIV/WEB – v adresáři s webovou aplikací
- Obsahuje akce pro manipulaci s dockerem, kterým lze spustit Apache s PHP 8.3, MariaDB, PHPMyAdmin a kontejner obsahující NPM.

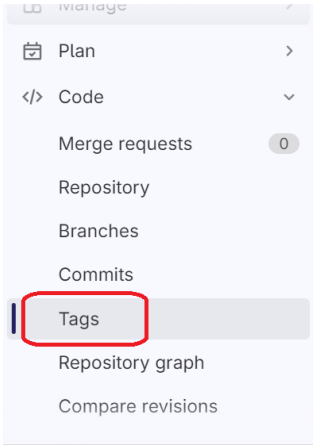
  Jedná se o akce s názvy:

  – `up` – Spustí aplikaci tím, že nastartuje Docker kontejner.
  – `down` – Vypne Docker kontejner.
  – `down-rmi` – Vypne Docker kontejner a smaže všechny jeho data.
  – `restart` – Restartuje Docker kontejner.
  – `install` – Spustí `composer install` v PHP kontejneru.
  – `phpstan-min` – Spustí `phpstan` pro splnění minimálních kritérií v PHP kontejneru.
  – `phpstan-dop` – Spustí `phpstan` pro splnění doporučených kritérií v PHP kontejneru.
  – `phpstan-max` – Spustí `phpstan` s maximální úrovní pravidel v PHP kontejneru.
  – `sh` – Spustí bash shell v PHP kontejneru.
  – `node-install` – Spustí `npm install` v Node kontejneru.
  – `node-sh` – Spustí shell v Node kontejneru.

## A.4 Odevzdání samostatné úlohy

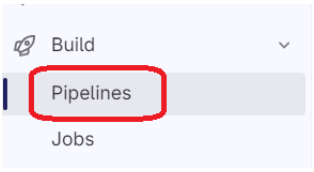
Pokud uznáte, že je vhodný okamžik úlohu odevzdat (pozor na termín odevzdání), je třeba dodržet následující postup.

1. Nahrát finální stav do repozitáře (`git commit`, `git push`) do větve `main`
2. Vytvořit TAG, který spustí odevzdávací proces. Na GitLabu v levé nabídce vlastního repozitáře v sekci „Code“ klikneme na „Tags“ (k vidění na obrázku A.2) a v pravém horním rohu klikneme na „New Tag“. Název vytvářeného tagu musí začínat kódovým označením úlohy (shodné s názvem kartičky na dashboardu, např. `01_UC`), případně k dispozici v zadání dané úlohy.




Obrázek A.2: Lokace sekce pro vytvoření tagu

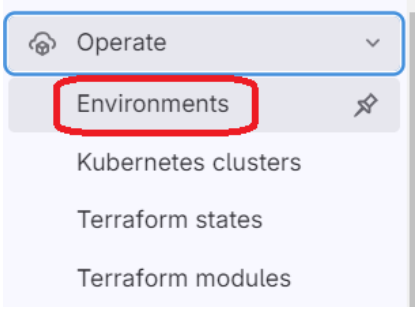
3. Zkontrolovat stav právě spuštěné CI/CD pipeline (v levém menu „Build“, kdy se po rozbalení otevře „Pipelines“ viz obrázek A.3, která se spustila po vytvoření TAGu (všechny joby musí být úspěšné, tzn. „veškerá kolečka“ by měla být obarvena zeleně A.4. Pokud některé svítí červeně, je třeba zkontrolovat log jobu a dle informací opravit úlohu a vytvořit nový tag stejným způsobem, jako v 2. bodu.



