FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA INFORMATIKY
A VÝPOČETNÍ TECHNIKY

Diplomová práce

Automatická evaluace výsledků samostatných prací v předmětech WEB a OKS

Jan Hinterholzinger

PLZEŇ

2024

FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA INFORMATIKY
A VÝPOČETNÍ TECHNIKY

Diplomová práce

Automatická evaluace výsledků samostatných prací v předmětech WEB a OKS

Bc. Jan Hinterholzinger

Vedoucí práce
Doc. Ing. Pavel Herout, Ph.D.

PLZEŇ **2024**

Citace v seznamu literatury:
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 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. Jan HINTERHOLZINGER**
Osobní číslo:	**A22N0045P**
Studijní program:	**N0613A140040 Softwarové a informační systémy**
Téma práce:	**Automatická evaluace výsledků samostatných prací v předmětech WEB a OKS**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se s validátorem dosud používaným v KIV/OKS, zejména s jeho základními možnostmi validace různých typů úloh. Dále prozkoumejte API systému GitLab s ohledem na využití jeho funkcí pro možnosti aktuálně nově vytvářené validace.
2. Navrhněte aplikaci, která bude využívat informace získávané z GitLabu a bude je interpretovat pro vyučujícího. Ten bude mít dodatečnou možnost korekcí finálního hodnocení artefaktu. Dále navrhněte sadu validačních úloh, které budou využívány v již existujícím technologickém stacku.
3. Realizujte navrženou aplikaci v PHP, přičemž se zaměřte na schopnost efektivně a jednoduše pracovat s jednotlivými hodnoceními.
4. Ověřte funkčnost aplikace na již existujících artefaktech a dále připravte reprezentativní sadu artefaktů pro negativní testy.

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 15. května 2024

. . . . . .

Jan Hinterholzinger

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 vytvoření systému pro automatickou evaluaci studenty odevzdaných semestrálních prací. Součástí této práce je návrh webové aplikace v jazyce PHP sloužící pro podporu výuky předmětu KIV/OKS na FAV ZČU. Aplikace získává data ze systému GitLab a dále je zpracovává a vizualizuje vyučujícímu. Byla vytvořena analýza GitLab API pro využití ve zmíněném systému. Dále byl vytvořen návrh validačních kritérií pro automatickou validaci jednotlivých odevzdávaných úloh.

V praktické části byla navržená aplikace implementována v jazyce PHP ve frameworku Nette dle požadavků zadavatele. Implementace využívá principy čistého kódu a rozšiřitelnosti. Pro ověření funkčnosti aplikace a celého systému byla vytvořena reprezentativní sada artefaktů použitelná v pozitivních i negativních testech.

Abstract

The master's thesis deals with the creation of a system for automatic evaluation of term papers submitted by students. This thesis includes the design of a web application in PHP language used to support the teaching of the subject KIV/OKS on FAV ZČU. The application retrieves data from GitLab and further processes and visualizes it for the teacher. A GitLab analysis of API was created for use in the aforementioned system. Furthermore, a draft of validation criteria for automatic validation of individual submissions was created.

In the practical part, the proposed application was implemented in PHP language in Nette framework according to the requirements of the client. The implementation uses the principles of clean code and extensibility. To verify the functionality of the application and the whole system, a representative set of artifacts was created, usable in both positive and negative tests.

Klíčová slova

PHP • Nette • GitLab • API • MariaDB

Poděkování

Tímto bych chtěl poděkovat doc. Ing. Pavlu Heroutovi, Ph.D. za jeho příkladné vedení této diplomové práce, za poskytnuté informace, předané zkušenosti a cenné rady. Dále bych chtěl vyjádřit vděk své rodině, blízkým a přátelům za jejich podporu v průběhu tvorby této diplomové práce a celého studia. Na závěr bych rád poděkoval firmě RTsoft, s.r.o., která poskytla odborné konzultace a dodávala cenné rady z pohledu praxe.

Jan Hinterholzinger,
autor šablony
(květen 2024)

Obsah

Úvod 1

Na Katedře informatiky a výpočetní techniky (KIV) dochází od akademického roku 2024/25 vzhledem k nové akreditaci studijních programů ke změnám organizace výuky některých vyučovaných předmětů. Jedním z těchto předmětů je Ověřování kvality softwaru (OKS). Předmět v rámci tohoto přechodu mění obsah výuky na testování zejm. webových aplikací napsaných v jazyce PHP namísto dosavadního jazyku Java. Z důvodu vzniku příhodné situace paralelní výuky s předmětem KIV/WEB bylo rozhodnuto o výrazné spolupráci v rámci plnění semestrálních prací studentů. Na předmětu WEB bez výrazných změn studenti nadále navrhují a vytvářejí vlastní webovou aplikaci psanou jazykem PHP. V předmětu OKS budou studenti svoji webovou aplikaci během jejího vývoje průběžně testovat.

Z důvodu velkého množství studentů na obou předmětech je smysluplné využití nějaké formy alespoň částečné automatické validace studentských prací, ovšem dosud používaný validátor toho není koncepčně schopný. Důsledkem čehož vznikla potřeba nového systému, který bude sloužit k odevzdávání studenty zpracovaných úloh, provádět jejich kontrolu a vizualizovat výsledky vyučujícímu. Základní navržení systému počítá se třemi částmi. Prvním znich je systém GitLab, přes který budou studenti uchovávat a tím odevzdávat své rozpracované i dokončené práce v rámci jednoho repozitáře. Ten bude v rámci svých možností podpory DevOps a CI/CD provádět kontrolu a publikovat studentské artefakty na studentské prostředí pro nasazení, jakožto druhé části systému. Třetí část počítá s vytvořením webové aplikace získávající informace ze systému GitLab, zpracovávající výsledky úloh a jejich vizualizaci. Vyučující zde bude mít možnost hodnocení jednotlivých studentů a úloh.

Součástí této práce je návrh a vývoj webové aplikace pro podporu výuky a hodnocení studentů. Dále se práce zabývá navržením validačních kritérií pro validační skripty a vytvořením reprezentační sady artefaktů pro negativní testy ověřující funkčnost aplikace a celého systému.

Analýza 2

2.1 Popis současné výuky předmětů KIV/WEB a KIV/OKS

V bakalářském studijním programu na katedře informatiky a výpočetní techniky se v druhém ročníku vyučují předměty KIV/WEB – Webové aplikace v zimním semestru a KIV/OKS – Ověřování kvality software v letním semestru. Tyto předměty v rámci přednášek ani cvičení nemají žádný průnik probírané látky ani náznak vzájemné návaznosti.

2.1.1 Popis současné výuky KIV/WEB

Předmět KIV/WEB se zabývá úvodním seznáním studentů s webovými technologiemi a tvorbou internetových aplikací. Kurz je určen pro úplné začátečníky, na jeho začátku se proto předpokládá nulová znalost webových technologií. Výuka začleňuje z počátku představení značkovacího jazyka HTML a kaskádových stylů CSS. Následně se pokračuje výukou programovacího jazyka PHP a jeho možností pro použití na webu přes jednoduché dynamické prvky po složitější strukturu webu a zapojením databázového systému MySQL. Kurz kromě toho teoreticky i prakticky představí studentům různé druhy útoků na aplikaci a možnostmi zabezpečení proti nim. Paralelně s tímto předmětem dle standardního průběhu studia je vyučován předmět KIV/DB1 – Databázové systémy 1, kde studenti získají základní znalosti databázových systémů, které využijí při tvorbě semestrálního projektu.

V rámci předmětu KIV/WEB je nutné zpracovat jednu semestrální práci, která je hodnocena až na konci semestru při osobním předvedením. Studenti si dle zadaných instrukcí navrhnou a implementují dle zadání vlastní webovou aplikaci. Do hodnocení práce se započítá velké množství faktorů a klade se důraz přítomnost většiny kurzem představených témat. Vyučující praktických cvičení referuje, že studenti považují tuto semestrální práci za velmi časově náročnou, ale sami přiznávají, že se jí nevěnovali průběžně v celé délce semestru.

2.1.2 Popis současné výuky KIV/OKS

Předmět KIV/OKS (Ověřování kvality software) se zaměřuje na seznámení studentů s principy a technikami zajišťování kvality a testování softwaru. Cílem předmětu je vybavit studenty základními znalostmi a praktickými dovednostmi v této oblasti od logování a jednotkového testování po testování webových aplikací a automatizace testů. Hlavním programovacím jazykem toto předmětu je Java a tento jazyk je využívá pro předvádění probírané látky a vyžadován při plnění semestrálních prací.

Kromě teoretických znalostí klade předmět důraz na praktickou aplikaci. Studenti během semestru odevzdávají deset úloh vždy na jednu probíranou oblast. Úlohy jsou centrálně odevzdávány a následně automaticky kontrolovány validátorem, který ověří správnost řešení a případně akceptuje odevzdání. Kontrolu úloh v tomto předmětu lze považovat za značně automatizovanou a díky tomu se snižuje vytíženost vyučujícího, který nemusí provádět podrobnou ruční kontrolu.

2.2 Popis stávajícího validátoru používaného v předmětu KIV/OKS

Validátor slouží k automatizovanému vyhodnocování elektronicky odevzdaných studentských prací. Byl vytvořen v rámci projektu racionalizace a objektivizace výuky s cílem automatizovat opakující se procesy v rámci kontroly úloh. Jeho implementace přinesla mnoho výhod jak pro studenty, tak pro pedagogy. Studenti díky němu mohou získat zpětnou vazbu na své práce okamžitě, což zvyšuje efektivitu plnění jejich povinností a umožňuje jim reagovat na případně problémy. Pro pedagogy znamená validační server snížení zátěže opakujících se úkolů spojených s ruční kontrolou a hodnocením studentských prací. Další výhodou validátoru je také přispívání k objektivnímu hodnocení prací, jelikož používá pevně stanovená kritéria a vyhodnocuje je konzistentně pro všechny studenty.

Vyučující modeluje úlohu definování testovací domény stanovením validačních pravidel spouštěním různých příkazů nebo také vložením souboru oproti kterému se bude testovat. Je možné například pro testování úlohy v jazyce Java nahrát `.jar` soubor s vytvořenými Java testy, které se následně spouští nad odevzdanou prací. Pomocí testovacích tříd, metod a příkazů `assert` lze ověřit funkce odevzdaného programu stejně jako testování libovolného jiného kódu. Avšak takové testování bývá typicky mířené na konkrétní aplikace o jasné struktuře, využití tohoto přístupu na úlohy i mírně jiného zadání tedy není možný.

Funkčně je validádor sestaven z speciálního softwarového systému, prostřednictvím kterého přijímá elektronické soubory s odevzdanými studentskými pracemi a automaticky je analyzuje podle stanovených kritérií. Proces fungování validačního serveru lze rozdělit do několika kroků [Val08]:

1. **Příjem souborů:** Studenti elektronicky odevzdávají své soubory semestrálních prací prostřednictvím odevzdávacího portletu [24a], který je součástí systému STAG. Tento portlet je propojen s validačním serverem, kterému jsou po odevzdání soubory předány s dalšími relevantními informacemi, jako je identifikace studenta nebo název úlohy.

2. **Identifikace domény:** Validátor začíná proces vyhodnocování tím, že na základě předaných informací identifikuje doménu, podle které se má daná práce validovat.

3. **Kontrola základních požadavků:** Validátor provádí kontrolu, zda odevzdané soubory plní vstupní požadavky jako přítomnost souborů s definovaným jménem a typem, velikost, počet souborů atd.

4. **Spuštění validace:** Validační server připraví prostředí pro ověření vytvořením nového vlákna, které bude validaci zpracovávat. Odevzdaný soubor je pro potřeby zpětného dohledání uložen do speciální složky k tomu určené. Pro úkon validace je vytvořen nový dočasný adresář pojmenovaný unikátním názvem. Následně se ve vytvořeném vlákně spustí samotná validace dle definovaného průběhu pro odpovídající doménu.

5. **Generování výsledků:** V průběhu validace se potřebné informace zapisují do výsledku validace. Výsledek validace je kromě dalších souborů převeden do přehledného HTML formátu. Ten je zobrazován studentovi prostřednictvím poskytnutého odkazu, který je vrácen společně s dalšími informacemi validačním serverem do systému s odevzdávacím portletem.

6. **Čištění a údržba:** Po ukončení validace jsou odstraněny vzniklé a nadále nepotřebné adresáře a soubory spojené s validací, dále je pročištěna vyrovnávací paměť a nakonec dojde k ukončení validačního vlákna. Tímto krokem je ukončena proces validace.

Princip automatizace na základě definovaných domén vztahující se ke konkrétnímu zadání přináší však značné omezení variability a z toho plynoucí limitace. Validátor není určen pro odevzdávání různých variant úloh a jeho možnosti použití dalších nástrojů pro validaci jsou velice omezené. Z toho důvodu je validátor využíván na úlohy menšího rozsahu o jasně definovaném zadání s přesně očekávanou strukturou studentských řešení. Pro validaci komplexních projektů nebo úloh s volně definovaným zadáním je nutné zvolit jiný kontrolní nástroj [Jác17].

2.3 Změny ve výuce v nově akreditovaném studijním programu

V nově akreditovaném bakalářském studijním programu Softwarové inženýrství přichází změna rozložení předmětů ve studijním plánu. Nově je předmět Databázové systémy 1 přesunut z výuky ze zimního semestru druhého ročníku do zimního semestru ročníku prvního. Z toho vychází, že studenti po splnění prvního semestru by měli být vybaveni základy práce s databází a byli seznámení s dotazovacím jazykem SQL, které se předpokládají při studiu předmětu Webové aplikace a dalších předmětů využívajících databázové úložiště.

Další klíčovou změnou je přesun výuky předmětu Ověření kvality software z letního semestru na zimní semestr druhého ročníku. Tato změna je pro tuto práci stěžejní, neboť výuka zmíněného předmětu bude probíhat paralelně s předmětem Webové aplikace (WEB). To umožňuje těsnou spolupráci mezi těmito předměty jako například propojení semestrálních prací těchto předmětů. Současně byla pozměněn obsah výuky předmětu Ověření kvality software, kdy se hlavním programovacím jazykem stal jazyk PHP, který se taktéž vyučuje v rámci předmětu WEB. Hlavní myšlenka této změny je, že studenti budou moci při plnění semestrálních prací předmětu OKS zároveň plnit závěrečnou práci předmětu WEB. Díky tomu studenti budou mnohem více vedeni k průběžné práci na svém projektu a aplikování znalostí z OKS povede ke zvýšení kvality projektu.

Novým předmětem ve studijním programu je předmět KIV/ZPP – Základy programátorské praxe, který seznámí studenty s technologiemi a postupy využívanými v reálném firemním prostředí. Předmět například poskytuje informace ze základů využívání linuxového systému, ukáže studentům práci se soubory ve verzovacích systémech Git, zdůrazní výhody aplikace CI/CD praktik a jejich možností a uvede do problematiky kontejnerů pomocí systému Docker. Studenti díky tomuto předmětu získají nové obzory a praktické povědomí o technologiích, které jsou v partnerských firmách fakulty reálně na denní bázi využívány.

Zmíněné úpravy studijního programu jsou evoluční změnou staršího programu a snaží se zapojit nejnovější technologie a mezioborovou praxi. Díky změnám v uspořádání předmětů dovoluje bližší mezipředmětovou spolupráci, na které mohou benefitovat jak studenti, vyučující, tak i softwarové firmy jako případní zaměstnavatelé studentů a budoucích absolventů.

2.4 Prozkoumání API systému GitLab pro možnosti validace

2.4.1 Verzovací systém GitLab

Git je bezplatný a open source distribuovaný systém pro správu verzí. Je určený pro správu a sledování změn kódu a jeho hlavní výhodou je funkce větvení vývojových běhů. Díky tomu je efektivně umožněno paralelizovat vývoj například pomocí techniky feature branch. Systém Git umožňuje udržovat historii změn, revidovat změny a koordinovat práci v týmu.

GitLab je online platforma pro správu Git repozitářů, která poskytuje uživatelské prostředí a analytické nástroje nad repozitářem. Nepracuje pouze s Git repozitáři, ale provozuje služby pro projektové řízení jako například správa přístupu, agilní plánování, registr balíčků a pro nás důležité i poskytování různých možností pro rozšíření (API) a zajištění platformy pro CI/CD.

2.4.2 GitLab GraphQL API

GraphQL je dotazovacím jazykem pro dotazování na rozhraní GraphQL API na serverové části. Hlavním specifikem jazyka je umožnění uživatelům upřesnit, jaká data potřebují, a získat je ve strukturizované odpovědi. Výhodou použití takového dotazovacího jazyku je optimalizace přenosu, protože přenáší pouze ty informace, které chceme opravdu potřebuje a definovali jsme si je. Součástí GraphQL je taktéž typový systém, který definuje datové typy atributů a jejich vztahy, díky kterému je poskytována datová kontrola při automatické validaci dotazů. Nevýhodou využití tohoto dotazovacího jazyka může být přílišná komplexnost při použití na menších projektech [Bes24].

Systém GitLab poskytuje rozhraní pro komunikaci pomocí GraphQL, ovšem dosud nepodporuje veškeré funkcionality, které jsou systémem nabízeny.

2.4.3 GitLab REST API

REST API je architektonický styl pro vytváření rozhraní mezi různými softwarovými systémy. Jeho hlavním principem je využití standardních HTTP metod GET, POST, PUT, DELETE, atd. pro manipulaci s daty a zdroji, které jsou identifikovány pomocí URI (Uniform Resource Identifier). REST API je navrženo tak, aby bylo jednoduché, efektivní, rozšiřitelné a snadno použitelné pro interakci mezi klienty a serverem [Che+17].

Systém GitLab poskytuje komplexní REST API, které umožňuje integraci s různými systémy a nástroji. Rozhraní GitLab REST API zpřístupňuje celou řadu datových zdrojů, které mohou být užitečné pro další využití ve vyvíjené aplikaci. Vý-

hodou použití REST API je například jednodušší pochopení principů, které jsou založeny na standardních HTTP metodách a URL adresace. Nevýhodou mohou být problémy s overfetching a underfetching[1] spojenými s neefektivní komunikací a špatná škálovatelnost.

2.4.4 Využití GitLab API

V rámci svého GitLab API poskytuje mnoho datových zdrojů pro jeho plné ovládání a čtení jak pro rozhraní REST API, tak částečně i pro GraphQL. V této sekci se zaměříme na možnosti použití zdrojů pro nově vytvářenou aplikaci a interakce se systémem pro validování studentských prací. Pro tyto účely si představíme datové zdroje, které mohou být novou aplikací využity pro získávání potřebných informací a vykonávání akcí.

2.4.4.1 Datový zdroj Repository (Repositories)

Datový zdroj Repository poskytuje informace o repozitářích uložených na platformě GitLab. Obsahuje detailní údaje o jednotlivých projektech, včetně jejich názvu, popisu, vlastníka, větví, tagů a dalších relevantních informací. Repository je v platformě GitLab podřízeným prvkem struktury projektu. Abychom získali údaje o repozitáři, musíme pracovat s datovým zdrojem Project (Projects). Využití tohoto datového zdroje pro účely aplikace a validace je získání informací o jednotlivých studentských projektech a repozitářů a získání jejich identifikátorů pro další dotazování.

V následujících příkladech se pokusíme ze systému GitLab získat základní informace o projektu a seznam větví repozitáře.

Příklad využití v GraphQL API.

Dotaz v GraphQL pro získání základních informací o projektu a názvy větví repozitáře:

```
query {
  project(fullPath: ":fullpath")
  {
    id
    name
    description
    createdAt
    repository
    {
      branchNames(searchPattern: "*" limit: 10 offset:
0)
```

[1] Přenášení nadbytečných dat, které nebudou využity, a nedostatečné získání potřebných dat

```
        }
      }
  }
```

Příklad využití v REST API.

Pro získání základních informací o projektu:

```
GET /projects/:id
```

Pro získání seznamu větví:

```
GET /projects/:id/repository/branches
```

Můžeme si povšimnout zajímavých odlišností, které budou doprovázet i další ukázky. Příklad v GraphQL je zásadně náročnější na přípravu, vyžaduje značnou znalost a orientaci v dokumentaci, protože musíme vypsat veškeré atributy, které chceme získat. Na druhou stranu přijímají se pouze taková data, která jsou vyžádána, čímž se snižují nároky na přenos a zpracování dat na straně klienta.

Varianta REST API je psána jednodušeji, i samotná dokumentace je čitelnější, ale můžeme si povšimnout, že na získání námi potřebných informací jsme potřebovali dva požadavky, což může působit výkonnostní problémy stejně jako jejich obsáhlá odpověď. Pro získání povědomí o velikosti odpovědí jednotlivých metod, byly představené dotazy vyzkoušeny na jednom projektu, kde odpověď GraphQL dotazu činila 18 řádek, narozdíl od požadavku REST, jehož odpovědi obsahovaly 141 a 119 řádků.

Všimněme si též u příkladu s GraphQL funkci filtrování a stránkování, tyto možnosti jsou typické pro oba přístupy požadavků a vedou k optimalizaci získávání dat a omezení dlouhých výpisů.

2.4.4.2 Datový zdroj Pipeline (Pipelines)

Důležitou funkcí, která je užitečná v kontextu vyvíjené aplikace, je spouštění běhů pipeline pro konkrétní projekt. Tím se spustí řada pipeline úloh a provede se kontrola a validace. Aplikace tedy může vytvářením nových pipeline implementovat např. funkci vyžádání nové kontroly nebo spuštěních dalšách procesů jako nasazení nebo inicializace v rámci systému. Vytváření nové pipeline zajišťuje přístupový bod Pipeline. Následující ukázky se tedy snaží o vytvoření nového běhu pipeline s předanými proměnnými z volajícího klienta.

Příklad využití v GraphQL API.