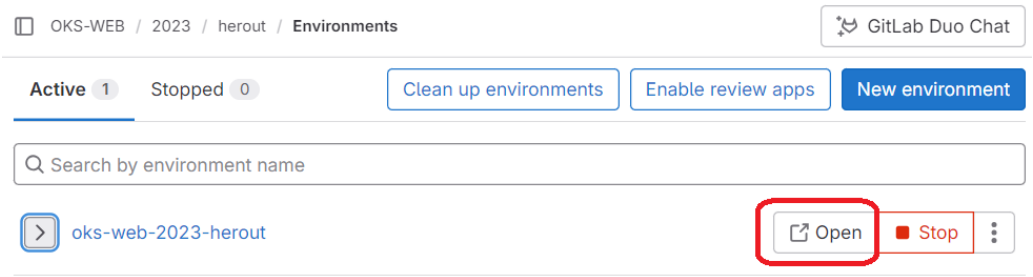

Obrázek A.3: Zobrazení CI/CD Pipelines

Obrázek A.4: Úspěšná pipeline

4. Zkontrolujete, zda se artefakty ze zpracované úlohy úspěšně propagovaly na dashboard, na který se lze jednoduše prokliknout po úspěšné pipeline (proběhl job `build, deploy`) přes levé menu a položku „Operate“ a proklik na „Environments“ viz obrázek A.5. Zde se otevře seznam s jedním prostředím obsahující Dashboard, na který se lze jednoduše prokliknout po stisku tlačítka dle obrázku A.6.




Obrázek A.5: Lokace sekce pro prostředí




Obrázek A.6: Lokace prokliku z GitLabu na dashboard.

5. Vyčkat na obarvení karty na dashboardu (dojde k němu po zpracování změny stavu aplikací), případně kontrola sekce body a hodnocení na dashboardu (je nutné se přihlásit přes jednotný systém přihlašování pomocí účtu ORION).

# Struktura elektronické přílohy

B

- /
  - `Aplikace_a_knihovny` ..... Kopíruje navrženou strukturu na GitLab
    - `OKS-WEB` ..... Bazový adresář vymezující předmět KIV/OKS a KIV/WEB
      - `2023` ..... Testovací ročník obsahující testovací studentské projekty
        - `empty` ..... Prázdný projekt
        - `herout` ..... Dokončený projekt
      - `images` ..... Adresář s vytvořenými Docker images
        - `java` ..... Image s Javou – zde jen konvertor UC
        - `kubectl-envsubst` ..... Image pro kubectl s envsubst
        - `node` ..... Image pro Node obsahující npm pro KIV/WEB
        - `php-apache` ..... Docker image pro Apache s PHP 8.3
        - `python` . Image s Python – zde např. program na generování pokrytí
        - `robot-framework` Image pro Robot Framework – zde jen Database Library
        - `validation-scripts` ..... Image pro validační skripty
      - `k8s-definition` ..... Obsahuje struktury pro k8s
      - `management-tools`
        - `kubernetes-namespace-creator` ..... Projekt pro inicializaci prostředí jednotlivých studentů
      - `oks-web-kca` ..... Konfigurace GitLab Cluster Agenta
      - `pipeline_template` ..... Projekt obsahující pipeline pro studentské projekty
      - `project_template` ..... Bazový studentský projekt, který je forkován
  - `Poster` ..... Adresář, který obsahuje Poster
    - `poster.pdf`
    - `poster.pub`
  - `Text_prace` . Adresář, který obsahuje text práce (`.pdf` soubory i LaTeXsoubory)

# Bibliografie


[Ahu23] AHUJA, Rohit Anand. *Securing the End Points of Microservices using Gitlab Client-Based Authentication* [online]. 2023. [cit. 2024-04-30]. Dostupné z: https://norma.ncirl.ie/id/eprint/6505. Dis. pr. Dublin, National College of Ireland.

[Atl21] ATLASSIAN. *Atlassian Documentation* [online]. 2021. [cit. 2024-02-25]. Dostupné z: https://confluence.atlassian.com/bitbucket/bitbucket-pipelines-792496469.html.

[Atl24a] ATLASSIAN. *Bitbucket* [online]. 2024. [cit. 2024-05-01]. Dostupné z: https://bitbucket.org/product.

[Atl24b] ATLASSIAN. *Continuous integration vs. delivery vs. deployment* [online]. 2024. [cit. 2024-02-11]. Dostupné z: https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment.

[Azu24] AZURE, Microsoft. *Azure DevOps Services | Microsoft Azure* [online]. 2024. [cit. 2024-02-11]. Dostupné z: https://azure.microsoft.com/en-us/products/devops.