GitLab ve svém GraphQL rozhraní neobsahuje přístupový bod (mutation) pro vytvoření resp. spuštění nové pipeline se zadanými parametry. Obsahuje pouze bod

`Mutation.pipelineTriggerCreate`, který využívá spouštění pipeline jiným způsobem přes trigger tokeny, ale nachází se aktuálně v experimentální fázi. Pro účely této práce tedy není vhodná alternativa v rozhraní GraphQL API.

Příklad využití v REST API.

Pro vytvoření a spuštění nové pipeline s proměnnou určující validaci 1. úlohy:

```
1    POST /projects/:id/pipeline?ref=production&variables
    [][key]=BUILD&variables[][value]=01\_UC
```

2.4.4.3 Datový zdroj Job (Jobs)

Datový zdroj poskytuje přístup k úlohám pipeline, které byly v projekty kdy naplánované. Lze pomocí nich přistoupit k výsledku úlohy, což může sloužit jako výsledek validace. Taktéž přes tento endpoint lze znovu úlohy spouštět nebo získat informace o jejich artefaktech. Následující ukázky získají pro identifikovaný projekt seznam zpracovaných úloh se stavem úspěšně dokončené nebo selhané.

Příklad využití v GraphQL API.

Dotaz v GraphQL seznamu úspěšných a neúspěšných pipeline úloh:

```
1    query {
2      project(fullPath: ":fullpath")
3      {
4        id
5        name
6        description
7        createdAt
8        jobs(statuses: [SUCCESS, FAILED])
9        {
10         nodes {
11           name
12           status
13           duration
14           createdAt
15           stage {
16             name
17           }
18           tags
19         }
20       }
21     }
22 }
```

Příklad využití v REST API.

Pro získání seznamu úloh pipeline:

```
1       GET /projects/:id/jobs?scope[]=success&scope[]=
    failed
```

2.4.5 GitLab Webhooks

Platforma GitLab, kromě výše představených možností pro získávání dat dotazováním, umožňuje využití mechanismu odeslání notifikace na externí systém v reálném čase. Funkcionalita GitLab Webhooks je užitečná pro automatizaci různých procesů a integraci GitLab s dalšími službami a vývojových nástrojů.

Když je v GitLabu vyvolaná určitá událost (např. dokončení běhu pipeline, nebo push do repozitáře), GitLab vytvoří a odešle HTTP požadavek obsahující informace o této události. Externí systém na nastavené URL naslouchá a po přijetí požadavků může vyvolat další akce v rámci svého systému. Díky implementaci GitLab Webhooks lze díky propojením s dalšími nástroji zlepšit efektivitu a automatizaci vývoje [24f].

GitLab zasílá informace odesílané přes webhooky ve formátu JSON. Příkladem obsahu požadavku může být tento požadavek notifikující přidání uživatele do skupiny (viz ukázka kódu 2.1).

Zdrojový kód 2.1: Příklad ukázky GitLab Webhooks (zdroj: vlastní)

```
1   {
2     "created_at": "2024-04-09T10:18:36Z",
3     "updated_at": "2024-04-09T10:18:36Z",
4     "group_name": "OKS-WEB",
5     "group_path": "oks-web",
6     "group_id": 45,
7     "user_username": "hintik",
8     "user_name": "Jan␣Hinterholzinger",
9     "user_email": "[REDACTED]",
10    "user_id": 36,
11    "group_access": "Developer",
12    "group_plan": null,
13    "expires_at": "2025-04-09T00:00:00Z",
14    "event_name": "user_add_to_group"
15  }
16
```

GitLab Webhooks notifikace lze nastavit pro celou GitLab instanci, jednotlivé repozitáře a v prémium verzi i na samostatné skupiny projektů (GitLab Groups). Pro pokročilé funkcionality lze i vytvářet vlastní události, které GitLab bude notifikovat.

2.4.6 GitLab a CI/CD

Pro naše potřeby je klíčovou funkcí poskytování nástrojů pro kontinuální integraci (CI) a nasazování (CD), což umožňuje automatizovat procesy testování a nasazování softwarových projektů. Vhodná implementace CI/CD praktik do projektu může vést k doručování kvalitního software a snížení doby na vývoj [Ana21].

Pipelines v GitLabu jsou automatizované pracovní postupy definované souborem `.gitlab-ci.yml` a skládají se z jednotlivých úloh „Jobs“, které mají jednoznačný účel (např. kompilace kódu, spuštění statické analýzy nebo nasazení do produkčního prostředí). Pro zvýšení možností paralelního zpracování úloh se joby shlukují do tzv. Stages, které logicky obalují úlohy například pro testování. Pipeline lze spustit s námi definovanými parametry, které mohou poskytovat data jednotlivým úlohám nebo variabilně řídit běh pipeline [22].

Běhy pipeline mohou být spuštěny několika způsoby. Základním způsobem je tzv. vytvoření pipeline přes rozhraní GitLabu, kde lze definovat vývojovou větev, nad jejímiž soubory budou úlohy pracovat, a variabilní počet proměnných.

Po vytvoření a spuštění běhu pipeline se postupně zahájí zpracování jednotlivých úloh. Úlohy jsou zpracovávány tzv. GitLab Runner, který dle konfiguračního souboru pipeline připraví běhové prostředí a vykoná definované příkazy. Po vykonání scénáře úlohy je úloha označena stavem úspěšnosti operací (tj. úspěšné zpracování, selhání, přeskočeno atd.). GitLab se snaží optimalizovat využití zdrojů pro běh pipeline, takže například po selhání úlohy nemá smysl pokračovat ve zpracovávání dalších úloh a jednoduše další úlohy přeskočí (nebude je vykonávat) [AS19].

GitLab Runner je samostatný proces, který je určen pro zpracovávání úloh. Tato služba periodicky přijímá definice úlohy a nad repozitářem vytvoří prostředí pro běh. Po ukončení zpracování a čištění je runner připraven pro zpracovávání další úlohy.

2.4.7 Další integrace s GitLab

Verzovací platforma GitLab podporuje různé druhy autentizace. Výchozím variantou je přihlášení pomocí přihlašovacího jména a hesla. Lze ji však kompletně nahradit dalšími metodami jako je například OAuth 2.0 nebo jeho nadstavba OIDC. Právě protokol OIDC je podporován ze strany univerzitního přihlašování k autorizaci studentů, vyučujících a dalších zaměstnanců univerzity. Instance GitLabu určeného pro výuku má přihlášení omezené pouze přes OIDC. Toto nastavení jednoznačně identifikuje přihlášeného uživatele a je velmi výhodné pro práci ve výpočetním prostředí ZČU.

Pro propojení GitLab uživatele s jednotným přihlašovacím systémem (SSO) stačí, aby se uživatel přes přihlašovací rozhraní SSO přihlásil do systému GitLab. V případě neexistujícího uživatele je mu vytvořen nový již propojený. Uživatele lze vy-

tvořit i programově například použitím API přístupu, kdy je potřeba uživateli pouze nastavit, kromě jiných, další atributy jeho parametrů entity například přihlašovací Orion[2] jméno (viz ukázka 2.2 pro propojení s OIDC) [24e].

Zdrojový kód 2.2: Atributy GitLab uživatele pro přihlášení přes univerzitní SSO (zdroj: vlastní)

```
1     {
2       ... // další atributy
3       "force_random_password: "true",
4       "provider": "openid_connect",
5       "extern_uid": "orion-login",
6       "saml_provider_id": "null",
7     }
8
```

2.5 Role aplikace v rámci nového hodnocení studentských prací

V závislosti na změně požadavků semestrálních prací v předmětu OKS, kdy se využívá vznikající aplikace jako semestrální práce předmětu WEB pro navázání hlubšího propojení vývoje a testování, již nebude možné využít dosud používaný validátor z předmětu OKS. Hlavním důvodem je značná variabilita zadání při tvorbě webové aplikace. Takovou podstatu nelze dostatečně efektivně zachytit definováním domén validátoru a případná generalizace by vedla k nemalé míře nedostatečné kontroly ze strany vyučujícího.

Z toho důvodu bylo rozhodnuto pro potřeby předmětu OKS opustit doposud využívaný validátor a vytvořit nový systém založený na principech DevOps, kontinuální integrace (CI) a nasazení (CD). Studenti namísto odevzdávání archívů do odevzdávacího portletu informační studijní agendy IS/STAG budou své úlohy odevzdávat formou verzování v systému Git. Na instanci GitLabu, provozované pod správou CIV pro KIV za tímto účelem je v rámci jiné práce vytvořen technologický stack jednotlivých úloh, které jsou dockerizované a je podchycen jejich účel pomocí CI/CD pipelines. Každá úloha má svoji sadu pipeline úloh, které jsou variabilně spouštěny na základě odevzdávání do předem pevně dané adresářové struktury.

Každému studentovi tak na začátku semestru vytvořen repozitář funkcí `fork` ze šablonového repozitáře, a je mu tam udělen přístup. Protože studentům již byla technologie Git představena na předmětu ZPP, je očekávána alespoň základní znalost praktik práce s repozitářem a průběžném vývoji.

[2] Název pro jednotné přihlašování (SSO) na ZČU

Součástí celého procesu je taktéž Kubernetes Cluster, kde je studentům vytvořením repozitáře inicializací založen namespace, kam pipelines automaticky dle odevzdávaných úloh nahrávají a tím i publikují své mezivýsledky. Student tam kromě výpisů z úloh pipeline může prokliknout skrz generovaný dashboard ke grafickým výstupům kontrol jednotlivých úloh. Například při odevzdávání případů užití ve formátu XML v první úloze jsou automatickým procesem při odevzdání převedeny do formátu HTML a jsou publikovány na adrese odpovídající artefaktům první úlohy.

Taktéž je zde připraveno PHP prostředí a automatické nasazení pro odevzdání webové aplikace k předmětu WEB. Výhodou tohoto přístupu je ověření studentům, že jejich aplikace funguje na produkčním prostředí. To je významná změna oproti současnému stavu, kdy byl vývoj omezen prakticky na jeden počítač s nainstalovaným WAMP serverem. Eliminuje se argument „na mém počítači mi to fungovalo, proč to nefunguje tady“. Dále se zmenší režie kontroly a tím zkrácení času na kontrolu, protože všechny semestrální práce by měly být dostupné na předem jasně definovaných adres na kterých se může vyučující kdykoli podívat na vypracovanou semestrální práci. To vše navíc podporuje využívání praktik a technologií v praxi hojně používaných jako jsou Git, Docker, CI (průběžné testování, statická analýza, apod.), CD (průběžné nasazování na staging a produkční prostředí).

2.5.1 Zasazení aplikace do vznikajícího ekosystému

Do těchto procesů je zasazena webová aplikace s pracovním názvem „OKS-WEB-GARANT-APP“, která má za úkol sledovat procesy ve zmíněném systému, zaznamenávat pokroky a evaluovat a vizualizovat výsledky. Mezi základní funkce aplikace patří zejména zobrazování stavů odevzdání jednotlivých úloh studentovi i vyučujícím, možnost udělovat body a zaznamenávat nedostatečná odevzdání. Pro tyto účely je důležité, aby aplikace ve významném rozsahu využívala poskytovaná data systému GitLab, udržovala konzistentní stav a byla dostatečně odolná vůči chybám. Zejména poslední zmíněné je v kontextu hodnocení studentů kritické a je potřeba tuto oblast dostatečně ověřit.

2.6 Požadavky zadavatele

Požadavkem zadavatele je vytvoření komplexního systému pro automatickou evaluaci studenty odevzdaných prací. Tato práce se zabývá částí vytvoření webové aplikace, která bude získávat informace z instance GitLab a prezentovat je ve svém rozhraní. Požadavky na tuto aplikaci jsou sepsány a očíslovány.

2.6.1 Funkční požadavky

RQM-F-01 – Nahrávání studentů. Zadavatel si přeje do aplikace jako vyučující nahrát seznam studentů po jednotlivých studentech, ale také hromadně pomocí CSV souboru exportovaného ze systému studijní agendy IS/STAG.

RQM-F-02 – Zobrazení výsledků. Aplikace obsahuje tabulku se sloupci obsahující jméno a příjmení, osobní číslo, počet dosažených bodů a další sloupce označené čísly 1 až 10 označující jednotlivé úlohy. Řádky tabulky pak zaplňují data jednotlivých studentů. Buňky tabulky ve sloupcích představující úlohy a obsahují tyto náležitosti, které mohou být zobrazeny současně:

- **počet bodů** – číselné označení obdržených bodů za konkrétní úlohu. Pokud úloha dosud není ohodnocena, zobrazuje se jako počet bodů „0“
- **indikace minimálního rozsahu** – pokud student nahlásil odevzdání v minimálním provedeném rozsahu (PMIN), je tato skutečnost indikována v buňce písmenem „M“ se zvýrazněním
- **indikace problému** – pokud vyučující v přidá komentář typu „problém“, je tato skutečnost v buňce indikována přítomnost písmene „P“ se zvýrazněním. Pokud jsou ve stejné úloze studentovi vyznačeny dva nebo více problémových komentářů, vždy se indikuje pouze jedním výskytem písmene „P“

Buňky úloh v tabulce se budou obarvovat dle stavu odevzdání odpovídajících úloh daného studenta. Barvy podbarvení pro jednotlivé stavy jsou definovány následovně:

- **oranžová** – automatická kontrola v pipeline nebyla úspěšně dokončena a je třeba odevzdat úlohu znovu a kvalitněji
- **žlutá** – automatická kontrola v pipeline úspěšně prošla, ale po termínu odevzdání (pozdní odevzdání)
- **světle zelená** – automatická kontrola v pipeline úspěšně prošla, ale vyučující dosud nepřidělil bodové hodnocení
- **zelená** – vyučující provedl ruční kontrolu a přidělil bodové hodnocení. Pokud byla úloha odevzdána po stanoveném termínu odevzdání, informace o tom je dostupná po rozkliknutí buňky.
- **žádné podbarvení** – úloha dosud nebyla odevzdána

Obarvovat se budou i buňky ve sloupci pro osobní číslo. Buňky se budou obarovat dle následujících pravidel v pořadí dle priority:

- **zelená** – student má nárok na zápočet. Student odevzdal všechny úlohy, obdržel dostatečný počet bodů pro získání zápočtu a počet indikovaných problémů nepřesáhl hranici počtu 4 (včetně) výskytů,
- **černá** – po 4. nevyhovující kontrole úlohy (problémem) vyučujícím student ztrácí nárok na zápočet a tato skutečnost je indikována černým pozadím buňky
- **odstíny červené** – buňka je zbarvována na základě počtu detekovaných problémů. Od prvního výskytu, kdy se položka zbarví do velmi lehce červené přes 2. incident označený světle červeně po 3. varování, které je představené rudou červenou barvou
- **žádné podbarvení** – žádná předchozích podmínek není splněna

RQM-F-03 – Zobrazení detailu odevzdání. Kliknutím na buňku zobrazující základní přehled hodnocení úlohy se objeví popup okno, které obsahuje podrobnější informace o odevzdání. Mezi takové informace patří například datum odevzdání, indikace pozdního odevzdání, rozsah odevzdané práce.

RQM-F-04 – Udělování bodového hodnocení. Při rozkliknutí detailu hodnocení, je zde umístěna i funkce bodování. Vyučující zde vidí kolik bodů aplikace navrhuje k ohodnocení. Tuto hodnotu vyučující může přijmout nebo ji upravit vlastní číselnou hodnotou a hodnocení uložit.

Navrhované bodování je složeno z několika složek. Počáteční hodnota je dána zvoleným rozsahem odevzdání, kde se bodový základ může lišit u každé úlohy a rozsahu. Vystupující bodový základ je snížen o penalizaci za pozdní odevzdání. Postih bodů je opět variabilní na konkrétní úloze.

Součástí úpravy evaluace je taktéž nastavení odevzdaného rozsahu z výběru DOP (doporučeno) a PMIN (minimální povinné).

Po udělení hodnocení již student nemá možnost zvolit si odevzdávaný rozsah úlohy.

RQM-F-05 – Zobrazení artefaktů úloh studenta. Student i vyučující má k dispozici odkazy, které vedou k aktuálně publikovaným artefaktů výsledného odevzdání.

RQM-F-06 – Zobrazení pipeline běhů studenta. Vyučující vidí seznam proběhlých pipeline běhů pro zadaného studenta. Je zde zobrazen zejména výsledný status pipeline, začátek a konec běhu a doba trvání.

RQM-F-07 – Komentáře a problémy. Mezi hodnocení studenta patří i zpětná vazba. Aplikace umožňuje studentům k jednotlivým úlohám zadávat komentáře a vkládat varování (tj. vyučujícím detekovaný problém v odevzdané úloze). Zadávání varovných komentářů indikuje nedostatečně vypracovanou úlohu, je vyžadováno přepracování a má další negativní konsekvence v celém systému.

RQM-F-08 – Nastavení kontrol jednotlivých kontrol. Aplikace musí být dostatečně vybavena uživatelskými vstupy, aby bylo možné jednoduše a bez změn v implementaci aplikace upravit kritéria zpracování kontrol. Jedná se zejména o mapování pipeline běhů a jejich jobů k jednotlivým úlohám v aplikaci.

RQM-F-09 – Nastavení bodů pro každou úlohu. Každé úloze lze vyučujícím nastavit základní hranici bodů pro rozsahy odevzdání DOP a PMIN a rovněž také vlastní hodnotu penalizačních bodů za pozdní odevzdání.

RQM-F-10 – Poskytování stavů úloh pro konkrétního studenta (API). Aplikace stavy odevzdání jednotlivých úloh poskytuje formou API, aby bylo možné se dotazovat na konkrétního studenta. Tato funkcionalita je vyžadována, zejména k získání dat pro vizualizaci stavu odevzdání na generovaném dashboardu nasazeném na produkčním prostředí každého studenta.

RQM-F-11 – Přihlášení studenta. Student musí prokázat svoji totožnost pro přístup do studentské části aplikace.

RQM-F-12 – Zobrazení hodnocení studentovi. Student se po autentizaci může přihlásit do studentské části aplikace, kde vidí informace pouze ke své osobě. Mezi zobrazované údaje je status získání zápočtu, počet bodů, stav hodnocení jednotlivých úloh s podrobnými informacemi včetně započteného času odevzdání, obdržených bodů a případně vyučujícím zadané zpětné vazby.

RQM-F-13 – Automatické bodování úloh. Vyučující má možnost u každé úlohy při její kontrole stisknout tlačítko u každé úlohy, kterým se udělí aplikací navrhovaný počet každému studentovi, který semestrální práci úspěšně odevzdal. A Podmínkou je, že automatická kontrola prošla v pořádku a dosud nemá bodové hodnocení. Tímto požadavkem se výrazně zrychlí bodování úloh studentů a na základě efektivních kontrol na straně automatické kontroly se zkrátí čas na manuální kontrolu odevzdaných artefaktů.

2.6.2 Nefunkční požadavky

RQM-N-01 – Získávání dat. Získávání dat pro aplikaci je zajištěno rozšiřováním systému GitLab pomocí jeho API.

RQM-N-02 – Požadavek na produkční prostředí. Aplikace je vyvinuta pro běh na prostředí katedrálního serveru s pevně danými náležitostmi.

RQM-N-03 – Konfigurace. Aplikace všechny provozně variabilní náležitosti umožňuje nastavovat v čitelných konfiguračních souborech, jejichž struktury a funkčnosti jsou náležitě popsány v dokumentaci a komentáři přímo v konfiguraci.

RQM-N-04 – Otestovaná aplikace. Aplikace je jednotkově otestovaná, jsou vytvořeny integrační testy. Zejména jsou otestovány identifikované kritické oblasti aplikace, jako např. hodnocení studentů.

RQM-N-05 – Logování. Aplikace si vede záznamy o provedených akcích formou logovacích souborů.

2.6.3 Požadavky na běžící prostředí

Analýza požadavků na běžící prostředí klade důraz na specifické požadavky zadavatele, které jsou klíčové pro správné nasazení a provoz aplikace. Zadavatel specifikuje, že aplikace bude provozována na serveru `ares.fav.zcu.cz`, což je katedrální server určený jejím potřebám. Tímto požadavkem je definováno cílové prostředí pro nasazení aplikace, což umožňuje přesné plánování a přípravu infrastruktury pro provoz.

Dále je důležité zdůraznit implementační požadavek, že aplikace musí být napsána v jazyce PHP. Specifikací produkčního prostředí je definováno, že aplikace bude nasazena na webový server Apache2 spolu s PHP verzí 8.2RC5, která je na něm přítomna. Tato konfigurace vyžaduje, aby aplikace byla plně kompatibilní s tímto běhovým prostředím a aby byla optimalizována pro správnou funkčnost.

Návrh řešení 3

3.1 Dekompozice požadavků

3.1.1 Uživatelské role

Návrh definuje 7 uživatelských rolí dvě skupiny. První skupinou jsou role, které jsou v rámci webového grafického rozhraní přístupné a očekává se lidský uživatel. Těmito uživateli jsou: Nepřihlášený uživatel, Student, Garant a Superadmin. Ve druhé skupině se pracuje s aktéry přístupem robotizovaným, kdy s aplikací komunikuje typicky jiný program. Pro tento účel byly definovány role: Klient veřejného API, GitLab Webhooks a Automatický zpracovatel.

3.1.1.1 Nepřihlášený uživatel

Při přístupu neautentizovaného uživatele do webového rozhraní aplikace, je uživatel značně omezen na poskytovaných funkcích. Takový uživatel má možnost přihlášení do role Student výhradně přes univerzitní jednotný přihlašovací systém nebo do role Garant stejnou autentikační metodou nebo validním přihlašovacím jménem a heslem.

3.1.1.2 Student

Uživatele role student reprezentuje student předmětu KIV/OKS, kterému je po úspěšné autentikaci zobrazen například postup jeho odevzdávaných úloh, jejich hodnocení a studentovo celkové hodnocení včetně informace o udělení zápočtu. Kromě těchto informací je uživateli za určitých podmínek umožněno určit rozsah plnění jeho odevzdaných semestrálních prací.

3.1.1.3 Garant

Klíčová uživatelská role, která představuje vyučujícího a hlavního uživatele celé aplikace. Systém poskytuje roli správu studentů, zpracování jejich výsledků a další

vizualizace. Garant není postaven do role pouhého konzumenta obsahu, ale z velké části se podílí na konfiguraci a vkládání vlastních dat.

3.1.1.4 Superadmin

Role Superadmin je uživatel s nejvyššími oprávněními a kontrolu nad celým systémem. Má schopnost spravovat uživatelské účty a konfigurovat aplikaci. Má přístup k veškerým funkcím a datům v systému.

3.1.1.5 Klient veřejného API

Softwarová aplikace nebo služba, která komunikuje s aplikací přes rozhraní API. Uživatel získává informace poskytované definovaným veřejným rozhraním.

3.1.1.6 GitLab Webhooks

Mechanismus integrovaný s GitLabem, který odesílá notifikace do aplikace v reakci na určité události v repozitářích. Může spouštět akce v aplikaci na základě událostí v GitLabu.

3.1.1.7 Automatický zpracovatel

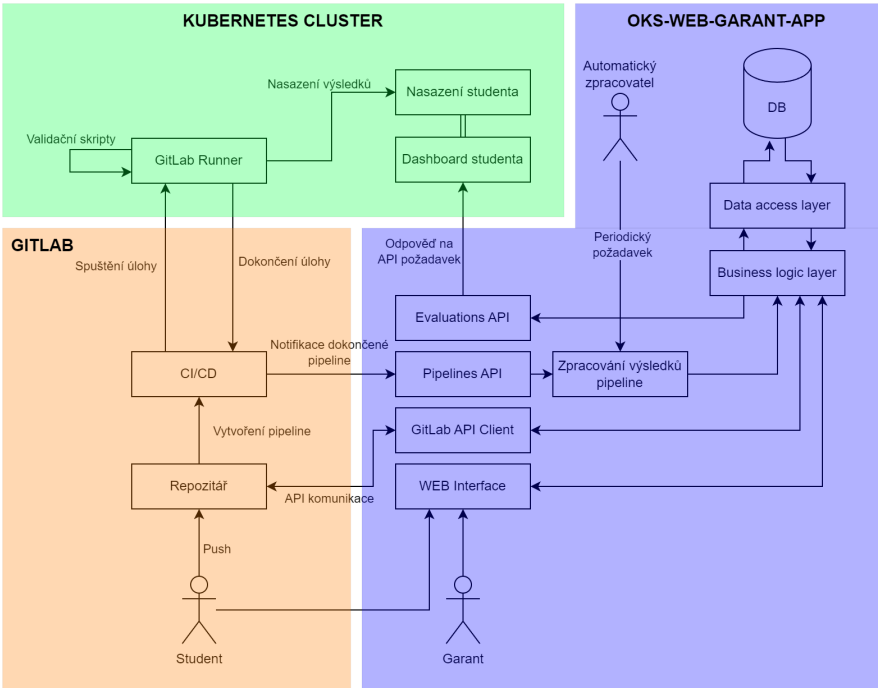
Automatizovaný proces, který periodicky spouští v aplikaci akce pro získání dat z fronty a jejich zpracování.

3.1.2 Struktura systému

Návrh počítá s vyvinutím samostatné webové aplikace v rámci systému pro poloautomatickou správu studentských prací. Role této aplikace v systému je zpracovávání dat získaná ze systému GitLab, jejich vizualizace vyučujícímu a studentům.

Celý systém lze rozdělit na tři části dle provozu samostatných systémů, jak je vidět na schématu celého systému (viz obr. 3.1). Jedním z těchto systémů (modré podbarvení) je navrhovaná aplikace, které se věnuje tato práce včetně zapracování návaznosti na systém GitLab (oranžové podbarvení ve shématu). Ostatní části systému jsou součástí jiné právě vznikající práce, s jejímž autorem bylo potřeba značně spolupracovat na návaznostech celého systému.

Aplikace je rozdělena čtyři hlavní moduly, dle druhu poskytovaného obsahu: Studentský modul, Garantský modul, API a Console. Všechny zmíněné moduly mají své uživatelské role, které obsluhují a poskytují jim služby.



Obrázek 3.1: Diagram systému s návaznostmi (zdroj: vlastní)

3.1.2.1 Studentský modul

Jedním z modulů je studentský, který slouží právě studujícím studentům předmětu OKS. Tento modul představuje jednoduché rozhraní o jenom pohledu na kterém jsou kondenzovány veškeré informace, které student o odevzdaných úlohách potřebuje znát. Kromě informací o jednotlivých úloh modul zobrazuje výsledky z celého semestru, případný nárok na zápočet a bodové hodnocení včetně penalizačních bodů za pozdní odevzdání nebo nedostatečnou kvalitou vypracovaných úloh.

3.1.2.2 Garantský modul

Primární modul aplikace pro správu celého předmětu. Jedná se o modul zejména pro uživatelskou roli Garant (vyučujícího), který zde definuje náležitosti automatických kontrol, má k dispozici menší studijní agendu v podobě správy studentů a hlavně pohled s vizualizací výsledků v tabulkové podobě, kde vidí průběh odevzdávání všech studentů.

3.1.2.3 API

Aplikace kromě funkcionalit určených pro obsluhu uživatelů pomocí webového grafického rozhraní poskytuje v rámci své návaznosti na další systémy API. Toto

rozhraní poskytuje informace o jednotlivých plnění úloh konkrétních studentů a přijímá data z externích systémů.

3.1.2.4 Console

Možnost volat funkcionality aplikace skrz příkazovou řádku. Jedná se zejména o funkci získání dat z fronty webhook požadavků, jejich zpracování a uložení výsledků. Tento proces bude spouštět automatický plánovací systém (např. CRON) přes příkazovou řádku na produkčním prostředí. Tento návrh je přítomen, protože je zde omezená doba, kdy aplikace musí odpovědět na příchozí webhook notifikaci. Proto je zpracování příchozích dat odloženo na později, kdy aplikace bude mít čas na zpracování.

3.1.3 Detekované entity

V rámci analýzy požadavků a dekompozice zadání byly pro aplikaci definovány modelové entity, které představují datové celky (objekty) s jasně popsaným významem. Vztahy mezi nejdůležitějšími entitami ilustruje ERD (viz obr. 3.2).

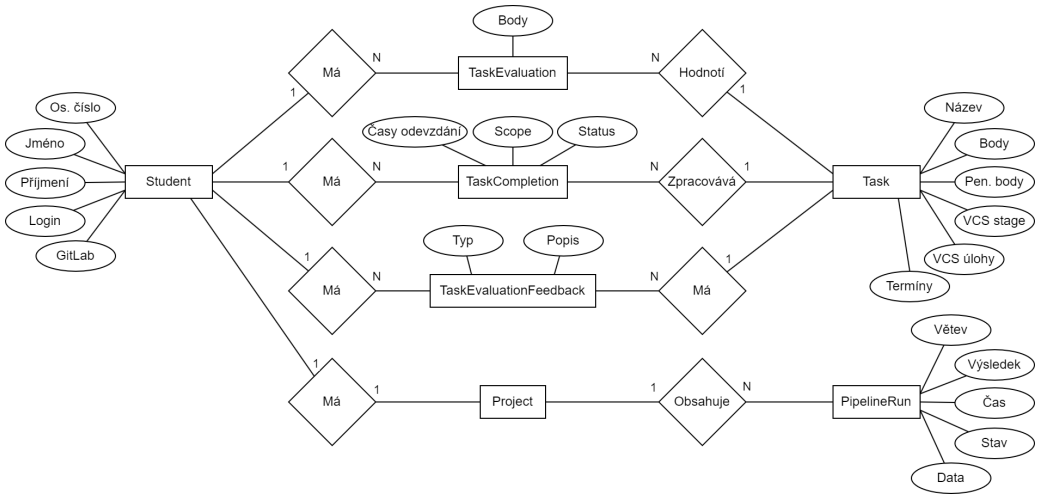
3.1.3.1 User

Datová entita sdružující informace o uživateli zejména informací nutné pro přihlášení, jeho role v aplikaci a platnosti uživatelského účtu. Mezi metody autentizace jsou přípustné tři hodnoty:

- **orion_only** – Uživatel se může přihlásit pouze svým orion účtem, tedy prostřednictvím systému jednotného přihlašování poskytovaného univerzitou
- **password_only** – Uživatel se může přihlásit pouze údaji k této aplikace tj. přihlašovacím jménem a heslem
- **both** – Uživatel má možnost přihlásit se oběma způsoby, je proto v tomto případě potřeba definovat údaje k oběma metodám přihlášení

3.1.3.2 Task

Struktura představující úlohu pro plnění studenty. Tato úloha má svůj název (plný a zkrácený), bodové hodnocení pro různé rozsahy plnění odevzdání i termín pro včasné odevzdání. Pro plnění účelu evaluace jsou potřeba atributy s názvem stage a seznam pipeline úloh, které musí být splněny při automatické kontrole. V poslední řadě entita obsahuje údaje pro projení s dashboardem studenta, pro který se zde definuje relativní cesta k souborům artefaktu úlohy a HTML identifikátor pro příslušnou kartu v zobrazení.



Obrázek 3.2: Diagram vztahů entit (zdroj: vlastní)

3.1.3.3 Student

Datová struktura studenta je původně získávána z extrakcí informací importovaného CSV souboru exportovaného ze systému STAG. V závislosti na tom byly vytipovány potřebné a dosačující údaje jako jsou například jméno a příjmení, osobní číslo, orion login a email. Dále entita obsahuje identifikaci uživatele v systému GitLab a projektu pro odevzdávání úloh.

3.1.3.4 Project

Entita student obsahuje propojení s entitou Project, která představuje GitLab projekt přiřazený studentu, kam student odevzdává své vypracované úlohy.

3.1.3.5 PipelineRun

Proběhlý běh pipeline je reprezentován entitou PipelineRun, která udržuje zejména surová data získaná z notifikační služby určené pro zpracování. Současně jsou v entitě obsaženy extrahované informace pro identifikaci projektu, výsledku pipeline a důležitých časových údajů pro pipeline běh. Entita je vázána vztahem na konkrétní entitu Projekt.

3.1.3.6 TaskCompletion

Stav automatické kontroly je dán entitou TaskCompletion, která spojuje entitu Student s entitou Task a určuje tak odevzdání úlohy. Entita obsahuje informaci o stavu kontroly, které mohou být:

- **pipeline_cancelled** – pipeline neproběhla nebo byla zrušena

- **pipeline_failed** – automatická kontrola proběhla s negativním výsledkem (pipeline s výsledkem failed nebo nebyly splněny náležitosti pro splnění úlohy)
- **submission_late** – automatická kontrola proběhla s pozitivním výsledkem, ale první úspěšná kontrola proběhla po termínu včasného odevzdání
- **pipeline_succeeded** – automatická kontrola proběhla s pozitivním výsledkem
- **submission_rejected** – garant při ruční kontrole objevil nedostatky a je vyžadována jejich náprava
- **evaluation_accepted** – garant odevzdání zkontroloval a udělil hodnocení splněné úlohy

3.1.3.7 TaskEvaluation

Entita TaskEvaluation představuje garantské hodnocení odevzdané úlohy. Vyznačuje se zejména uděleným počtem bodů pro konkrétního studenta a konkrétní úlohu.

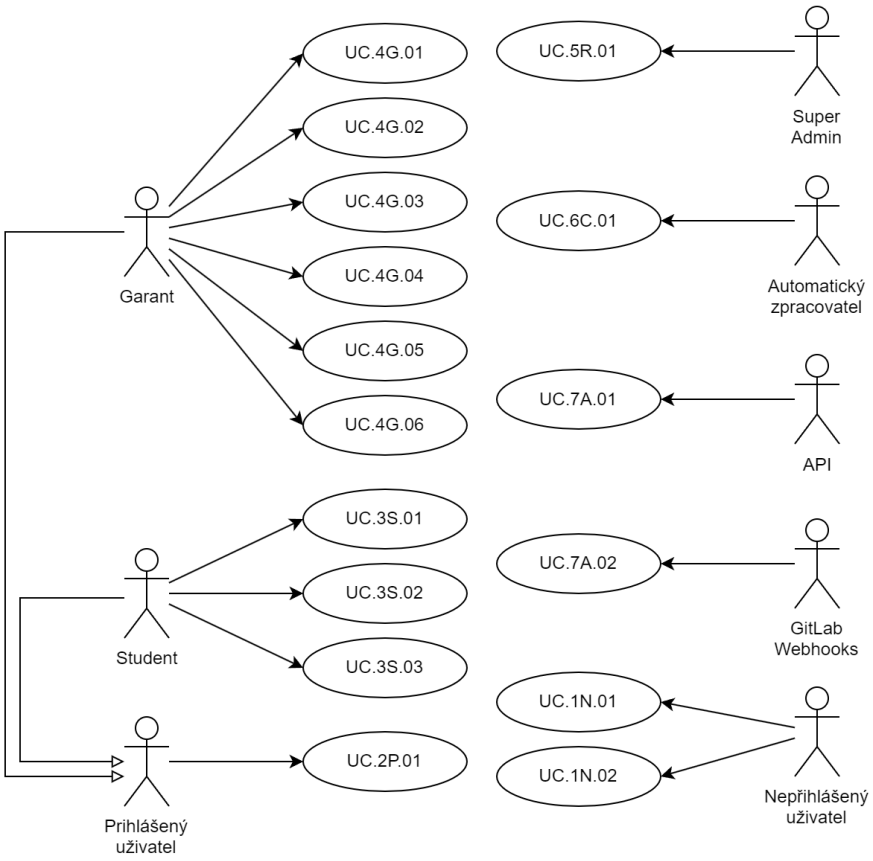
3.1.3.8 TaskEvaluationFeedback

K hodnocení lze taktéž přiložit slovní hodnocení, tzv. zpětnou vazbu, která je zastoupena entitou TaskEvaluationFeedback. Tato zpětná vazba může být dvojího typu:

- **comment** – komentář sloužící k standardnímu slovnímu hodnocení úlohy pro např. opravě špatných informací, vysvětlení nebo i pochvalu
- **problem** – varování při objeveném problému při odevzdání úlohy. Jedná se o důrazný způsob jak studenta upozornit na jeho nevhodnou činnost a nařídit nápravu. Tato přidaná varování způsobují další návaznosti na hodnocení studenta.

3.1.3.9 Settings

Entita představující konfigurovatelnou položku nastavení. V rámci systému lze konfigurovat určité aspekty aplikace, jejichž hodnoty je nutné persistentně ukládat. Struktura Settings tedy obsahuje samotnou hodnotu, která nese konfigurační informaci. Dále také obsahuje metadata například datový typ nebo identifikátor nastavení ke které hodnota patří.



Obrázek 3.3: Diagram případů užití (zdroj: vlastní)

3.2 Případy užití

Z požadavků dílčí specifikace byly sestaveny případy užití popisují funkce a chování aplikace. Některé případy užití byly seskupeny do větších celků „skupin“ dle jejich funkční příslušnosti (např. případy užití pro vytvoření, editaci a smazání některé entity jsou seskupeny do skupiny pro operace s danou entitou) pokud není důvod k jinému členění. Struktura případů užití jsou ilustrována use-case diagramem (viz obr. 3.3) zobrazující i vztahy mezi jednotlivými uživatelskými rolemi a jejich příslušnost k jednotlivým případům užití.

3.2.1 (UC.1N) – Nepřihlášený uživatel

UC.1N.01 – Přihlášení studenta. Nepřihlášený uživatel má přístup na stránku pro přihlášení do studentské části aplikace. Zobrazí se mu tlačítko „Přihlásit přes ORION OIDC“ pro přihlášení pomocí služby třetí strany. Po kliknutí na tlačítko je přesměrován na univerzitního jednotného přihlašování. Na stránce jednotného přihlašování uživatel vyplní své údaje a odešle je. Při úspěšném přihlášení je student

přesměrován zpět do aplikace OKS-WEB-GARANT-APP, která ho přihlásí a zobrazí mu stránku s jeho hodnocením. Při neúspěšné autentikaci je uživatel přesměrován zpět do aplikace na stránku pro přihlášení, kde je zobrazena chybová hláška o neúspěšné akci.

UC.1N.02 – Přihlášení do garanské části. Nepřihlášený uživatel má přístup na stránku pro přihlášení do garantské části aplikace. Na této stránce je uživateli zobrazen formulář pro vyplnění přihlašovacích údajů konkrétně „Přihlašovací jméno“ a „Heslo“. Také stránka obsahuje tlačítko „Přihlásit přes ORION OIDC“. Uživatel má možnost zvolit si způsob přihlášení. Po úspěšném přihlášení je uživatel přihlášen dle přiřazené role a přesměrován na úvodní stránku garantské části aplikace.

3.2.2 (UC.2P) – Přihlášený uživatel

UC.2P.01 – Odhlášení uživatele. Přihlášený uživatel má k dispozici tlačítko signalizující akci odhlášení uživatele. Uživatelským zmáčknutím tlačítka je uživatel odhlášen a přesměrován na stránku přihlášení konkretizovanou dle role, ke které byl přihlášen.

3.2.3 (UC.3S) – Student

UC.2P.01 – Zobrazení celkových informací o semestru.

Přihlášený uživatel role Student na stránce s hodnocením vidí údaje agregované z celého semestru. Je zde zobrazen počet získaných bodů, počet penalizačních bodů a počet celkově obdržených bodů. Taktéž se zobrazuje informace o nároku na zápočet.

UC.2P.02 – Prohlížení informací o hodnocení jednotlivých úloh.

Student na stránce s hodnocením vidí přehled všech úloh se stavy a údaji o jejich odevzdání a hodnocení. Studentovi jsou zde také vypsány získané zpětné vazby připojené k hodnocení.

UC.2P.03 – Volba rozsahu odevzdání práce pro jednotlivé úlohy.

Student na stránce s hodnocením u informací k jednotlivým má dostupné zaškrtávací políčko „Rozsah PMIN“. Tento vstup je aktivní pokud uživatel dosud danou úlohu nemá zkontrolovanou garantem a jeho hodnotou student volí rozsah odevzdávané úlohy (zaškrtnuté odpovídá odevdáním úlohy v rozsahu PMIN a tedy s nižším ziskem bodů).

3.2.4 (UC.4G) – Garant

Skupina funkcionalit určených pro uživatelskou roli Garant. Předpokládáme tedy v případech užití přihlášeného uživatele s rolí minimálně Garant.

3.2.4.1 (UC.4G.01) – Správa studentů

Skupina případů užití seskupující funkcionality správy studentů.

UC.4G.01.01 – Seznam studentů. Uživatel Garant z libovolné stránky garantské části pomocí hlavní navigace přistoupí k seznamu studentů odkazem „Studenti“. Zobrazí se stránka „Seznam studentů“, kde se nachází karta s tabulkou dat. Tabulka má sloupce „Orion login“, „Jméno“ a „Os. číslo“ a řádky představují datové údaje jednotlivých studentů.

UC.4G.01.02 – Přidání studenta.

Uživatel na seznamu studentů klikne na tlačítko „Přidat studenta“. Zobrazí se modál nebo nová stránka „Přidání studenta“ obsahující formulář s položkami „Login“, „Křestní jméno“, „Příjmení“, „Email“, „Os. číslo“. Po vyplnění údajů a stisknutí tlačítka „Uložit“ se uživatel dostává na stránku „Seznam studentů“, aplikace vytvoří nového studenta a uloží ho.

UC.4G.01.03 – Importování studentů.

Aplikace umožňuje přidávat záznamy studentů hromadnou formou vložením CSV souboru exportovaného ze systému IS/STAG. Tento výpis obsahuje veškeré informace, které systém vyžaduje. Uživatel při vkládání souboru tedy vyplňuje již pouze informaci do které skupiny student patří.

UC.4G.01.04 – Upravení studenta.

Vytvořeného studenta lze upravovat přistoupením skrz tabulku se seznamem studentů a kontextovou nabídku, ve které se nachází tlačítko „Upravit“. Zobrazí se formulářové zobrazení, kde uživatel vidí aktuální informace a může je měnit. Kromě informací shodujících se s formulářem pro přidávání studentů se zde vyskytují i formulářové pole pro specifikaci identifikátorů uživatele „GitLab User Id“ a projektu „GitLab Project ID“ systému GitLab.

UC.4G.01.05 – Inicializace studentova projektu.

Před zahájením odevzdávání je potřeba inicializovat studentův repozitář na platformě GitLab. Tuto volbu v případě pokud uživatel dosud nemá přiřazen inicializovaný projektu lze spustit kontextovou nabídkou v tabulce seznamu studentů tlačítkem „Inicializace“.

UC.4G.01.06 – Odstranění studentova projektu.

Studentský repozitář lze taktéž smazat, pokud existuje, využitím kontextové nabídky v seznamu studentů tlačítkem „Odstranit projekt“.

UC.4G.01.07 – Zobrazení prostředí pro nasazení.

Každý student s inicializovaným repozitářem má vytvořené produkční prostředí, kam se nasazují artefakty úloh propojené s dashboardem ve stejném umístění. Na tento dashboard se lze prokliknout ze seznamu studentů a kontextové položky „Nasazení“.

3.2.4.2 (UC.4G.02) – Správa úloh

Pro zvýšenou variabilitu a upravitelnost systému je nastavení a definice úloh přístupná vyučujícímu. Vyučující díky tomuto může například vytvářet nové úlohy, upravovat ty stávající, měnit požadavky na splnění úlohy a stanovit bodové hodnocení.

UC.4G.02.01 – Seznam úloh.

Tabulka se seznamem úloh, které studenti mají za úkol plnit. Zobrazuje uživatelské roli „Garant“ potřebné informace, které potřebuje znát k identifikaci úlohy, které si sám nastavuje tzn. sloupce „Zkrácený název“, „Název“, a sloupce s definicí bodových základů pro oba rozsahy odevzdané práce. Tabulka obsahuje též proklik na úpravu každé úlohy a tlačítko pro definování nové. Funkčně ještě tabulka poskytuje možnost změnit pořadí úloh, čímž se změní zobrazované výpisy v celé aplikaci.