[Bad23] BADER, Sarah. *GitHub vs. Bitbucket vs. GitLab: Which one is right for your dev team? - Rewind* [online]. 2023. [cit. 2024-04-25]. Dostupné z: https://rewind.com/blog/github-vs-bitbucket-vs-gitlab-comparison/.

[Dha22] DHADUK, Hiren. *Jenkins vs CircleCI: Which is the best CI/CD tools - Hiren Dhaduk - Medium* [online]. Medium, 2022 [cit. 2024-04-15]. Dostupné z: https://medium.com/@HirenDhaduk1/jenkins-vs-circleci-which-is-the-best-ci-cd-tools-558bbe447ccc.

[Doc24a] DOCKER. *Docker Compose overview* [online]. 2024. [cit. 2024-04-27]. Dostupné z: https://docs.docker.com/compose/.

[Doc24b] DOCKER. *Dockerfile reference* [online]. 2024. [cit. 2024-03-31]. Dostupné z: https://docs.docker.com/reference/dockerfile/.

[Fow90] FOWLER, Glenn. A case for make. *Software: Practice and Experience*. 1990, roč. 20, č. S1, S35–S46. Dostupné z DOI: https://doi.org/10.1002/spe.4380201305.

[Git24a] GITHUB. *GitHub Actions documentation - GitHub Docs* [online]. 2024. [cit. 2024-04-26]. Dostupné z: https://docs.github.com/en/actions.

[Git24b] GITHUB. *Understanding GitHub Actions - GitHub Docs* [online]. 2024. [cit. 2024-02-13]. Dostupné z: https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions.

[Git21] GITLAB. *Unify the entire DevSecOps lifecycle with GitLab* [online]. 2021. [cit. 2024-03-31]. Dostupné z: https://about.gitlab.com/stages-devops-lifecycle.

[Git22] GITLAB. *What is CI/CD? | GitLab* [online]. GitLab, 2022 [cit. 2024-02-12]. Dostupné z: https://about.gitlab.com/topics/ci-cd/.

[Git24c] GITLAB. *Installing the agent for Kubernetes | GitLab* [online]. 2024. [cit. 2024-05-01]. Dostupné z: https://docs.gitlab.com/ee/user/clusters/agent/install/index.html.

[Gup19] GUPTA, Arun. Kubernetes. *DZone* [online]. 2019, roč. 24, s. 2019 [cit. 2024-04-30]. Dostupné z: https://dzone.com/storage/attachments/14131598-dzone-kubernetes-bundle.pdf.

[Hat20] HAT, Red. *Red Hat OpenShift | Red Hat Developer* [online]. 2020. [cit. 2024-04-27]. Dostupné z: https://developers.redhat.com/products/openshift/overview.

[Hat24] HAT, Red. *What is CI/CD?* [online]. 2024. [cit. 2024-02-02]. Dostupné z: https://www.redhat.com/en/topics/devops/what-is-ci-cd.

[Hed20] HEDDINGS, Anthony. *What Does Docker Do, and When Should You Use It?* [online]. How-To Geek, 2020 [cit. 2024-04-21]. Dostupné z: https://www.howtogeek.com/devops/what-does-docker-do-and-when-should-you-use-it/.

[HD20] HENSCHEL, Jack; DI, Mario. *A comparison study of managed CI/CD solutions* [online]. 2020. [cit. 2024-03-31]. Dostupné z: https://git.cubieserver.de/jh/cs-e4000-seminar/raw/commit/a9b2cba6ec2c187f7a9b34c1f4a9ab8dc5097f71/cs-seminar_2020-04-11.pdf.

[Hin24] HINTERHOLZINGER, Jan. *Automatická evaluace výsledků samostatných prací v předmětech WEB a OKS*. Plzeň, 2024. Diplomová práce. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky. Vedoucí práce Ph.D. DOC. ING. PAVEL HEROUT.

[CM23] CHOI, Brendan; MEDINA, Erwin. Learning Ansible Basic Concepts III: Git, Tags, Managing Files, and Services. In: *Introduction to Ansible Network Automation: A Practical Primer*. Berkeley, CA: Apress, 2023, s. 403–457. ISBN 978-1-4842-9624-0. Dostupné z DOI: 10.1007/978-1-4842-9624-0_9.