UC.4G.02.02 – Přidání úlohy.

Vyučující v aplikaci vytváří úlohy, které studenti při svém studiu plní. Při tvorbě nové úlohy jsou vyžadovány standardní údaje jako „Název“, „Termín odevzdání“, bodové hodnocení pro jednotlivé rozsahy pro potřeby aplikace, ale potřeba také správně propojit automatickou kontrolu v systému GitLab. To je zajištěno sadou nastavení jako název odpovídající testovací stage a výpisem názvů identifikátorů jednotlivých jobs, které musí v rámci pipeline být splněny, aby byla úloha označena jako úspěšně automaticky zkontrolována.

UC.4G.02.03 – Úprava úlohy.

Úprava využívá formulář pro přidání úlohy a vložené hodnoty uloží do upravované hodnoty. Změněné hodnoty se projeví od chvíle uložení pro všechny další nadcházející použití entity úlohy.

3.2.4.3 (UC.4G.03) – Správa skupin

UC.4G.03.01 – Seznam skupin.

Systém na stránce „Seznam skupin“ vypisuje skupiny z platformy GitLab, které získává přes implementovanou komunikaci s ní. Uživatel tak vidí aktuální data ze systému a může s nimi dále v aplikaci pracovat. Tabulka s výpisem obsahuje identifikátor a název skupiny. Taktéž lze vyvolat akce nad skupinou jako úpravu a nastavení jako aktuální skupiny.

UC.4G.03.02 – Přidání skupiny.

Uživatel na stránce „Seznam skupin“ může přidat novou skupinu tlačítkem „Přidat skupinu“. Zobrazí se formulář s poli „Název“ a „URL Identifikátor“. Při uložení se spustí požadavek do systému GitLab, aby vytvořil definovanou skupinu v základní skupině pro výuku předmětů.

UC.4G.03.03 – Úprava skupiny.

Při úpravě skupiny se zobrazí formulář stejný jako při jejím vytváření, kde jsou pole vyplněna hodnotami aktuálně upravované skupiny. Uživatel může libovolné hodnoty upravovat a následně uložit. Uložená změna se projeví i v systému GitLab.

UC.4G.03.04 – Nastavení aktuální skupiny.

Vyučující má možnost nastavit zvolenou skupinu jako aktuální, umožní aplikaci tímto způsobem zobrazovat data aktuální pro zvolený ročník

3.2.4.4 (UC.4G.04) – Detail studenta

UC.4G.04.01 – Zobrazení pipeline běhů.

Na stránce detailu studenta se nachází tabulka „Seznam pipeline projektu studenta“, obsahující aplikací přijatá data z pipeline ke studentovo projektu. Nachází se zde sloupce jako například „Status“, datum a čas vytvoření a dokončení pipeline a délka trvání běhu pipeline. Tabulka současně obsahuje odkaz, který uživatele přesměruje do platformy GitLab na odpovídající detail pipeline.

UC.4G.04.02 – Zobrazení hodnocení úloh studenta.

Stránka detailu studenta obsahuje výpis jednotlivých úloh a stav studenta k jejich odevzdání. Tabulka obsahuje podrobnější informace jako bodové hodnocení, indikace přítomnosti problému, rozsah, datum a čas poslední kontroly a prvního úspěšného odevzdání. Tabulka obsluhuje i funkce pro opětovné spuštění kontroly a otevření hodnotícího zobrazení pro konkrétní úlohu.

UC.4G.04.03 – Zpracování běhu pipeline. Vyučující má možnost znovu spustit zpracování úlohy dostupného běhu pipeline studenta. Využití této funkce je například při změně podmínek hodnocení úlohy. Zpracování lze spustit tlačítkem „Zpracovat“ u konkrétního běhu v tabulce „Seznam pipeline projektu studenta“.

3.2.4.5 (UC.4G.05) – Hodnocení úloh

UC.4G.05.01 – Zobrazení stavu úloh jednotlivých studentů. Aplikace zobrazuje vyučujícímu tabulkový výpis studentů s vizualizací jeho stavu odevzdání jednotlivých úloh a jejich celkovým hodnocením. Tabulku lze filtrovat a stránkovat.

UC.4G.05.02 – Zobrazení podrobného hodnocení úlohy.

Kliknutím na buňku v tabulce plnění úloh se zobrazí pohled detailu úlohy, ve kterém se nachází detailní informace o úloze a odevzdání (např. datum a čas odevzdání, výsledek automatické kontroly a odkaz na artefakt úlohy studenta). Součástí detailu odevzdání je taktéž zobrazení hodnocení.

UC.4G.05.03 – Zadání bodového hodnocení.

Zadání bodového hodnocení probíhá prostřednictvím komplexního formuláře na bázi kalkulačky, kde lze vyplnit rozsah hodnocení (výběr mezi „PMIN“ a „DOP“), penalizaci za pozdní odevzdání nebo vlastní celkové bodové hodnocení. Po uložení se uloží bodové hodnocení a změní se stav odevzdání úlohy.

UC.4G.05.04 – Úprava bodového hodnocení.

Při úpravě bodového hodnocení již není dostupné zadávání formou komplexního formuláře jako při přidávání nového hodnocení. Namísto toho je zde jednoduché číselné pole s výchozí hodnotou aktuálně uděleného počtu bodů pro zadání číselného hodnocení a volbou rozsahu mezi „DOP“ a „PMIN“, který však již neovlivňuje výslednou hodnotu bodů.

UC.4G.05.05 – Přidání komentáře k hodnocení.

Vyučující má možnost přidat slovní hodnocení ke každé úloze. Tato funkci lze najít v zobrazení hodnocení v části „Zpětná vazba“. Zde se nachází formulář pro zadání zpětné vazby. Lze zvolit mezi typem komentáře a problémem a zadat textový obsah zprávy. Odesláním formuláře tlačítkem „Přidat“ se slovní komentář uloží.

UC.4G.05.06 – Odstranění komentáře k hodnocení.

V zobrazení hodnocení v části „Zpětná vazba“ se ve výpisu jednotlivých komentářů u každého z nich nachází ikona odpadkového koše sloužící jako tlačítko pro odebrání zpětné vazby.

3.2.4.6 (UC.4G.06) – Nastavení aplikace

UC.4G.06.01 – Nastavení GitLab skupin.

Aplikaci je možno parametrizovat dle aktuálních potřeb, proto nabízí možnosti pro nastavení skupin přes uživatelské rozhraní. Garant předmětu má k dispozici formulář nastavení GitLab skupin, kde může měnit nastavení identifikátorů aktuální skupiny, mateřské skupiny pro výuku předmětů a identifikátor základního repozitáře pro kopírování na studentské projekty.

UC.4G.06.02 – Nastavení bodových parametrů.

Nastavení obsahuje taktéž bodového hodnocení. Ve formuláři „Nastavení penalizačních bodů“ lze nastavit počet bodů za jednotlivé varování (první až třetí) a limit počtu bodů pro nárok na zápočet. Nastavené hodnocení se bude od uložení projevovat pro nové hodnocení studentů.

3.2.5 (UC.5R) – SuperAdmin

3.2.5.1 (UC.5R.01) – Správa uživatelů

UC.5R.01.01 – Seznam uživatelů.

Uživatelská role SuperAdmin má zobrazit seznam uživatelů, kteří se mohou do aplikace (do garantské části) přihlásit a operovat v ní. Pohled se seznamem uživatelů poskytuje také možnosti pro vytvoření nového uživatele a úpravu stávajícího.

UC.5R.01.02 – Vytvoření uživatele.

Uživatel SuperAdmin má možnost vytvářet nové uživatele do systému. Při tom volí jejich uživatelské jméno, typ přihlašování a údaje k přihlášení. Kromě toho též volí roli v systému pro specifikování pravomocí v aplikaci.

UC.5R.01.02 – Úprava uživatele.

Pro úpravu uživatele slouží formulář stejný jako pro vytvoření uživatele, zde přistupující osoba upravuje již existující instanci uživatele, u kterého má možnost změnit heslo nebo účet deaktivovat.

3.2.6 (UC.6C) – CLI

UC.6C.01 – Zpracování běhů pipeline z fronty.

Úloha aplikace pro automatické zpracování běhů pipeline optimalizovaná pro spuštění z automatického spouštěče. Funkce získá data z fronty a proběhne zpracování každého běhu.

3.2.7 (UC.7A) – API

UC.7A.01 – Poskytování stavů odevzdání studenta.

Aplikace poskytuje přístupový bod, který pro platné volání vrátí dostupné stavy jednotlivých úloh pro zobrazení v dashboardu na produkčním prostředí každého studenta.

UC.7A.02 – Přijímání notifikačních dat.

Pro přijímání webhook dat ze systému GitLab, aplikace poskytuje zabezpečené místo pro volání této služby. Přijatá data jsou uložena do fronty pro budoucí zpracování.

3.3 Vztahy mezi entitami

Návrh popisuje klíčové vztahy mezi hlavními entitami určujícími studentův postup a jeho hodnocení.

3.4 Návrh uživatelského rozhraní

Přednesený návrh řešení byl pro oboustrannou shodu prezentován formou wireframů, které udávají přibližnou podobu rozvržení finálního produktu a ilustrují funkčnosti budoucího systému. Myšlenkou tohoto kroku je sjednocení představ o vzhledu a funkčnosti konečné aplikace mezi zadavatelem (budoucím hlavním uživatelem) a vývojářem, v roli návrháře, ještě před samotným vývojem. Benefity tohoto přístupu pramení ze skutečnosti, že je snadnější a rychlejší (a komerčně tedy i levnější) měnit požadavky na produkt v jeho návrhové části (resp. nejrannější fázi projektu) ještě před zahájením vývoje, než v implementační fázi nebo v téměř dokončeném stavu.

Forma sjednocení pohledů na systém může být různá. Jednou z nich jsou právě zde prezentované wireframes (česky drátěné modely), které jednoduchým vyznačovacím způsobem pomocí čar a jednoduchých geometrických objektů ukazují pozici různých komponent a dodatečnými popisky představujících jejich informační schopnosti ve vyvíjené aplikaci. Kromě tohoto modelu mohou být použití další metody [Cur10]:

- Grafický návrh – Návrh je zpracován vizuální a často třeba i interaktivní formou vytvořením grafického prototypu bez dodané funkčnosti systému. Mezi takové návrhy můžeme zařadit například vytvoření prototypu uživatelského rozhraní pomocí HTML a CSS nebo zvoleným specializovaným nástrojem jako je například Figma nebo Adobe XD.

- Textový popis – Návrh uživatelského rozhraní je podrobně popsán za pomoci jazykových prostředků. Zde je potřeba klást zvýšený důraz na ujištění sjednocených představ všech stran. Zároveň popis musí být dostatečně detailní a jasný, aby omezil možnosti chybné interpretace ve fázi vývoje.
- Kombinace – Návrh může být libovolnou kombinací výše zmíněných. Návrh může být sestaven libovolnou formou, která vyhovuje oběma stranám a jejich volba se nemusí omezovat pouze na zmíněné metody.

Zmíněné modely mohou být různě podrobné a detailní v závislosti na potřebách zúčastněných stran. Na základě míry detailu popisu může návrh zmínit i konkrétní implementační mechanismy (např. použití specifického HTML elementu).

3.4.1 Ukázky pohledů

V rámci návrhu aplikace pro správu předmětu OKS byly identifikovány klíčové pohledy pro uživatelské role Student a Garant, jejichž některé ukázky jsou uvedené v následující části.

3.4.1.1 Studentský přehled

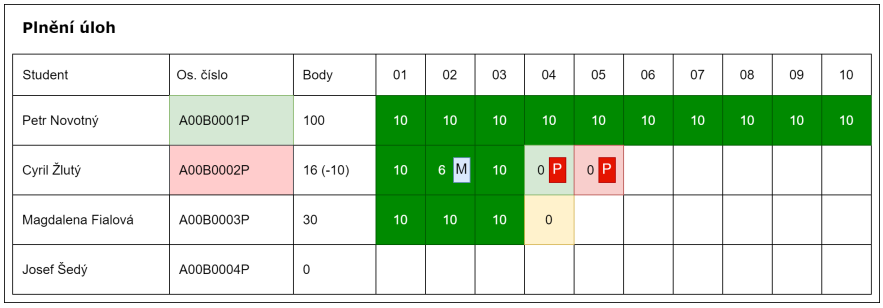
Studentský pohled přebírá zodpovědnost zobrazit v jednoduché formě a na jedné stránce všechny informace, které studenta zajímají. V navrženém pohledu (viz obr. 3.4) je zpracováno rozložení do tří sekcí: celkové hodnocení, statistika a údaje k jednotlivým úlohám a zobrazení zpětné vazby. V první sekci celkového hodnocení se student dozvídá hlavní parametry svého dosavadního výsledků, tedy agregované bodové hodnocení a zda má nárok na zápočet. V druhé sekci jsou již uvedeny detailnější informace seskupené do tabulky. Ke každé úloze je například uveden stav odevzdání, datum odevzdání a počet obdržených bodů. Zároveň zde student formou zaškrtávacího tlačítka u každé úlohy volí rozsah odevzdání mezi PMIN a DOP. Poslední sekce se zabývá informací o zpětné vazbě od vyučujícího ke studentovi pro konkrétní úlohu. Z důvodu zobrazení delšího textu není tato zpětná vazba součástí tabulky s podrobným hodnocením, takže jsou záznamy vypisovány samostatně na konci stránky.

3.4.1.2 Přehled vyučujícího na plnění úloh studentů

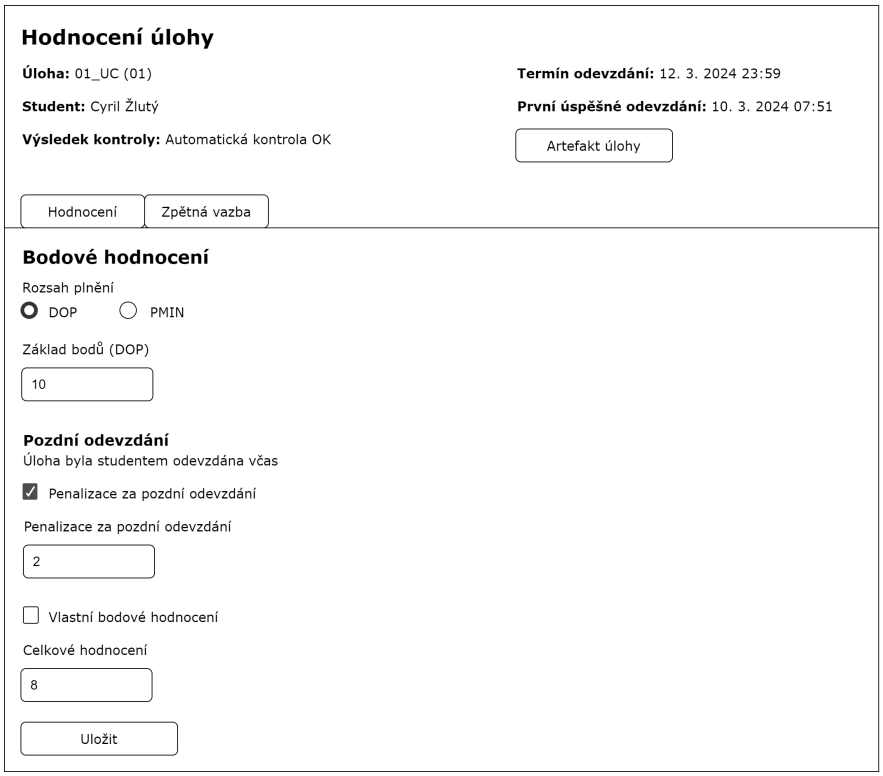
Mezi hlavní pohledy uživatelské role Garant patří celkový souhrn zobrazení studentů s jejich postupem odevzdávání úloh. Jedná se o klíčovou část aplikace, která musí vizualizovat aktuální stav s informacemi v zhuštěné míře dle požadavků zadavatele. Přehled se sestává z tabulkového seznamu studentů se sloupci (01 až 10) odpovídající každé úloze (viz obr. 3.5). Každá buňka svým podbarvením stanovuje

Obrázek 3.4: Wireframe návrhu studentského pohledu (zdroj: vlastní)

stav odevzdání (např. tmavě zelená odpovídá úspěšnému odevzdání s přiřazenými body nebo žlutá signalizuje pozdní odevzdání). Vnitřek buňky stanovuje počet získaných bodů (případně 0, pokud hodnocení není dosud uděleno) a konkretizuje detaily odevzdání jako rozsah práce nebo zpětnou vazbu typu „problém“. Buňky ve sloupci „Os. číslo“ se podbarvují v závislosti na nároku na zápočet (zelená) nebo dle počtu nalezených problémů (odstíny červené). V návrhu se počítá s použitím dynamických tabulek podporující filtrování a stránkování.



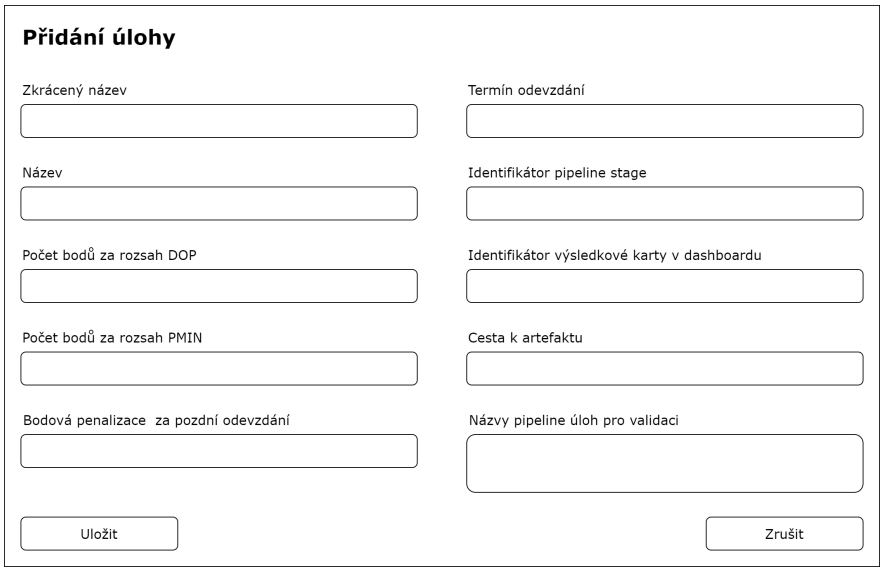
Obrázek 3.5: Wireframe přehledu plnění studentských úloh studenty (zdroj: vlastní)



Obrázek 3.6: Wireframe zadání hodnocení (zdroj: vlastní)

3.4.1.3 Hodnocení studenta

Vyučující má možnost udělit bodové hodnocení každé odevzdané studentské úloze. Zobrazení (viz obr. 3.6) se dělí na tři části, v první se zobrazují obecné informace o odevzdání jako datum a čas odevzdání s jeho výsledkem, název úlohy, název studenta a odkaz na artefakt úlohy. Zobrazení také upozorní vyučujícího, že úloha byla odevzdána po standardním termínu odevzdání. Druhá část se zabývá samotným zadáváním hodnocením, které se uděluje formou bodového součtu vycházející z bodového základu dle zvoleného rozsahu plnění. Následně se od této báze odečítá počet penalizačních bodů za pozdní odevzdání, kdy je vyučujícímu umožněno dodatečně penalizaci započítat. Celý formulář funguje jako kalkulačka, výsledek bodového hodnocení na základě zvolených hodnot se dynamicky mění a zobrazuje aktuálně udělované bodové hodnocení. Pokud má vyučující důvod zadat jiné hodnocení, je mu to umožněno deaktivací „kalkulačkového režimu“ a zvolit si vlastní hodnotu udělovaného počtu bodů. Pokud je již hodnocení uděleno,Případně je umožněno upravovat bodové hodnocení i zpětně, ale již není dostupný interaktivní režim kalkulačky. Poslední třetí část zobrazení se pak týče zpětné vazby. Vyučující zde přidává komentáře nebo indikuje detekovaný problém s potřebným popisem. Případně v případě chybného zadání zpětné vazby je umožněno je v jejich výpisu smazat.



Obrázek 3.7: Wireframe zadání hodnocení (zdroj: vlastní)

3.4.1.4 Definice úlohy

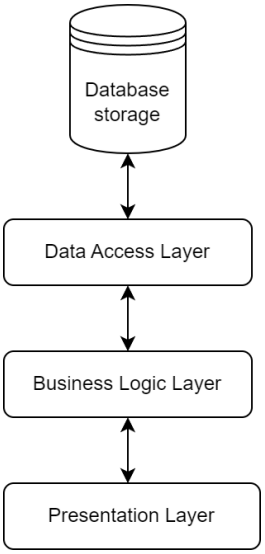
Aplikace je navržena, aby byla dostatečně odolná vůči možným změnám ve výuce. Je proto možné jednotlivé úlohy definovat včetně propojení se zpracováváním úloh GitLabu. Formulář pro vytváření a editaci zadání (viz obr. 3.7) poskytuje vstupy pro základní informace o úloze jako název, zkrácený název a standardní termín odevzdání. Dále je možno nastavit vlastní hodnoty pro bodování této úlohy pro jak pro různé rozsahy PMIN a DOP, tak penalizační body pro pozdní odevzdání. V souvislosti s propojením s dalšími navázanými systémy a pro správnou validaci se nastavuje název úrovně (stage) pipeline, jednotlivé názvy pipeline úloh, které musí být pro splnění úlohy při svém dokončení úspěšné, identifikátor výsledkové karty ve studentském dashboardu a relativní cesta k artefaktu.

3.5 Architektura aplikace

3.5.1 Vrstvená architektura

Strukturování softwarových systémů na několik vrstev je běžnou technikou vedoucí k oddělení zodpovědnosti jednotlivých vrstev, modulární designu a znovupoužití. Technika vrstvení spočívá v definování jasných pravidel oddělení vrstev a stanovení rozhraní resp. kontraktů komunikace mezi jednotlivými vrstvami. Způsob jakým se bude systém vrstvit je vždy v závislosti na konkrétním projektu stejně tak jako počet vrstev a jejich zodpovědnost [Ric15].

Třívrstvá architektura je v softwarovém inženýrství běžně využívaným designovým vrstevným směrem organizace aplikace a zejména u webových aplikací. Tento architektonický styl rozděluje aplikaci na tři nezávislé celky dle datového modelu, řídící logiky a prezentaci dat (viz obr. 3.8). Každý z těchto celků lze chápat jako vrstvu, která myšlenkově obstarává úlohu z jiné úrovně a s jinou zodpovědností.



Obrázek 3.8: Diagram vrstvené architektury (zdroj: vlastní)

- **Datová vrstva** – Skládá se například z nějakého mechanismu pro persistenci dat (např. databáze, souborový systém, API) a vrstvy datového přístupu, která implementuje rozhraní pro přístup k datům nezávisle na aplikační vrstvě.
- **Řídící vrstva** – Nebo taktéž aplikační vrstva, či Business logic layer je logickou vrstvou mezi prezentační a datovou vrstvou. Její zodpovědností je zpracovávání aplikačních funkcionalit, které tvoří logiku celé aplikace.
- **Prezentační vrstva** – Jedná se o nejvyšší stupeň zpracování v aplikaci, kdy se vrstva stará o prezentaci dat uživateli, nebo reprezentaci dat od uživatele.

Cíl rozdělení aplikace do více vrstev vychází z principu rozdělení odpovědností (Separation of Concerns - SoC), který popisuje aplikaci jako systém služeb určité zodpovědnosti. Myšlenkou SoC je zajištění, by se zodpovědnosti mezi jednotlivými službami co nejméně překrývaly, každá se starala pouze jí přidělenou funkcionalitu a aby jednotlivé vrstvy mezi sebou komunikovaly skrze rozhraní [MEM04].

Využívání vrstvené architektury můžeme pozorovat na různých částech aplikací a v různých formách od rozdělení závislostí pomocí adresářové struktury, modulů, nebo v menších projektech i pouze jednotlivými logickými úseky kódu. Díky tomuto

oddělení je kromě jiného umožněno vývojářům zaměřovat se na svoji vývojovou oblast přičemž se vylepšuje paralelizace úkonů a zjednodušuje se přidávání nových funkcionalit.

Od takto zobecněné vrstvené architektury vychází řada dalších designových architektur, které interpretují vrstevnost s mírnými rozdíly [SW14].

- **Model-View-Controller (MVC)**
  - Model (Model) – Reprezentuje data a logiku aplikace jako součást nezávislá na uživatelském prostředí.
  - View (Pohled) – Slouží k reprezentaci prvků uživatelského rozhraní, kdy zobrazuje data uživateli a předává uživatelské akci kontroleru.
  - Controller (Kontrolér) – Prostředník mezi modelem a pohledem. Jeho zodpovědností je přijímání uživatelských vstupů, jejich zpracování, případnou aktualizaci modelu a pohledu.
- **Model-View-ViewModel (MVVM)**
  - Model (Model) – Stejně jako u MVC reprezentuje data a logiku aplikace jako součást nezávislá na uživatelském prostředí.
  - View (Pohled) – Obdobně jako u MVC, zobrazuje prvky uživatelského rozhraní
  - ViewModel (Zobrazovaný model) – Také funguje jako prostředník mezi modelem a pohledem vytvářením datových vazeb mezi nimi a upozorněními na změnu stavu prostřednictvím událostí.
- **Model-View-Presenter (MVP)**
  - Model – Stejně jako u MVC reprezentuje data a logiku aplikace jako součást nezávislá na uživatelském prostředí.
  - View (Pohled) – Podobně jako u MVC, prezentuje prvky uživatelského rozhraní
  - Presenter (Prezentér) – Přebírá požadavky uživatele z pohledu a aktualizuje model. Narozdíl od MVC se o aktualizaci pohledu stará přímo prezentér.

Volba konkrétního architektonického stylu vychází vhodnosti aplikování na potřeby a kontextu každého projektu. Takové rozhodnutí vychází z role softwarového architekta, který uváží veškeré potřeby a další závislosti pro konkrétní projekt.

3.5.2 SOLID principy

S designem softwarového projektu souvisí i řada dalších doporučení jak organizovat strukturu a vést vývoj k čitelnému a čistému kódu. Mezi takové principy patří například soubor obecných zásad pojmenovaný SOLID (nebo též SOLID principy), který seskupuje 5 rad použitelných na téměř jakýkoli softwarový projekt [Mar00]:

- **Single Responsibility Principle (Princip jedné odpovědnosti)** – Třída by měla mít zodpovědnost za právě jednu věc vystiženou jejím názvem. Porušením tohoto principu vede k obsáhlým třídám, které jsou zodpovědné za spoustu záležitostí. Tím jsou nepřehlednější, složitější, mají častokrát mnoho závislostí. Tímto doporučením se vede kód, který má sice více tříd ve své struktuře, ale mají podstatně méně metod a tím jsou jednodušší.
- **Open-Closed Principle (Princip otevřenosti a uzavřenosti)** – Princip poukazuje na otevřenost softwarového kódu pro rozšíření a uzavřenost pro jeho úpravy. Třídy by měly být psány způsobem, aby bylo možné přidávat funkcionality psaním nového kódu bez nutnosti upravovat již stávající kód. Implementačně je tohoto docíleno využitím dědičnosti, rozhraními, abstrakcí a polymorfismem.
- **Liskov Substitution Principle (Liskovové princip zastoupení)** – Instance podtříd by měly být schopny nahradit instance svých nadřazených tříd aniž by došlo k porušení funkčnosti programu. Princip zaměnitelnosti úzce souvisí s dědičností a polymorfismem a zajišťuje jak správně tyto vlastnosti používat [24g].
- **Interface Segregation Principle (Princip oddělení rozhraní)** – Závislost pouze na takových rozhraních, které jsou využívány. Tento princip se snaží omezit obsáhlá rozhraní plné částí, které nejsou zapotřebí, ale zatěžují klienta implementací. Řešením je využíváním menších, ale konkrétnějších rozhraní více přizpůsobeným potřebám klienta. Tím je zmírněno zatížení klientů zbytečnými metodami a závislostmi.
- **Dependency Inversion Principle (Princip obrácené závislosti)** – Moduly nebo třídy vyšších úrovní abstrakce by neměly záviset na těch z nižších úrovní. Implementačně je toho docíleno způsobem, že závislosti jsou zaměřeny na abstraktní třídy nebo rozhraní a nikoli na konkrétní implementaci. Tento přístup přináší spoustu výhod jako například redukci závislostí kódu, nezávislost na úpravě implementace konkrétní funkcionality a celkově podporuje flexibilitu, údržbu a testovatelnost.

Je potřeba vzít na vědomí, že tyto principy jsou velmi obecné a téměř nikdy je nelze aplikovat v úplném rozsahu. Avšak již maximalizací jejich využívání přináší vede ke kvalitnějšímu návrhu.

3.6 Evaluace

Procesem evaluace studentských prací se z pohledu aplikace dělí na několik podprocesů počínaje získáním dat z různých zdrojů přes jejich zpracování až po vizualizaci.

3.6.1 Získávání dat z GitLabu

3.6.1.1 Notifikování pomocí Webhooks

Návrhem je zvoleno, že aplikace bude přijímat notifikační služby webhooks ze systému GitLab. S tím se pojí jistá omezení na implementaci následného zpracování např. rychlost odezvy aplikace na webhook notifikaci. Z toho důvodu je navrhováno, aby se přijatá data uložila do fronty, ze které je aplikace bude periodicky číst a zpracovávat. Výhodou použití webhooks je notifikování právě v momentu, kdy se nastavená událost přihodí. To je hlavní výhodou oproti způsobu dotazování pomocí REST API nebo GraphQL přičemž nevíme kdy se událost stala a dotazování v moment požadavku na data vy znamenala stahování a zpracovávání velkého množství dat, které snižují efektivitu aplikace.

V systému GitLab je navrženo nastavení skupinové nebo repozitářové (dle možností) notifikační služby Webhooks, který bude POST požadavkem odesílat na připravený přístupový bod směřující do API modulu informace o nastalé události. Aplikace v případě validního požadavku perzistentně uloží přijatá data do fronty a systému GitLab odpoví zprávou s kódem úspěšné operace.

K ukládání do fronty bylo přistoupeno z důvodu časového limitu na odpověď na získaná data. Je zde přítomna obava, že v časech vyššího vytížení systému by aplikace při získaných datech je nestíhala zpracovávat do stanoveného časového limitu.

Pro další zpracování postupů se notifikuje zejména informace o dokončení běhu pipeline. Takto získaná data obsahují kromě identifikačních údajů i seznam spuštěných úrovní (stage) pipeline (viz obr. 3.1) a výsledky o jednotlivých pipeline úlohách.

Zdrojový kód 3.1: Část webhook payload se seznamem spuštěných úrovní (zdroj: vlastní)

```
1     ...
2     "stages": [
3     "WEB_HTML_BUILD",
4     "WEB_HTML_DEPLOY",
5     "05_RFA",
```

```
    "docker_build",
    "deploy"
    ],
    ...

```

Když se podíváme na ukázku ze seznamu proběhlých pipeline jobs (viz obr. 3.2), obsahují důležité informace jako `stage` a `name` podle kterých můžeme identifikovat a přiřadit je ke konkrétní úloze v aplikaci. A zároveň důležitou hodnotou je `status`, která ukazuje výsledek proběhlého pipeline úlohy, tu využijeme při evaluaci výsledku úloh. Tento status může nabývat těchto hodnot, pro nás jsou však důležitá hodnota „success“, která sděluje úspěšné dokončení. Ostatní stavy budeme považovat za neúspěšně dokončenou úlohu „failed“.

Zdrojový kód 3.2: Část webhook payload se seznamem proběhlých jobs (zdroj: vlastní)

```
    ...
    "builds": [
    ...
    {
      "id": 2243,
      "stage": "05_RFA",
      "name": "05-robot",
      "status": "success",
      "created_at": "2024-03-05␣15:01:49␣UTC",
      "started_at": "2024-03-05␣15:03:24␣UTC",
      "finished_at": "2024-03-05␣15:04:00␣UTC",
      "duration": 35.797567,
      "queued_duration": 18.728052,
      "failure_reason": null,
      "when": "on_success",
      "manual": false,
      "allow_failure": true,
      "user": {
        "id": 35,
        "name": "Jan␣Hinterholzinger",
        "username": "hintik",
        "avatar_url": "https://secure.gravatar.com/avatar/
d5b4e0c8f0eec2fdb6c698805467dcd4397762f64ceecd66ddaff92695c7e750
?s=80&d=identicon",
        "email": "[REDACTED]"
      },
      "runner": {
        // informace o~běžícím prostředí GitLab Runner
      },
      "artifacts_file": {
        "filename": "artifacts.zip",
```

```
30          "size": 242302
31        },
32        "environment": null
33      },
34      ...
35      ]
36      ...
37
```

3.6.1.2 Dotazování rozhraní API

V aplikaci jsou takové případy užití, pro které je potřeba získávat data ze systému GitLab i jinou cestou než notifikováním v okamžiku, kdy se stane nějaká událost. To by aplikace musela sbírat veškeré informace a udržovat si aktuální stav oproti systému GitLab, který by se jen těžko dokázal synchronizovat. Proto pro tyhle účely, které se většinou skládají z úkonů vyžadující menší množství prakticky nezpracovaných dat je vhodné se na ně dotázat ve chvíli, kdy jsou potřeba. Takové případy užití jsou například skupinou 3.2.4.3, kde výpis skupin lze implementovat v rámci datové vrstvy namísto dotazování do databázového serveru tak přes API volání na server GitLab. Obdobným způsobem mohou být implementována i další přístupy využívající GitLab API. Zde se nabízí využití dotazovacích možností GraphQL nebo REST API. Ve fázi návrhu je volba finálního použití API ponechána na jako implementační detail, který je následně v aplikaci odstíněn vrstvou datového přístupu. Avšak doporučuje se využít spíše rozhraní REST API, které je svou formou méně náročné a plánovaný nízký počet využití dotazovacího způsobu získávání dat a je považováno jako bezkonkurenčně výhodnější pro budoucí implementaci.

3.6.2 Zpracování dat

Součástí návrhu je implementace samostatného procesu, který bude periodicky získávat data z fronty a v aplikaci zpracovávat. Tento samostatný proces může být například formou naplánovaného příkazu z CLI pomocí služby CRON nebo jiného podobného nástroje.

Zpracování probíhá v následujících krocích:

1. **Získání dat o úlohách** – Aplikace získá podrobnosti o úlohách včetně informací o propojení se systémem GitLab jako například k úloze odpovídající název stage a potřebné názvy jobs, které musí být splněny v dané stage.

2. **Preparace výsledků z běhu pipeline** – Zpracováváním výsledků jobs se získávají data o jejich jednotlivých výsledcích.

3. **Automatická evaluace** – Získaná data se porovnají s požadavky na jednotlivé úlohy a kontroluje se, zda jsou splněny. Tímto krokem je dokončeno zpracování automatické kontroly.

4. **Ruční evaluace** – Vyučují na základě vizualizace dat a následné manuálního zhodnocení udělí finální bodové hodnocení úlohy.

3.6.3 Automatická evaluace

Významným bodem celé práce je krok automatické kontroly a propagace výsledků do vyvíjeného systému. Jak již bylo řečeno, úlohy jsou kontrolovány v rámci spuštěné pipeline definovanými kontrolami jednotlivých jobs. Tyto kontroly jsou stále v kontextu pipeline organizovány do jednotlivých stage, které odpovídají každé odevzdávané úloze a podle toho jsou spouštěny. Po skončení pipeline je odeslána notifikace ze systému GitLab na aplikační server, který požadavek přijme, zkontroluje veškeré náležitosti a uloží do fronty pro zpracování.

Automatická úloha pro zpracování následně získá z fronty data ze získané notifikace a zahájí samotné zpracování. Cílem je určit které úlohy byly pipeline kontrolovány a určit výsledek dle specifikace nastavené ve vyvíjené aplikaci. V té se řeší propojení úlohy pomocí identifikačních názvů jednotlivých jobs, přičemž v rámci kontroly musí být všechny tyto jobs splněny, aby byly splněny požadavky aplikační úlohy a byla považována za splněnou. Pokud naopak nějaká úloh z definovaných jobs nebyla vykonána nebo nebyla vykonána s úspěšným výsledkem, je celá úloha považována za nesplněnou resp. obdrží stav „pipeline_failed“ sdělující, že automatická kontrola objevila nedostatky.

3.6.4 Vizualizace dat

Pro vizualizační část evaluace je potřeba se držet požadavků zadavatele a respektovat jeho potřeby pro efektivní práci s aplikací. Důležité je, aby aplikace měla dostatečně zpracovaná data, pro okamžité zobrazování výsledků. Je navrhováno využití tabulkového zobrazení s podbarvenými buňkami signalizující stav evaluace jednotlivých úloh.

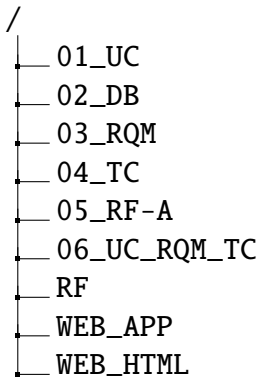
Validační úlohy a artefakty

4

4.1 Návrh validačních úloh pro použití v technologickém stacku

Validační úlohy mají v celém systému zásadní roli. Svými kritérii určují výslednou podobu odevzdávaných úloh. V rámci nově vznikajícího systému jehož technologický stack je součástí jiné práce jsou validační skripty spouštěny jako Python testy v rámci GitLab pipeline. Úkolem této práce je navrhnout validační úlohy, které jsou použitelné ve zmíněném systému a smysluplně automaticky kontrolují odevzdané úlohy.

Technologický stack se sestává ze struktury projektů, které zajišťují prostředí pro odevzdání úloh na univerzitní GitLab instanci, jejich kontrolu a publikování. To je dosaženo prostřednictvím struktury GitLab projektů, které umožňují komplexní nastavení pipeline pro potřeby celého systému. Kontrola validačních úloh je jedním z několika podsystémů a je mu vyčleněn jeden (`images/validation-scripts`) ze zmíněných projektů. Adresářová struktura tohoto projektu je organizována dle jednotlivých odevzdávaných artefaktů, tedy odpovídá jednotlivým úlohám (viz obr. 4.1). Každý adresář obsahuje sadu artefaktů pro pozitivní testy validace a bude obsahovat taktéž soubor sad artefaktů pro negativní testy, který je vytvořen v rámci této práce. V případě potřeby obsahuje taktéž soubory pro zajištění prostředí (typicky Dockerfile). Jak již bylo řečeno, technologický stack předpokládá, že validační skripty jsou psány jako Python testy a jsou uloženy do adresářů v adresářové struktuře jako soubory s názvem `check_<název>.py`. Tyto skripty mají strukturou připravenou podporu pro pozitivní a negativní testy, které ověří správnost validačních skriptů na vlastní sadě příkladů.



Obrázek 4.1: Struktura adresáře pro oddělení validačních skriptů dle úloh (zdroj: vlastní)

4.1.1 Validace úloh

4.1.1.1 Validace úlohy 01_UC

Úloha spočívá ve vytváření případů užití ve formátu XML v kontextu studentovy aplikace. Student získá předem XML strukturu, ve které vyplní pouze určité části. Výsledné soubory jsou poté automaticky v pipeline převedeny do HTML souboru a ty jsou publikovány jako artefakt úlohy. Požadavkem na studenty je vytvoření alespoň stanoveného minimálního počtu případů užití se smysluplným textem.

Návrhem byla sestavena sada validačních úloh pro úlohu `01_UC`:

1. Kontrola na přítomnost validních souborů – Validace kontroluje odevzdaný adresář na přítomnost souborů ve správném formátu. Neodpovídající názvy nebo typy souborů způsobí negativní výsledek validace.
2. Počet vytvořených případů užití – Validace spočte počet vytvořených případů užití a zkontroluje, zda počet splňuje minimální kritéria pro splnění úlohy.
3. Kontrola délky řetězce vložených hodnot – Validační úloha nalezne položky názvu, zkratky a popisu a zkontroluje jejich délku.
4. Kontrola vyplnění hodnot – Skript zkontroluje konzistenci číslování souborů s číslováním v obsahu souboru. Dále se provede validace definice názvu, kontextu, rozsahu, aktéra a popisu případu užití včetně vstupních a výstupních podmínek a postupu k dosažení.

4.1.1.2 Validace úlohy 02_DB

Studenti dle zvolené domény své semestrální práce z WEB a již vytvořených případů užití sestaví strukturu své databáze (1. SQL skript) a naplní ji ukázkovými daty

(2. SQL skript). Následně pro takto sestavenou databázi napíší databázové testy v nástroji Robot Framework (RF). Výsledné soubory uloží do předpřipravené struktury pro odevzdání.

Návrhem byla sestavena sada validačních úloh pro úlohu `02_DB`:

1. Kontrola odevzdaných souborů – Validace zkontroluje přítomnost obou souborů pro inicializaci struktury a pro naplnění dat.
2. Inicializace databáze – Při vytváření struktury databáze se získá počet vytvářených tabulek a počet vkládaných záznamů. Při těchto operacích se tyto hodnoty validují oproti odpovídajícím minimálním počtům
3. Spuštění testů – Validace spustí studentovy testy v nástroji RF, výsledkem této kontroly je i výsledek validace.
4. Kontrola počtu testů – Z výpisu se získá počet proběhlých testů, a ten se srovná se stanovenými minimálními kritérii.

4.1.1.3 Validace úlohy 03_RQM

Úkolem této úlohy je na základě jasné představy o plánovaných funkcionalitách ve své webové aplikaci a vytvořených případů užití vytvořit v nástroji SquashTM seznam členěných RQM pokrývající veškeré UC. Výsledkem je odevzdání exportovaných dat z nástroje SquashTM, který se následně v pipeline analyzuje a konvertuje.

Návrhem byla sestavena sada validačních úloh pro úlohu `03_RQM`:

1. Kontrola odevzdaných souborů – Validace provede kontrolu odevzdaných souborů, zda je odevzdán jejich dostatečný počet ve správném formátu.
2. Pokrytí případů užití – Zkontrolování, zda jsou požadavky pokryty veškeré případy užití.
3. Dostatečný počet kritických požadavků – Kontrola, zda je dosaženo minimální počtu požadavků s kritickou prioritou.

4.1.1.4 Validace úlohy 04_TC

Řešením čtvrté úlohy je přiřazení testovacích případů (TC) k jednotlivým požadavkům (RQM). Pozor si zde studenti musí dát například na odpovídající prioritu v systému SquashTM mezi testovacími případy požadavky. Výstupem je exportovaný JSON soubor z nástroje Squash.

Návrhem byla sestavena sada validačních úloh pro úlohu `04_TC`:

1. Kontrola počtu souborů – V prvotní fázi se zkontroluje počet všech, TC a RQM souborů a validuje se, zda je dosažen jejich minimální počet.
2. Kontrola pokrytí RQM pomocí TC – Validace, zda je každý požadavek pokryt aspoň jedním testovacím případem.
3. Dostatečný počet kritických TC – Kontrola, zda sepsán dostatečný počet kritických testovacích případů vůči požadavkům s kritickou prioritou.

4.1.1.5 Validace úlohy 05_RF-A

Úloha pro vytvoření testování vůči statickým HTML stránkám pro vybrané klíčové případy užití za pomoci nástrojů Robot Framework a Browser Library (RFBL).

Návrhem byla sestavena sada validačních úloh pro úlohu `05_RF-A`:

1. Spuštění automatických testů – Jsou spuštěny automatické testy webového rozhraní.
2. Počet spuštěných testů – Zkontroluje se, zda počet spuštěných testů splňuje minimální požadavky.
3. Splnění testů – Veškeré testy musí být dokončeny s pozitivním výsledkem, jinak je kontrola vyhodnocena jako nesplněná.
4. Pokrytí TC testováním – Veškeré případy užití (TC) jsou pokryty implementovanými testovými případy pomocí RF.