[IBM24a] IBM. *What Is Containerization? | IBM* [online]. 2024. [cit. 2024-03-22]. Dostupné z: https://www.ibm.com/topics/containerization.

[IBM24b] IBM. *What Is Continuous Deployment? | IBM* [online]. 2024. [cit. 2024-02-28]. Dostupné z: https://www.ibm.com/topics/continuous-deployment.

[Jen24] JENKINS. *Jenkins User Documentation* [online]. 2024. [cit. 2024-04-25]. Dostupné z: https://www.jenkins.io/doc/.

[JR23] JOSHI, Divyansh; RAWAT, Hari Singh. Automating Deployment of Various Microservices Kubernetes. 2023. Dostupné také z: http://www.ir.juit.ac.in:8080/jspui/handle/123456789/9826.

[Kha24] KHARLANTSEVA, Maria. *What Is Github: Main Features, Benefits, Pricing [2024]* [online]. 2024. [cit. 2024-04-30]. Dostupné z: https://everhour.com/blog/what-is-github/.

[Kub23] KUBERNETES.IO. *Kubernetes.io* [online]. 2023. [cit. 2024-03-24]. Dostupné z: https://kubernetes.io/docs/concepts/overview/.

[Mic23] MICROSOFT. *Containers vs. virtual machines* [online]. 2023. [cit. 2024-03-28]. Dostupné z: https://learn.microsoft.com/en-us/virtualization/windowscontainers/about/containers-vs-vm.

[Ort20] ORTEGA, Rodolfo. *Best CI/CD Tools for DevOps: A Review of the Top 10* [online]. 2020. [cit. 2024-01-29]. Dostupné z: https://bluelight.co/blog/best-ci-cd-tools.

[Pat18] PATIL, Sachin. *What is a Makefile and how does it work?* [online]. 2018. [cit. 2024-04-28]. Dostupné z: https://opensource.com/article/18/8/what-how-makefile.

[Ren15] RENSIN, David. *Kubernetes*. O'Reilly Media, Incorporated, 2015. ISBN 9781491931875.

[Vas15] VASE, Tuomas. Advantages of Docker. *Jyx.jyu.fi*. 2015. Dostupné z DOI: https://jyx.jyu.fi/handle/123456789/48029.

[Wil22] WILSON, Bibin. *Podman Tutorial For Beginners: Step by Step Guides* [online]. 2022. [cit. 2024-03-17]. Dostupné z: https://devopscube.com/podman-tutorial-beginners/.

# Seznam zkratek

**API** Aplication Programming Interface.

**CD** Continuous Deployment.

**CI** Continuous Integration.

**CIV** Centrum informatizace a výpočetní techniky.

**CRUD** Create, Read, Update, Delete.

**DB1** Databázové systémy 1.

**GIT** distribuovaný systém pro správu verzí.

**HTML** Hypertext Markup Language.

**IDE** Integrated Development Environment.

**IS/STAG** Informační systém určený pro administraci studijní agendy vysoké školy.

**IT** Information Technology.

**JSON** JavaScript Object Notation.

**KIV** Katedra Informatiky a Výpočetní techniky.

**NPM** správce balíčků pro javascript.

**OKS** Ověřování Kvality Software.

**PHP** Hypertext Preprocessor.

**RBAC** Role-based Access Control.

**RF** Robot Framework.

**RFBL** Robot Framework Browser Library.

**RQM** Requirement.

**S2I** Source to Image.

**SAST** Static Aplication Security Testing.

**SCRUM** Iterativní a inkrementální způsob řízení vývoje softwaru.

**SQL** Structured Query Language.

**SSH** Secure Shell.

**SSO** Single Sign-on.

**SW** Software.

**TC** Test case.

**UC** Use Case.

**UI** User Interface.

**URL** Uniform Resource Locator.

**VCS** Version Control System.

**VM** Virtual Machine.

**VPN** Virtual Private Network.

**WEB** Webové aplikace.

**WSL** Windows Subsystem for Linux.

**XML** Extensible Markup Language.

**XSLT** Extensible Stylesheet Language Transformations.

**YAML** Ain't Markup Language.

# Seznam obrázků

# Seznam výpisů