4.1.1.6 Validace úlohy 06_UC_RQM_TC

Úloha spočívá v doplnění již vytvořených případů užití (UC), požadavků (RQM) a testovacích případů (TC) o popis alternativních toků. Sada validačních úloh se sestává z kombinací dříve zmíněných kroků pro validaci typově stejných souborů. Pouze je analogicky stejným stylem doplněna validace nově vyplňovaných položek.

4.1.1.7 Validace úlohy 07_RF

Studenti upravují sadu testů z přechozí úlohy na práci s RF na vyvinuté dynamické aplikaci. Současně jsou implementovány nové dynamické testy v Robot Framework, které ověřují klíčové funkcionality aplikace.

Navržená sada úloh pro úlohu `07_RF` odpovídají sadě z úlohy `05_RF-A` s tím rozdílem, že jsou upravena minimální kritéria.

4.1.1.8 Validace úlohy 08_LOG

Úloha spočívá v začlenění mechanizmů pro logování akcí v dynamické webové aplikaci pro sledování vykonaných operací. Studenti implementují do své vyvíjené aplikace logování za pomocí knihovny Monolog, nastaví výstupní soubory a doplní logovací příkazy o různých úrovních do určených částí své aplikace. Výstupem jsou logovací záznamy členěné dle zadání (např. dle časové hodnoty nebo úrovně událostí).

Návrhem byla sestavena sada validačních úloh pro úlohu `08_LOG`:

1. Kontrola existence logovacích záznamů – Validace přistoupí do adresáře, kde se očekávají výstupní soubory logování a zkontrolují strukturu jejich pojmenování.
2. Počet logovacích záznamů – Analýzou logovacích souborů se ověří, zda počet záznamů dosahuje minimálního počtu a je možné provést další kroky validace.
3. Validace testovacích úrovní – Kontrola analyzuje vytvořené logovací záznamy a ověří použití všech doporučených logovacích úrovní.
4. Zprávy logovacích záznamů – Ověření, zda záznamy logování obsahují smysluplnou (dostatečně dlouhou) zprávu s případně vyplněnými kontextuálními informacemi.

4.1.1.9 Validace úlohy 09_RF

Účelem úlohy je vytvoření komplexních E2E a negativních testů pro plné pokrytí funkcionalit dynamické webové aplikace, za pomoci nástroje RFBL. Výstupem je sada RF testů ověřující funkčnost aplikace.

Návrhem byla sestavena sada validačních úloh pro úlohu `09_RF`:

1. Kontrola existence testů – Kontrola existence souborů o správné struktuře ve specifikovaném adresáři.
2. Spuštění testů – Spuštění E2E testů a získání výsledného logu. Pokud při testování některý z testů selže, je celková validace ukončena jako nesplněná.
3. Kontrola počtu testů – Získání počtu spuštěných testů a ověření splnění minimálních kritérií.

4.1.1.10 Validace úlohy 10_STAT

Úloha se zabývá zlepšování kvality psaného kódu a dodržování standardů za pomoci statické analýzy kódu. Ta je prováděna nástrojem PhpStan a studenti v celém kódu odevzdávané dynamické webové aplikace musí splnit stanovenou úroveň PhpStan validnosti.

Návrhem byla sestavena sada validačních úloh pro úlohu `10_STAT`:

1. Spuštění kontroly – Spuštění nástroje PhpStan s minimální úrovní pro splnění úlohy a na soubory se studentským kódem (nikoli kód knihoven). Pokud nástroj analyzační nástroj PhpStan nalezne nedostatky ve validitě kódu, je ukončen s chybou a stejně tak je ukončena validace.

4.2 Ověření funkčnosti aplikace na existujících artefaktech

Pro účely ověření funkčnosti celého systému byl zadavatelem připraven a dodán soubor studentských artefaktů pro jednotlivé úlohy. Tato sada úloh simuluje práci studentů, která má být odevzdávána do systému GitLab. Slouží proto jako referenční sada, na které se ověřuje funkčnost systému a ladí se vzniklé nedostatky.

4.2.1 Příprava a provedení negativních testů s reprezentativní sadou artefaktů

Při přípravě souborů artefaktů pro negativní testy byla využita zmíněná sada artefaktů pro ověření funkčnosti aplikace a modifikována tak, aby obsahovala artefakty s identifikovanými nedostatky. Pro každou kontrolovanou úlohu vznikly specifické sady artefaktů s nevyhovujícími soubory, které byly následně systémem testované.

Jako chyby se do artefaktů zanášely různé typy nedostatů simulující nedostatečné nebo špatně odevzdávané studentské úlohy. Důležité při tvorbě reprezentativní sady artefaktů byly zkušenosti zadavatele, které byly využity při tvorbě takových nedostatků, které se ve studentských prací v podobných systémech vyskytují nejčastěji. Typové chyby zanášených nedostatků jsou například:

- Chybný název souboru – Požadovaný soubor pro další validaci není nalezený, protože jeho název například obsahuje překlepy, chybný formát nebo je uložen v jiném než požadovaném adresáři.
- Neprávná struktura souboru – Obsah souboru neodpovídá požadované struktuře. To může být v reálném použití způsobené například výraznou úpravou studenta vedoucí k nedodržení datové formátu nebo připravené struktury.

- Nesmyslné a odfláklé práce – Jedná se o nedostatky, kdy student odevzdá úlohu s nesmyslnými popisky nebo kódem, který sice funguje, ale reálně neplní svůj účel např. (test, který nic netestuje).
- Nesplnění minimálních kritérií – Tento popsaný lze považovat za velmi obecný důvod pro nevalidní odevzdání, který je pro každou úlohu specifický. Například v úloze „01_UC“ je kritériem pro splnění počet vytvořených případů užití. V jiných úlohách se jedná například o počet testů, obsáhlost textu a další metriky.

Cílem pozitivního a negativního testování bylo identifikovat chyby, nedostatky a odchylky od specifikace v aplikaci i celého systému a zahrnout je do opravného procesu. Výsledkem tohoto procesu bylo ošetření všech dosud známých problémů, například doplnění a zpřesnění validačních skriptů nebo oprava nesprávného spouštění validací. Během testování byly identifikovány i nedostatky v automatickém výpočtu bodového hodnocení a signalizace nároku na zápočet studenta. Taktéž se objevily nesoulady s požadavky zadavatele, které byly konzultovány a vyžadovalo se jejich upřesnění či změna. Tento proces pomohl zajistit stabilitu a spolehlivost systému při manipulaci s různorodými vstupními daty a situacemi.

Při testování byl kladen důraz na různorodé sady odevzdávaných artefaktů pro negativní testování. Soubor artefaktů pro pozitivní sadu testů byl dodán zadavatelem úlohy přičemž artefakty pro negativní testy vznikaly zanášením chyb a nedostatků do již existujících artefaktů.

Implementace 5

Po návrhu aplikace je na řadě fáze samotné implementace. Kapitola přináší detailní pohled na implementační detaily vyvinuté aplikace a její funkcionality.

5.1 Architektura implementace

Pro implementaci navržené aplikace bylo použito PHP Frameworku Nette, který je jedním z předních aktuálně používaných frameworků. Byly dodržovány navržené návrhové styly a vrstvená struktura projektu.

5.1.1 Implementační framework Nette

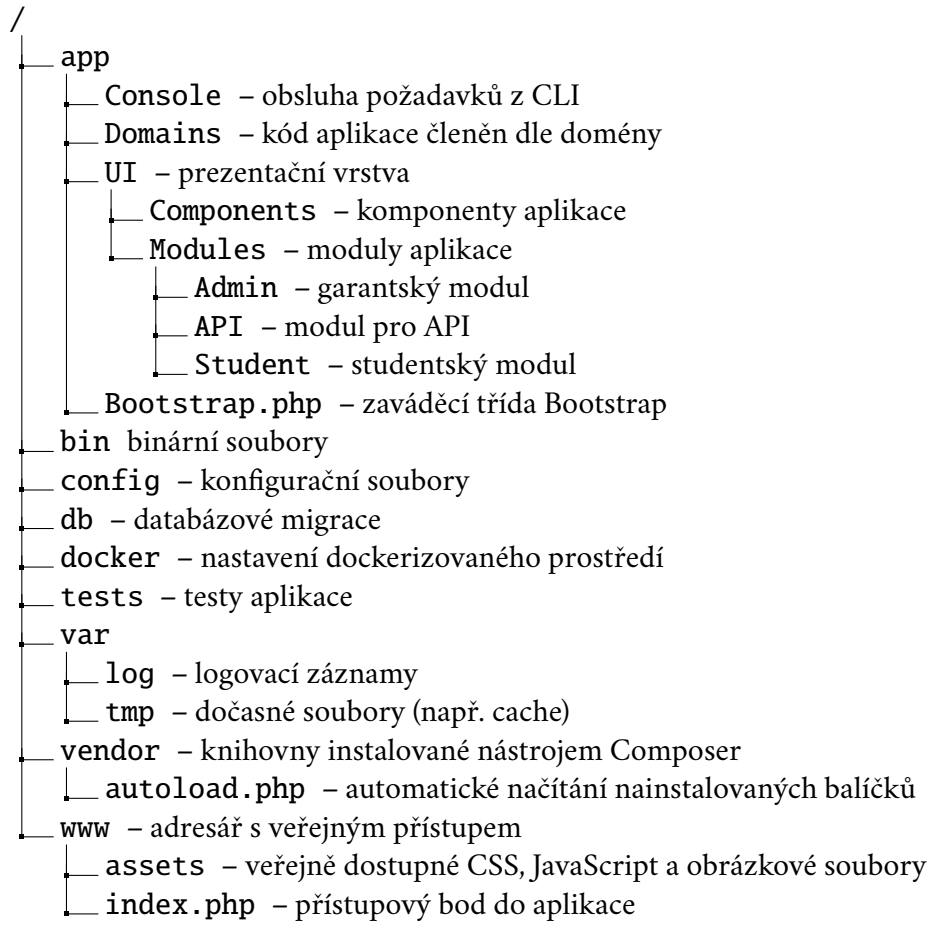
Nette je oblíbený framework pro tvorbu webových aplikací v PHP. Jedná se o open-source projekt zaměřující se na efektivní vývoj s důrazem na bezpečný, čistý a udržitelný kód [Gru23]. Součástí Nette je i subsystém Tracy poskytující značné ladící možnosti a šablonovací engine Latte. Díky balíčkům, rozsáhlé komunitě a aktivnímu fóru se jedná, zejména v českých končinách, oblíbený PHP Framework. Volba Nette pro implementaci navržené aplikace byla učiněna pro výhody ve své jednoduchosti, důrazu na bezpečnost a již získané zkušenosti ve vývoji v tomto frameworku.

5.1.2 Adresářová struktura

Adresářová struktura (viz obr. 5.1) vychází z výchozí struktury Nette Framework a MVP architektury. Zároveň je při organizaci kódu brán ohled na „best practices“ a navržené SOLID principy.

5.1.3 Vrstvená struktura

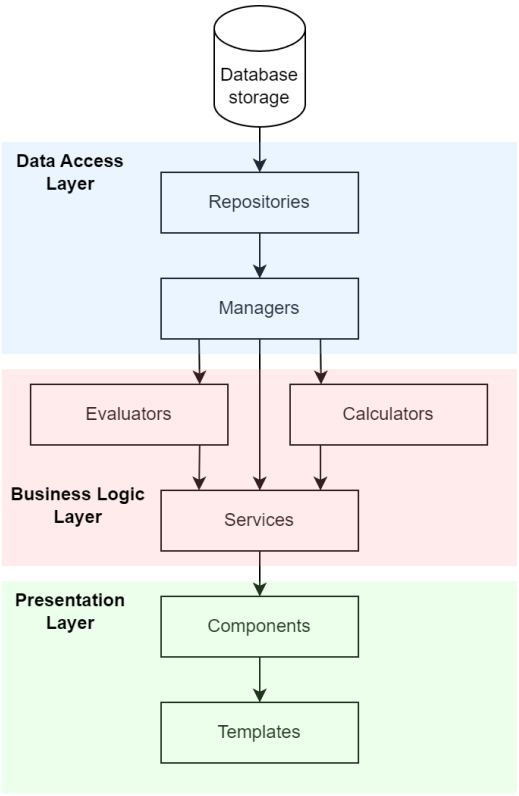
Využití frameworku Nette částečně samo o sobě vede k rozdělení aplikace do tří pomyslných vrstev MVP (Mode-View-Presenter). Avšak z hlediska testovatelnosti a SOLID principů je potřeba aplikaci vrstvit i v jiném směru a to pro oddělení vrstev



Obrázek 5.1: Základní adresářová struktura projektu (zdroj: vlastní)

dle jejich odpovědnosti. Aplikace se v tomto ohledu dělí na tři vrsty ve kterých následně probíhá další struktura organizace tříd (viz schéma na obr. 5.2):

- **Data Access Layer** – Třídy starající se o získání dat. Skládají se z repozitářů (Repositories), které jsou nutné při použití Doctrine ORM. Avšak nedovolují dostatečně rozšiřovat aplikace, proto je zde další přídavná vrstva manažerů (Managers), která získává data z repozitářů a rozšiřuje jejich funkčnost.
- **Business Logic Layer** – Vrstva obsahující logiku aplikace. Zde se jedná o servisní třídy (Services) využívající data získané z vrstvy datového přístupu a používající tříd různých kalkulátorů (Calculators) nebo zpracovatelů (Evaluators), které mají jasně danou zodpovědnost a mohou být snadněji a samostatně testovány.
- **Presentation Layer** – Vrstva pro třídy prezentérů a dalších komponent volající poskytované funkce různých služeb a předávající data k vykreslení do šablon nebo jiného cílového zobrazení.



Obrázek 5.2: Diagram systému s návaznostmi (zdroj: vlastní)

5.1.4 Dependency injection

Vkládání závislostí je programovací technika, která směřuje k vytváření čistého a snadno čitelného kódu. Tato metoda spočívá v delegování zdrojů (služeb), které daná třída nebo funkce potřebuje ke splnění svých úkolů. Cílem je zabránit přímému získávání zdrojů uvnitř objektu, což by mohlo vést k nejastnostem ohledně toho, s jakými zdroji systém pracuje a jak jsou získávány. Tato situace vytváří skryté vazby v aplikaci a vede k neudržitelnému a těžko čitelnému kódu [GF23].

Pro implementaci vkládání závislostí se často využívá nástroj nazývaný Dependency Container. Tento nástroj slouží k uchovávání instancí jednotlivých služeb a jejich vytváření v případě potřeby a pokud jsou závislosti aktuálně vyžadovány nějakou částí aplikace.

Nette pro implementaci Dependency Containeru používá svoji vlastní knihovnu `Nette DI Container` obstarávající automatické generování a aktualizace tříd kontejneru na základě konfiguračního souboru, kde jsou zdroje zaregistrovány. Díky této technologii programátor nemusí vytvářet vlastní řešení a může se plně soustředit na vývoj logiky aplikace. Doplňkem, který Nette dále poskytuje, je funkce Autowire. Tento nástroj automaticky předává závislosti z dependency kontejneru

do konstruktorů a parametrů dalších funkcí, díky tomu nemusí programátor řešit ani způsob získávání zdrojů z kontejneru [Gru24c].

Při implementaci DI je klíčovou součástí tzv. Dependency Container, který slouží k uchování jednotlivých zaregistrovaných služeb a k jejich vytváření, pokud nejsou k dispozici.

5.1.5 Formuláře

Framework Nette si zakládá na bezpečnosti a to se týká i uživatelských vstupů. Je tedy využito modelového popisu formulářů, kdy se validují do nich zadané hodnoty dle předepsaných pravidel a odstiňují různé typy útoků jako například útoky Cross-Site Request Forgery (CSFR) nebo Cross Site Scripting (XSS) [Gru24b].

Tvorba Nette formuláře probíhá stejně jako použití běžné komponenty. Tovární metody vrací instanci `Nette\Application\UI\Form` představující formulářový objekt s definovanými formulářovými prvky. Ty, jak bylo řečeno, mohou mít různá validační kritéria, o které se framework stará a při odeslání formuláře jsou zkontrolovány. Po úspěšné kontrole je volána obslužná metoda, která je odpovědná za vykonání akce úspěšně odeslaného formuláře (např. uložení dat) a je jí do parametrů předána instance objektu formuláře a struktura s validními daty (`ArrayHash`).

Nejjednodušší možnost, jak vytvořit formulář, je přímo v prezentéru (Presenter), kde se vytvoří tovární metoda (viz ukázka kódu 5.1), která vytvoří formulářový objekt, definuje formulářové prvky a přidá callback na obslužnou metodu (v ukázce 5.1 se zavolá metoda `formSucceeded()`). Takto vytvořený formulář je tovární metodou vrácen a může být prezentérem dále použit. Následně lze takový formulář vykreslit v šabloně pomocí značky `control` a názvem komponenty.

Zdrojový kód 5.1: Ukázka definice formulářového objektu (zdroj: vlastní)

```
    {
      public function createComponentRegisterForm(): Form
      {
        $form = new Form();
        $form->addText('login', 'Uživatelské jméno');
        $form->addEmail('email', 'E-Mail');
        $form->addText('password', 'Heslo');

        $form->addSubmit('send', 'Registrovat');

        $form->onSuccess[] = [$this, 'formSucceeded'];
        return $form;
      }
    }

```

V implementované aplikaci je vytvořena třída `App\UI\Form\BaseForm`, která je potomkem zmíněné Nette formulářové třídy. Vlastní implementace potomka je využita pro doplnění vlastních formulářových prvků. V celé aplikaci je využívána právě tato třída `BaseForm`.

5.1.6 Tabulkové gridy

Vyvinutá aplikace se z velké části skládá ze zobrazování informací v tabulkovém formátu. Proto byl využit balíček `ublaboo\datagrid`, který poskytuje dynamické zobrazení tabulek. Umožňuje například překreslování pomocí ajaxových požadavků, stránkování, řazení a filtrování. Taktéž, díky upravené šabloně, dodržuje stejný vzhled tabulek v rámci celé aplikace.

DataGrid tabulky získávají zdroje standardně z dynamického zdroje obsahu (např. Nette Selection nebo Doctrine QueryBuilder), které umožňují úpravu dotazování na určitá data a tím si tabulka vyžádá potřebná data. Například při stránkování instance tabulky upravením zdroje zajistí potřebnou limitaci a offset záznamů. Definice těchto tabulek se kromě předání zdroje dat skládá ze specifikace zobrazovaných sloupců a dalších vlastností DataGrid instance [24b].

Zdrojový kód 5.2: Ukázka definice formulářového objektu (zdroj: vlastní)

```
1    {
2      public function createComponentGrid(): BaseGrid
3      {
4        $grid = new BaseGrid();
5        $grid->setTranslator($this->translator);
6
7        $grid->setDataSource($this->taskService->
   getAllDataSource());
8
9        $grid->setSortable();
10       $grid->setSortableHandler('taskListGrid:sort!');
11       $grid->setPagination(false);
12
13       $grid->addColumnText('shortName', "Zkrácený␣název"
   )
14       ->setFitContent()
15       ;
16
17       $grid->addColumnText('name', 'Název');
18
19       $grid->addColumnNumber('maxPoints', 'Body␣(DOP)')
20       ->setFitContent()
21       ;
22
23       $grid->addColumnNumber('minPoints', 'Body␣(PMIN)')
```

```
24          ->setFitContent()
25          ;
26
27          $grid->addAction('edit', 'Upravit', 'Task:edit')
28          ->setClass('ajax btn btn-sm bg-gradient-secondary mb-0')
29          ->addParameters(['isModal' => true])
30          ;
31
32          return $grid;
33      }
34  }
35
```

Na ukázce kódu 5.2 můžeme vidět konkrétní implementaci tovární metody pro vytvoření tabulky zobrazující seznam úloh vyučujícímu. Kód začíná vytvořením instance třídy `BaseGrid` a předání překladátoru. Na řádce 7 je předán datový zdroj (konkrétně zde se jedná o QueryBuilder z Doctrine ORM), který při vykreslení poskytuje entity `Task`. Řádky 13–30 pak obsahují definici sloupců s jejich názvy a případně upravenými hodnotami pro vykreslení. Nakonec je tabulka tovární metodou vrácena nadřízenému kontrolního prvku.

5.2 Implementace vrstvy datového přístupu

Nedílnou součástí frameworku Nette je i jeho přístup ke komunikaci s databázovým úložištěm pomocí Nette Database Explorer, který využívá vlastnosti abstrakce a odstínění od psaní SQL dotazů přímo v kódu aplikace. Nette Explorer tímto způsobem provádí i různé optimalizace při skládání dotazů.

Avšak pro implementaci navržené aplikace bylo využito technicky jiného řešení pro vytvoření vrstvy datového přístupu. Bylo zvoleno objektově relační mapování (ORM) konkrétně za využití knihovny Doctrine ORM, resp. její úpravy pro použití v Nette frameworku Nettrine ORM [23a].

5.2.1 ORM a Doctrine ORM

Technika ORM umožňuje propojit objektově orientované programování (OOP) s relační databází a umožňuje pracovat s daty jako s objekty, aniž by bylo potřeba psát SQL dotazy. Pro toto využití ORM zajišťuje mapování mezi objekty a záznamy v tabulkách včetně odpovídajících datových typů a různými vztahy mezi tabulkami. Výhodou ORM bývá zvýšená produktivita, zjednodušení údržby, zlepšení čitelnosti kódu a lepší testovatelnost. Naopak nevýhodou je jistá složitost definování entit a vztahů mezi nimi, snížení výkonu z důvodu režie zajištění konverze mezi objekty

a daty. Využití ORM přístupu je také potřeba dobře navrhnout, protože při špatném návrhu se snižuje výkonnost a flexibilita práce s daty.

Doctrine ORM je populární open-source knihovna přinášející ORM techniky do PHP, umožnující kromě dříve zmíněných funkcí také automatické generování SQL dotazů, validaci dat a správu transakcí. Knihovna Doctrine je rozdělena na dvě části Doctrine DBAL a Doctrine ORM. Část DBAL je abstraktní vrstva databázového přístupu umožňující pracovat s různými typy databází za použití stejného rozhraní. Díky tomu má aplikace podporu pro komunikaci s vícemi databázovými systémy. Druhá ORM část se zabývá vlastní implementací ORM nad abstrakcí DBAL a poskytuje funkce pro mapování objektů na tabulky a práci s daty [24c].

Ve vyvíjené aplikaci má každá databázová tabulka svého protějška ve formě aplikační entity držící informace odpovídající datovým záznamům. Získávání těchto entit z databázových dat je zajištěno pomocí dvou způsobů:

1. Repozitáře – Repozitář je nejnižší člen vrstvy datového přístupu aplikace a stará o získávání konkrétních dat. Tato služba poskytuje řadu metod pro vybírání, filtrování, řazení apod., které doplňuje samotná knihovna Doctrine ORM. Další metody lze vytvářet skládáním knihovnou poskytovanými metodami. Z třídy repozitáře standarně získáváme přímo entity dle její příslušnosti.

2. QueryBuilder – Oproti repozitáři, který získává entity z databáze při volání metody, je přístup QueryBuilder odlišný a více nízkoúrovňový. Svoji funkcionalitou připomíná Nette Database Explorer. Za pomoci QueryBuilder objektu formujeme dotaz voláním parciálních metod a tím definujeme samotný dotaz v jazyce DQL (Doctrine Query Language). Jedná se o abstraktní dotazovací jazyk podobný jazyku SQL, ale využívající názvy entit a jejich atributů. Díky tomu je DQL odstíněn od konkrétně zvoleného databázového systému a ve fázi zpracování dotazu je DQL přeložen na SQL používaného typu databáze. QueryBuilder je v aplikaci využíván jako datový zdroj do dynamických tabulek DataGrid. Ty využívají tuto techniku například při stránkování nebo filtrování, kdy upravují podobu DQL dotazu. Díky tomu si tabulky získávají jenom ty záznamy, které právě zobrazují a zvyšují tím efektivitu. Pokud je QueryBuilder definován nad nějakou entitou a nezískává informace, které daná entita nepojme, vrací jako získaná data již namapovanou entitu [24d].

5.3 Poskytované API

Aplikace pro komunikaci s dalšími prvky celého systému poskytuje komunikační rozhraní REST API na bázi HTTP. Toto API umožňuje komunikaci aplikace se

dvěma různými systémy. Výchozí adresa pro přístup k API modulu aplikace je dle bázové adresy následující:

```
https://<base_address>/api/v1
```

5.3.1 Studentský dashboard

Prvním systémem, se kterým aplikace navazuje spojení, je studentský dashboard provozovaný na produkčním prostředí pro studentské artefakty. Tento dashboard slouží jako rozcestník pro výstupy jednotlivých úloh a zobrazuje jejich stav. Informace o postupu odevzdání a dalších stavů jsou získávány z vyvinuté aplikace pomocí poskytovaného API přes následující přístupový bod:

```
GET /evaluations/<orion_login>
```

Odpovědí na tento požadavek je seznam úloh s jejich stavy a dalšími informacemi ve formátu JSON (viz ukázka 5.3). Z podstaty získávání dat ze strany dashboardu pomocí jednoduchého JavaScript volání vykonávaného prohlížečem studenta je přístupový bod veřejný a není potřeba žádného ověření identity.

Zdrojový kód 5.3: Ukázka odpovědi API na postup plnění úloh (zdroj: vlastní)

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

```

5.3.2 GitLab Webhooks

Druhým systémem, pro který vyvíjená aplikace poskytuje API, jsou GitLab Webhooks notifikace. Díky požadavkům, které přijímá tento přístupový bod, je umožněno dynamicky reagovat na události a na jednotlivé validace proběhnuté v pipelines v době, kdy doběhly. To je zajištěno zasíláním POST požadavků na následující přístupový bod:

```
1    POST /webhooks
```

Příchozí požadavky jsou kontrolovány na jejich pravost pomocí tajného tokenu uloženého v hlavičce `X-Gitlab-Token`. Přístup k tomuto přístupovému bodu je tedy omezen na konfigurovanou instanci platformy GitLab. Struktura obsahu požadavků odpovídá Webhook struktuře (např. ukázka 2.1) pro systém GitLab.

5.4 Uživatelské rozhraní

Implementace uživatelského rozhraní zahrnuje využití frontendového frameworku Soft UI Dashboard, který vychází z populárních stylů Bootstrap. Tato volba poskytuje uživatelům systému esteticky přitažlivé prostředí, responzibilitu a vyšší použitelnost. Díky licenci MIT je tento framework volně dostupný ve své verzi zdarma a nabízí širokou škálu možností pro tvorbu moderního uživatelského rozhraní [18].

Průběh implementace uživatelského rozhraní byl koordinován dle požadavků a potřeb zadavatele, kdy návrhy jednotlivých pohledů byly podrobeny diskuzi v rámci pravidelných konzultací, což umožňovalo dynamicky reagovat na vzniklé nejasnosti. Po integrování veškerých požadavků bylo uživatelské rozhraní implementováno do své finální podoby.

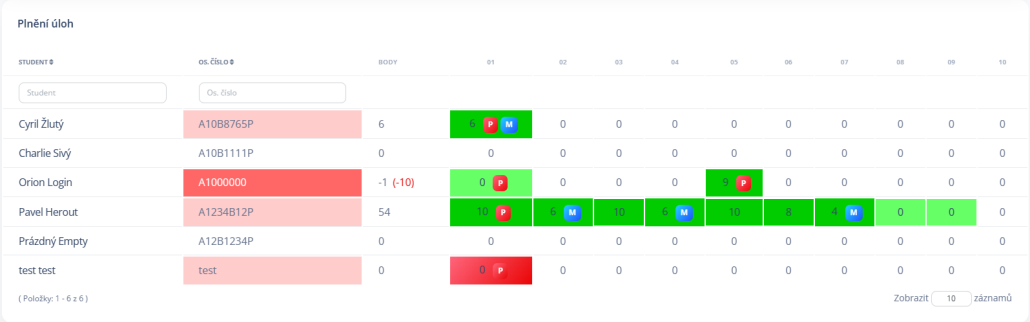
Kromě estetického provedení je uživatelské rozhraní navrženo tak, aby maximalizovalo uživatelskou přívětivost, intuitivnost a zážitek z používání (UX), což přispívá k snadnému a pohodlnému používání aplikace. Uživatelské rozhraní proto obsahuje například dynamické tabulky, modálová okna nebo hlášení o proběhlých akcích.

5.4.1 Ukázky uživatelského rozhraní

V této sekci jsou prezentovány reálné snímky obrazovky z vyvinuté aplikace, které ilustrují její vzhled uživatelského prostředí a funcionalitu. Jsou vybrány ukázky z již prezentovaných návrhů drátových modelů, které byly popsány v rámci této práce. Proto budou obrázky doprovázeny pouze krátkým popisem s důrazem na odlišnosti oproti návrhu.

5.4.1.1 Přehled vyučujícího na plnění úloh studentů

Nejdůležitější vizuální bod pro vyučujícího, kde vidí postup jednotlivých studentů v jejich plnění úloh tabulkovou formou (viz obr. 5.3). Pomocí buněk s odevzdanými úlohami se zobrazí modálové okno s podrobnějšími informacemi a hodnocením. Sloupec s osobním číslem indikuje stav zápočtu kombinovaně s počtem indikovaných nedostatků. Je zde využita dynamická tabulka Datagrid umožňující filtrování pomocí osobního čísla a jména studenta.



Obrázek 5.3: Tabulka s přehledem odevzdáváním (zdroj: vlastní)

5.4.1.2 Formulář hodnocení studentské úlohy

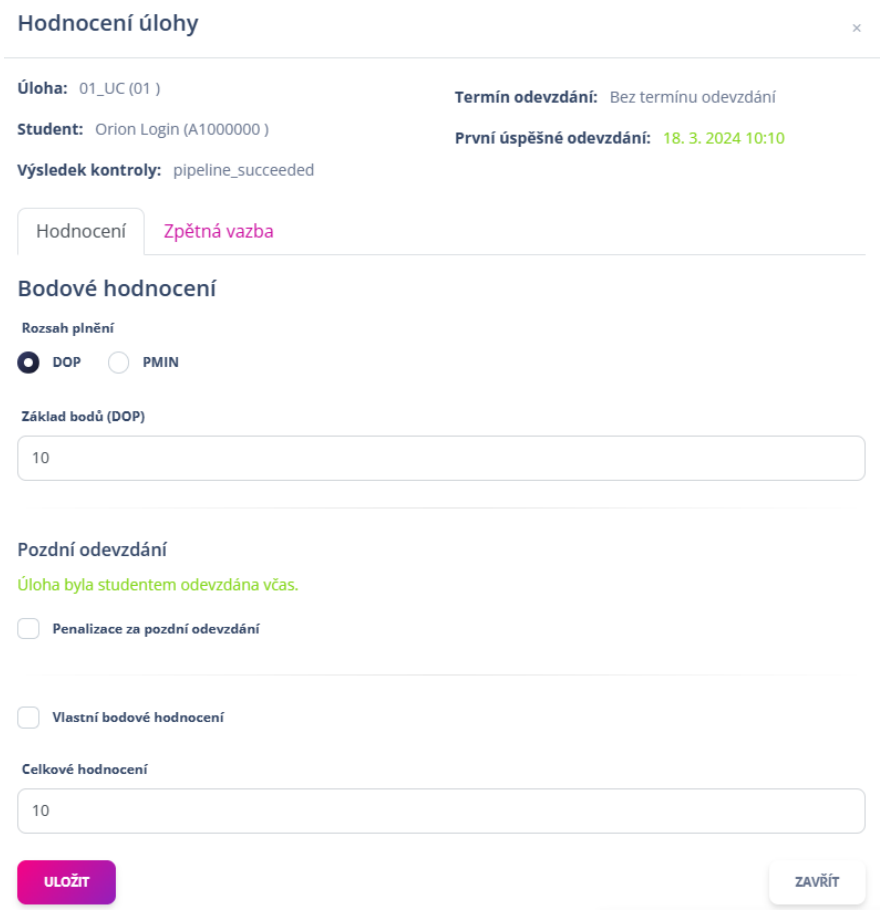
Rozkliknutím buňky odevzdané úlohy v tabulce přehledu odevzdávání se otevře modálové okno s hodnocením studenta a podrobnými informacemi o odevzdání (viz obr. 5.4), zde může vyučující zadat bodové hodnocení nebo v kartě „Zpětná vazba“ zadat slovní komentář nebo vytknout nedostatky.

5.4.1.3 Formulář definování úlohy

Pro ilustraci spojení pipeline procesů v GitLab a úloh v garantské aplikaci je zde ukázka formuláře definující úlohu (viz obr. 5.5). Spojení je realizováno zapomocí identifikujících názvů pipeline procesů (jobs) a úrovně (stage). Další propojení je systémem dashboardu provozovaného na studentském produkčním prostředí přistupující k datům pomocí API poskytované garantskou aplikací. Zde je propojení provedeno identifikátorem HTML elementu pro zobrazení výsledků a relativní cesty k artefaktu pro každou úlohu.

5.5 Konfigurace

Aplikace je navržena a implementována způsobem, který umožňuje přepínání hodnot z konfiguračních souborů. Ty jsou uloženy v adresáři `/config`, ve kterém se vy-



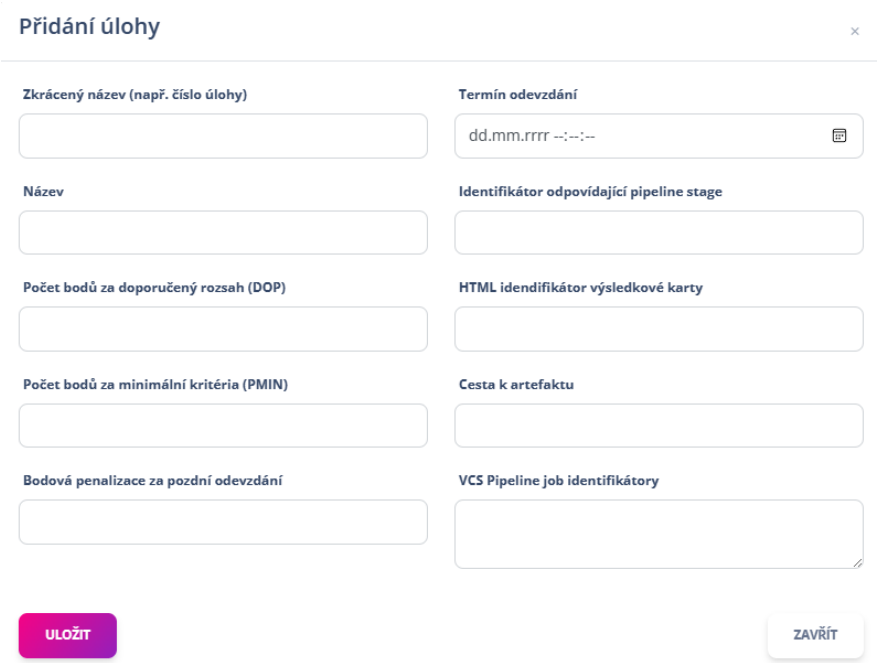
Obrázek 5.4: Formulář pro udělení hodnocení (zdroj: vlastní)

skytuje i klíčový konfigurační soubor `local.neon` obsahující konfiguraci připojení k databázovému serveru, přístupové údaje k GitLab API a OIDC a další parametry. Tento soubor poskytuje základní nastavení pro chod aplikace. Konfigurace v souborech je psána ve formátu NEON, což je vlastní formát pro konfiguraci frameworku Nette podobný formátu YAML.

Druhá část konfigurace je možná přímo z prostředí aplikace a je dostupná pro úpravy od role Garant. Ten může upravovat konfigurační hodnoty, které po uložení aplikace okamžitě využívá. Jedná se zejména o nastavení, která se často mění a je potřeba k nim mít přístup bez přístupu ke konfiguračním souborům.

Ukázka 5.4 konfigurace spočívá v nastavení kontextu aplikace pro zajištění její ho chodu, proto se soubor nazývá `local.neon`. Před instalací je namísto něj k dispozici ukázkový soubor `local.neon.example`. Konfigurace z ukázky 5.4 se dělí na několik částí:

- **Kontextu databáze (řádky 3–7)** – Zabývá se údaji na připojení k databázi, skládá se z adresy databázového serveru, názvu databáze, uživatele a hesla.



Obrázek 5.5: Formulář pro vytvoření úlohy (zdroj: vlastní)

- **Identifikace úložného prostoru (řádek 9)** – Jednoznačný identifikátor aplikace, který rozděluje namespace pro userStorage, aby nevznikl konflikt se session s jinou aplikací běžící na stejném serveru.
- **Testovací režim (řádky 11–13)** – Možnost nastavení testovacího (ladícího) režimu aplikace. Aplikace například v tomto módu nezasílá emaily do schránek opravdových studentů. Email je však odchycen dalšími ladícími mechanismy (např. `FakeLogMailer` z 22. konfigurační řádky).
- **Spojení s GitLab (řádky 15–19)** – Nastavení spojení s instancí GitLab. Je možné nastavit libovolnou adresu instance GitLab, přístupový token uživatele s dostatečnými oprávněními a nastavení `k8sNamespaceCreatorProjectId`, to odpovídá identifikátoru projektu určeného pro inicializaci proměnných a nasazení na Kubernetes Cluster, kde studentské práce mají své produkční prostředí. Hodnota token `webHookSecretToken` slouží k zabezpečení a ověření identity přijímaných GitLab Webhook požadavků.
- **Nastavení autentizace (řádky 23–26)** – Aplikace umožňuje přihlášení pomocí univerzitního SSO na bázi OIDC (nadstavba OAuth 2.0), kdy poskytovatel vystaví pro produkční prostředí klientský secret. Tento secret a klientská identifikace se zde nahrazuje namísto `client-secret` a `client-id`.

Zdrojový kód 5.4: Ukázka nastavení aplikace (zdroj: vlastní)

```
{
  parameters:
  database:
  host: 'DB_HOST'
  dbname: 'DB_NAME'
  user: 'DB_USER'
  password: 'DB_PASSWORD'

  appspace: 'oks-web-garant-app'

  config:
  testing:
  enabled: true

  gitlab:
  url: 'https://gitlab-vyuka.kiv.zcu.cz/'
  token: 'glpat-token-here'
  k8sNamespaceCreatorProjectId: 8
  webHookSecretToken: 'secret-token'

  services:
  - App\Model\Mail\FakeLogMailer
  - App\Model\Security\Authenticator\
ZcuOidcAuthenticator\ZcuOidcProvider([
  clientId: 'client-id'
  clientSecret: 'client-secret'
  ])
  tracy:
  showBar: true
}

```

5.6 Dockerizované prostředí

Pro vývoj byla využita technologie kontejnerizace Docker, přinášející značné výhody pro stabilitu, spolehlivost a konzistenci nasazení do produkčního prostředí. Tento přístup umožňuje simulovat produkční infrastrukturu v lokálním prostředí vývojáře, což zajišťuje do jisté míry kompatibilitu s cílovým prostředím a minimalizuje nekonzistenci a výskyt chyb při procesu nasazení. Přestože vyvíjená aplikace na produkci nebude spuštěna v kontejneru z důvodu specifických požadavků a limitů serverové infrastruktury a její politiky, přínosy zavedení dockerizovaného prostředí přetrvají zejména pro proces vývoje.

5.7 Autentizace a bezpečnost

Autentizace uživatelů je v aplikaci řešena pomocí dvou přístupů. Prvním z nich je standardní přístup pomocí třetí strany univerzitního SSO, které je blíže popsáno dále. Druhou možností je přihlášení přes uživatelské jméno a heslo, které je spíše určeno pro nutné zásahy do aplikace uživatelem bez Orion přístupu, pro účely instalace a počáteční incializace, pro autentizaci uživatelů nechtějící využívat univerzitní SSO. Při přihlášení pomocí jména a hesla jsou údaje (resp. hesla jsou) ukládána v databázovém systému ve formě jejich otisku pomocí hashovacího algoritmu BCRYPT využívaný nativně frameworkem Nette pomocí třídy `Nette\Security\Passwords`.

Další formou zabezpečení produkčního prostředí je zajištění komunikace přes zabezpečenené spojení HTTPS zařízené na straně produkčního serveru.

5.7.1 Přihlášení přes OIDC

Aplikace umožňuje přihlášení přes službu třetí strany (univerzitní SSO) pro autentizaci studentů a vyučujících. Díky tomu se omezuje nutnost spravovat hesla všech uživatelů přistupující do aplikace a je vyřešen problém se zobrazením dat pouze studentům, kterým patří. Aplikace funguje jako OIDC klient, který je zaregistrován organizací CIV, a pomocí přístupových tokenů komunikuje s autentikačním přístupovým bodem.

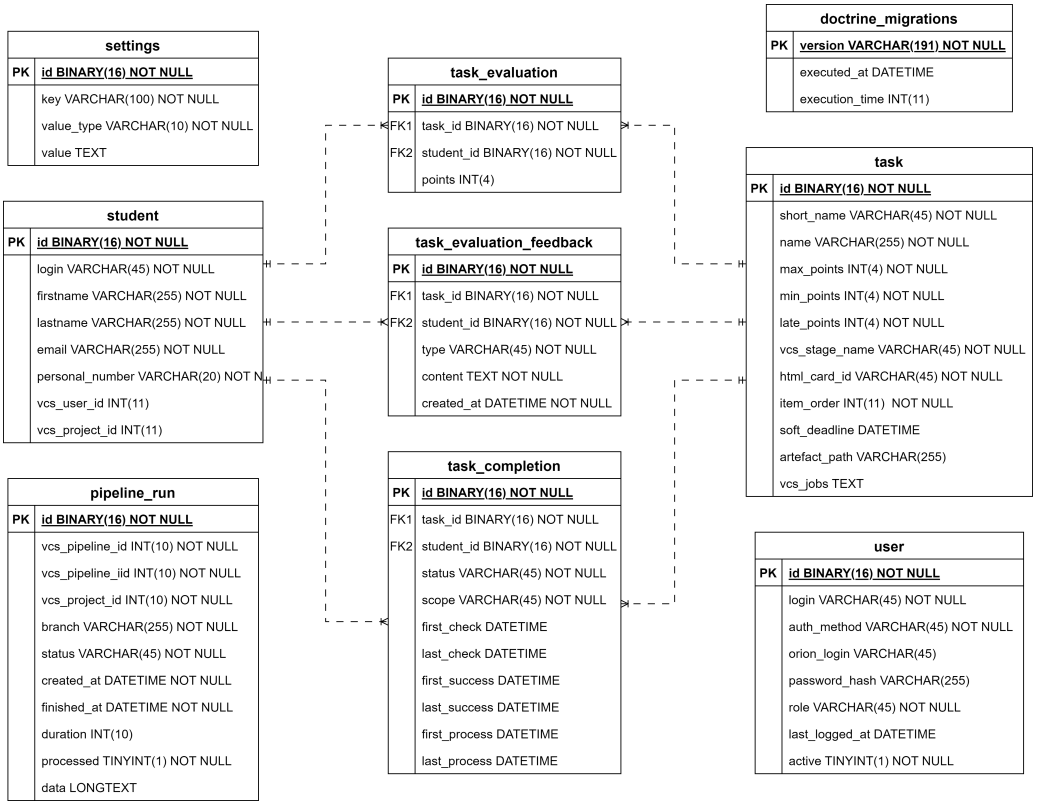
Protokol OIDC je nadstavbou nad OAuth 2.0 vytvářející jednoduchou autentizační vrstvu pomocí JSON Web tokenů. Umožňuje aplikacím jednoduché napojení s různými poskytovateli identit [23b], jako jsou například zde využívané univerzitní identity zaměstnanců a studentů [Sak+14].

Pro využití OIDC byl v implementaci využit balíček `contributte/oauth2-client`, který poskytuje abstrakci nad OAuth protokolem a zjednodušuje integraci s různými poskytovateli identit. Aplikace rozeznává jednotlivé uživatele s univerzitními identitami pomocí Orion přihlašovacího jména.

5.8 Databázové úložiště

Pro implementaci byl zvolen databázový systém MariaDB díky své otevřenosti dlouhodobé udržovovanosti nezávislou skupinou. Systém MariaDB, stejně jako MySQL je velmi oblíbenou SQL databáze pro vývoj webových aplikací i díky možnosti jednoduché instalace, správy i rozsáhlé kompatibilitě. Ostatně byl zvolen i z důvodu již existující běžící instance na produkčním prostředí a proto se nabízí její využití při provozu aplikace. Ve vyvíjeném systému je taktéž navrženo ukládání příchozích požadavků do fronty. Ta, z důvodu omezení produkčního prostředí, byla implementována formou běžné tabulky v databázovém systému, kde záznamy mají kromě

svých sloupců i stavy a indikaci zpracování. Stejné zabezpečení je taktéž využíváno při komunikaci se systémem GitLab, OIDC přístupovým bodem u poskytovaného API.



Obrázek 5.6: Diagram schématu databáze (zdroj: vlastní)

5.8.1 Databázová struktura

Z návrhového vztahu jednotlivých entit (viz obr. 3.2) byla vytvořena podoba databázové struktury, jež je vidět na schématu 5.6 a skládá se dohromady z 9 tabulek (včetně tabulky nutné pro databázové migrace). Základem implementované struktury je spojení mezi tabulkami `student` (obsahující informace o každém studentovi a jejich propojení s platformou GitLab) a `task` (definice odevzdávaných úloh s propojením se systémem GitLab) M:N relací rozložená celkově do tří tabulek:

- `task_completion` – uchovávající stav jednotlivých úloh po zpracování, ukládá také časy o zpracování, prvním úspěšném odevzdání, posledním podevzdání apod.,
- `task_evaluation` – uchovává bodové hodnocení úlohy,

- `task_evaluation_feedback` – reprezentuje zpětnou vazbu, jakožto komentáře a nedostatky odevzdané práce, udělenou vyučujícím.

Další tabulky nejsou napojené na tuto základní strukturu, jsou potřebné pro chod aplikace, jejích operací nebo jako podpůrný mechanismus:

- `user` – tabulka uživatelů s přístupem do aplikace, zde se nachází právě záznamy vyučujících, kteří mají do systému přístup a mohou například udělovat hodnocení,
- `settings` – uchovávající nastavení aplikace, které lze měnit prostřednictvím UI, kde každá nastavitelná položka má svůj záznam v této tabulce s jedinečným identifikátorem a popisem datového typu uložené hodnoty,
- `pipelin_run` – seznam přijatých notifikací čekajících na své zpracování,
- `doctrine_migrations` – tabulka nástroje pro řízení databázových migrací knihovny Doctrine ORM.

5.8.2 Databázové migrace

Migrace databáze jsou nástrojem pro verzování databázové struktury a operací prováděných v ní. Umožňuje tedy, aby struktura databáze odpovídala verzi vyvíjeného kódu. To je v implementaci zajištěno využitím funkcí Doctrine ORM, které poskytují nástroj pro databázové migrace. Ve zjednodušení proces funguje vytvořením migrace, kde se definuje název, popis a změnu databáze SQL dotazem. Tím je vytvořen soubor (třída) reprezentující jednu migraci (verzi databáze). Příkazem pro spuštění migrace se následně spustí dosud neproběhlé migrační verze, aby po dokončení byla databáze ve stavu odpovídající verzi kódu. Pro zajištění neduplicitního spouštění jednotlivých verzí. Do databáze, se kterou je pracováno, je uložena speciální tabulka se seznamem provedených verzí. Tím se zajistí, že každá verze proběhne právě jednou.

5.9 Plánování a průběh projektu

Vývoj aplikace byl realizován inkrementálně s důrazem na průběžnou iterativní dodávku předem dohodnutých funkcí. Ty byly konzultovány na pravidelných schůzkách se zadavatelem a s autorem další práce zabývající se jinou částí celkového vyvíjeného systému. Pravidelné schůzky byly konány jednou týdně, to umožnilo dynamickou reakci na zpětnou vazbu a průběžné upřesňování požadavků.

Během procesu vývoje bylo potřeba se potýkat s různými problémy. Například bylo náročné synchronizovat se s paralelním vývojem prací na technologickém

stacku pro validaci (součástí jiné práce). Taktéž bylo nutné se potýkat s problémy ohledně nestability univezitní instance systému GitLab a pružné povaze požadavků, které v průběhu projektu byly nejen zpřesňovány, ale také měněny. Díky pravidelným schůzkám se zadavatelem a dalšími zainteresovanými osobami byla ale možná rychlá a případná náprava.

Celkově lze říct, že projekt byl řízen agilně s náležitou pružností a schopností reagovat na změny a problémy během procesu vývoje.

5.10 Vyvinutá aplikace

Výsledná aplikace byla vyvinuta do verze 1.0 obsahující veškeré požadované funkce v žádaném rozsahu. Je proto připravena pro nasazení na produkční prostředí před začátkem nového semestru, kdy bude využita v rámci výuky předmětů OKS a WEB. Po nasazení budou zadavateli bezpečnou formou předány veškeré údaje nutné pro provoz a další vývoj aplikace. Aplikace stejně jako celý nově vzniklý systém, kterého je součástí, bude rozšiřována a vylepšována po zkušenostech z prvního ročníku aktivního používání. Na tyto úpravy je aplikace připravena.

Testování a validace 6

Testování aplikace bylo po celou dobu návrhu a vývoje zásadním atributem. Jelikož se jedná o aplikaci, která pracuje s hodnocením studentských prací, následně získává a drží počty bodů a signalizuje nárok na získání zápočtu, považují se tyto části za kritické. Proto je důležité zejména tyto části mít pokryté různými druhy testů.

6.1 Testování aplikace

Framework Nette má své nástroje pro testování svých aplikací nazývající se Nette Tester. Jedná se o nástroj, díky kterému je možné psát testy formou PHP skriptu a lze je pouštět samostatně i hromadně [Gru24a].

6.1.1 Jednotkové testy

Jednotkové testy slouží k ověření částí aplikace izolovaně od ostatních. Cílem je zajistit, že každá část kódu pracuje podle očekávání a splňuje specifikace požadavků. Testy se zaměřují na ověření jednotlivých komponent systému tj. např. funkcí, metod, tříd [Kho20]. Externí závislosti jsou nahrazeny falešnými implementacemi (např. FakeLogger) nebo technikou mockování knihovnou `Mockery`, aby bylo dosaženo izolování testované části [BM24]. Jednotkové testy jsou obvykle rychlé a snadno automatizovatelné, což umožňuje časté spouštění během vývoje. K tomuto účelu byla vytvořena GitLab pipeline, která při přírůstcích aplikace mimo jiné spustí i jednotkové testy.

Zdrojový kód 6.1: Ukázka jednotkového testu (zdroj: vlastní)

```
1    {
2      public function testAuthenticateInvalidPassword():
   void
3      {
4        $login = 'testUser';
5        $testUser = new User($login);
6        $testUser->setId(Uuid::uuid4());
7        $testUser->changePasswordHash('passwordHash');
```

```
8         $testUser->setAuthMethod(User::
    AUTH_METHOD_PASSWORD);
9
10        $mockUserService = \Mockery::mock(UserService::
    class);
11        $mockUserService->allows('getByLogin')->with(
    $login)->andReturn($testUser);
12
13        $mockPasswords = \Mockery::mock(Passwords::class);
14        $mockPasswords->expects('verify')
15        ->andReturns(false);
16
17        $authenticator = new UserAuthenticator(
    $mockUserService, $mockPasswords);
18
19        Assert::exception(function () use ($authenticator)
     {
20          $authenticator->authenticate('testUser', '
    invalidPassword');
21        }, AuthenticationException::class, 'The␣password␣
    is␣incorrect.');
22      }
23    }
24
```

Jednotkovými testy bylo pokryto 71% zdrojového kódu pomocí testováním modelu, logické vrstvy a vrstvy datového přístupu. Struktura organizace testovacích tříd odpovídá adresářové struktuře ve zbytku aplikace.

6.1.2 Integrační testy

Integrační testy pro ověření funkčnosti návaznosti různých systémů byly v rámci této práce provedeny manuálním způsobem. To bylo rozhodnuto z důvodu složité a neefektivní simulace platformy GitLab, kdy aplikace potřebuje pro zpracování dat získávat data pomocí API. Manuální metody pro integrační testy spočívají v simulaci odevzdávání studentských artefaktů do inicializovaného repozitáře. Tím se spouští pipeline, která úlohy zpracuje, zkontroluje a vypublikuje. Výsledek dokončeného hběhu pipeline je odeslán notifikací do aplikace, která ho zpracuje. Vytovření reprezentativní sady artefaktů a ověření funkčnosti aplikace byly již popsány v sekci 4.2.

6.1.3 Databázové testy

Součástí testování jsou i databázové testy, které ověřují výsledky CRUD operací na některých entitách a integračních omezení na testovacích datech. Tyto testy

jsou ověřovány pomocí již zmíněného Nette Testeru využívající Doctrine ORM abstrakce ve vrstvě datového přístupu. Při testování bylo využito scénářových testů, které podle připravéného scénáře ověřují funkce aplikace a datové vrstvy [Her22].

6.2 Logování

Aplikace pro účely trasovatelnosti vykonaných akcí implementuje knihovnu `Monolog` sloužící k vytváření logovacích záznamů aplikace. Díky logovacím záznamům lze snadněji vysledovat příčinu selhání nebo jiné odchylky. Soubory s logy jsou ukládány do adresáře `/var/log` [24h].

Závěr 7

Tato diplomová práce se zabývala implementací webové aplikace pro efektivní správu a řízení systému odevzdávání semestrálních prací studentů. Byla implementována PHP aplikace ve frameworku Nette za použití technologií jako Doctrine ORM a dodržení SOLID principů. Vyvinutá aplikace má za cíl získávat data o odevzdaných pracech ze systému GitLab pomocí jeho API a notifikování Webhooks, zpracovat je a vizualizovat.

Aplikace poskytuje celkem 22 pohledů webových stránek a dvou přístupových bodů ke komunikaci skrze své API. Pro implementaci aplikace bylo vytvořeno celkem 230 tříd a rozhraní a 69 souborů šablony, z toho 43 tříd slouží k definování znovupoužitelných komponent. Všechny vrstvy aplikace byly ověřeny různými typy testů.

V současné době je aplikace nasazena v testovacím režimu pro provedení sady ověření, aby byla aplikace připravena na zahájení dalšího semestru, kdy bude vystaven oponentuře reálného použití při výuce předmětu OKS. Samotné produkční nasazení aplikace proběhne dle stanovení zadavatele v období červen až září 2024.

v rámci této práce byl zpracován taktéž návrh validačních úloh pro ověření studentských prací, který bude využit při tvorbě validačních skriptů automatické validace. Ve spojením s tím byl na základě referenční sady artefaktů odevzdávaných úloh vytvořen soubor sad vypracovaných úloh pro negativní testování. Pomocí těchto sad artefaktů pro pozitivní a negativní testování byla ověřena funkčnost aplikace včetně zpracovávání a hodnocení.

7.1 Další možný vývoj

Aplikace včetně celého systému má mnoho potenciálu jak v univezitním prostředí růst. Byl projeven zájem o prozkoumání možností integrace celého techologického stacku a procesů pro využití ve výuce dalších předmětů. Právě pro tyto případy je aplikace vytvořena dostatečně abstraktně, aby úprava pro využití v jiných předmětech vytvářela minimální požavky na změnu v aplikaci.

Návod k instalaci

Tento návod slouží k provedení procesem instalace webové aplikace OKS-WEB-GARANT-APP vyvinuté pomocí PHP a Nette frameworku. Následující kroky popisují snadné nastavení prostředí pro nasazení aplikace.

A.1 Požadavky na nasazení

Pro nasazení aplikace je vyžadováno běhové prostředí webového serveru s modulem PHP 8.2 a databázovým systémem MySQL nebo MariaDB. Nástroj Composer k dodání závislostí je vyžadován. Pro bezpečnost je doporučeno využít SSL certifikátu pro zabezpečené HTTPS spojení. K zprovoznění OIDC přihlašování je potřeba mít vytvavené přístupové údaje a schválenou návratovou adresu do aplikace.

A.2 Instalační kroky

1. **Stažení zdrojových kódů** – Stažení zdrojových kódů z repzitáře GitLab, kde je umístěna a vyvíjena. Předpokládejme, že zdrojové kódy jsou umístěny na katedrální instalaci GitLab určeného pro výuku.

```
git clone ssh://git@synergia.civ.zcu.cz:31222/oks-web/management-tools/web-oks-garant-app.git
```

2. **Instalace závislostí** – Přejděte do adresáře s kódem (obsahuje `composer.json`) a spusťte Composer pro instalaci všech potřebných závislostí.

```
composer install --ignore-platform-reqs
```

3. **Konfigurace** – v adresáři aplikace se nachází složka `/config` obsahující konfiguraci aplikace. Zkopírujte soubor s ukázkovou konfigurací `/config/local.neon.example` do `/config/local.neon`. Otevřte soubor `/config/local.neon` pro editaci a nastavte v konfiguraci následující položky.

- Nastavte databázové spojení. K tomu potřebujete údaje pro připojení k databázi: adresu serveru, název databáze, jméno uživatele a jeho heslo. Změňte tedy následující konfiguraci:

```
        database:
        host: database
        dbname: pia-sp2
        user: root
        password: root_pristup_123
```

  Na konfiguraci s hodnotami zvolené databáze:

```
        database:
        host: db.example.com
        dbname: oks-web
        user: oksuser
        password: oks_user_example_password
```

- Pokud se na serveru nachází více aplikací využívající session, je doporučeno změnit hodnotu `appspace` na jedinečný identifikátor pro instalovanou aplikaci. Změňte tedy konfiguraci:

```
        appspace: 'WebAppNamespace'
```

  Na změněnou hodnotu:

```
        appspace: 'OKS-WEB-GARANT-APP'
```

- Nastavujeme-li nasazení, které je produkční nebo není testovací, změňte hodnotu pro nastavení testovacího režimu:

```
        config:
        testing:
        enabled: true
```

  Na hodnotu:

```
        config:
        testing:
        enabled: false
```

- Nyní nastavte spojení se systémem GitLab. Pro toto nastavení potřebujete znát bázovou adresu serveru GitLab, osobní přístupový token (návod na získání v sekci A.3), číselný identifikátor projektu s inicializační pipeline (např. 9) a vygenerovaný náhodný řetězec jako token pro Webhook komunikaci (ten si uschovejte pro nastavení v sekci A.4. Konfiguraci tedy změňte z:

```
        gitlab:
        url: 'https://gitlab.kiv.zcu.cz'
        token: 'TOKEN\_HERE'
```

```
        k8sNamespaceCreatorProjectId: 8
        webHookSecretToken: 'secret-token'
```

Na konfiguraci:

```
        gitlab:
        url: 'https://gitlab-vyuka.kiv.zcu.cz'
        token: 'glpat-example-token'
        k8sNamespaceCreatorProjectId: 9
        webHookSecretToken: 'abc-example-string-
    xyz'
```

- Pro nastavení OIDC přihlašování potřebujete znát klientský identifikátor (`clientId`) a tajemství klienta (`clientSecret`), které nám sdělí poskytovatel OIDC identity (v tomhle případě ZČU). Návratová adresa pro aplikaci je `<base_address>/auth/zcu-oidc-callback`. Získaný identifikátor a token vložte do konfigurace:

```
        services:
        - App\Model\Mail\FakeLogMailer
        - App\Model\Security\Authenticator\
    ZcuOidcAuthenticator\ZcuOidcProvider([
        clientId: 'CLIENT_ID'
        clientSecret: 'CLIENT_SECRET'
        ])
```

Výsledkem čehož bude přibližně následující konfigurace:

```
        services:
        - App\Model\Mail\FakeLogMailer
        - App\Model\Security\Authenticator\
    ZcuOidcAuthenticator\ZcuOidcProvider([
        clientId: '6sdf4s5df5s4f45sdfsdf54'
        clientSecret: 's5d4f54sd-4sdf54s45f-4
    sdf45sdf-445fgfgd'
        ])
```

4. **Vytvoření databázové struktury** – Provedením databázových migrací vytvořte strukturu aplikace a naplňte je incializačními daty. Migraci databáze spusťte následujícím příkazem z kořenové ho adresáře aplikace:

```
    ./bin/console migrations:migrate --no-
  interaction
```

5. **Spuštění webového serveru** – Využitím PHP vývojového serveru lze spustit vývojový server příkazem:

```
    php -S localhost:80 -t www
```

Aplikace bude spuštěna na zadané adrese. Pokud však nasazujete aplikaci na produkční prostředí, bude standardně potřeba aplikaci přesunout do určeného adresáře dle používaného webového serveru. V tom případě zajistěte, aby byl veřejně přístupný adresář `/www` a aby aplikace měla právo zápisu do adresářů `/var/tmp` a `/var/log`.

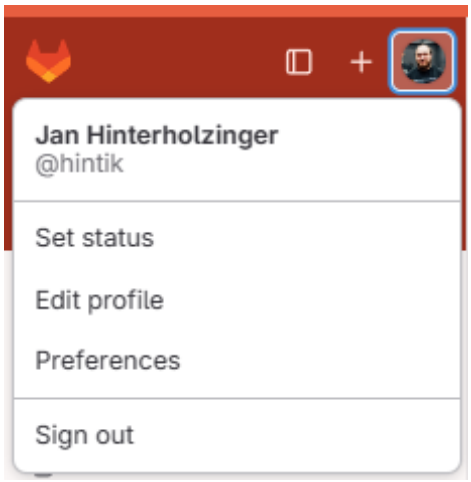
Pokud jsou splněny veškeré předchozí body, aplikace by měla být dostupná na adrese dle webového serveru. Přičemž na URL `<base_address>/student` se nachází studentský modul aplikace a na `<base_address>/admin` se nachází modul pro vyučující.

A.3 Získání osobního přístupového tokenu

Pro úspěšnou instalaci aplikace je potřebné nastavit osobní přístupový klíč autorizující požadavek k vykonávání akcí a získávání dat jako přihlášený uživatel. Následující kroky vedou k získání osobního klíče:

1. Přistoupíme na adresu využívající instance systému GitLab (např. v našem případě katedrální instance pro výuku: `https://gitlab-vyuka.kiv.zcu.cz`) a přihlásíme se.

2. Přes kliknutí na avatara uživatele a zobrazenou nabídku (viz obr. A.1) zvolíme volbu „Preferences“, tím se dostaneme do nastavení profilu uživatele.



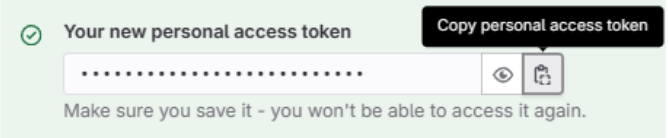
Obrázek A.1: Menu uživatelských možností pro přístup k nastavení účtu (zdroj: vlastní)

3. V levém navigačním menu zvolíme možnost „Access Tokens“, zobrazí se nám pohled se seznamem osobních přístupových kódů.

4. V hlavičce tabulky osobních přístupových kódů stiskneme tlačítko „Add new token“, zobrazí se nám možnosti vytvoření nového klíče.

5. V nabídce pro vytvoření nového klíče vyplníme název (pro naši identifikaci klíče), vhodné expirační datum klíče (volte dostatečně dlouhé, aby v průběhu semestru nedošlo k výpadku služeb aplikace z důvodu neplatného tokenu) a oprávnění tokenu zvolte `api`, `read_api`, `read_user`, `read_repository`, `read_registry` a `admin_mode`. Zejména poslední administrátorské nastavení je klíčové pro dostatečné oprávnění v rámci celého systému GitLab. Po zvolení výše uvedených možností vytvoříme token kliknutím na tlačítko „Create personal access token“.

6. Po vytvoření tokenu je zobrazen pohled, kde je zobraneno pole se skrytým tokenem (viz obr. A.2). Kliknutím na tlačítko „Copy personal access token“ se token uloží do schránky vašeho počítače. Lze jej pak následně kamkoli vložit pomocí kontextové nabídky vyvolané pravým tlačítkem myši nebo klávesovou zkratkou CTRL+V.



Obrázek A.2: Zobrazení tokenu po jeho vytvoření (zdroj: vlastní)

A.4 Nastavení Webhook komunikace

Nastavení Webhook komunikace lze nad jednotlivými repozitáři nebo ve verzi GitLab Premium i na skupinách. Tento návod bude postupovat pro nastavení Webhook nastavení pro skupiny, ale nastavení repozitářů je takřka totožné. Pro nastavení komunikace budeme potřebovat znát vygenerovaný token, který jsme zadávali v konfiguraci aplikace do nastavení `webHookSecretToken`. V první řadě přistoupíme na instanci systému GitLab, přihlásíme se (viz bod 1 sekce A.3) a budeme postupovat následujícími kroky.

1. V levém navigačním menu zvolíme položku „Groups“, kde se zobrazí skupiny, ke kterým má uživatel přístup. Pomocí tohoto seznamu se navigujeme až do skupiny, které chcemete nastavit Webhook notifikace.

2. Ve zvolené skupině zvolíme v levém navigačním menu položku „Settings“ a v podmenu možnost „Webhooks“. Zobrazí se tabulka existujících skupinových Webhook nastavení.

3. V hlavičce s výpisem skupinových Webhook klikneme na tlačítko „Add new webhook“. Zobrazí se možnosti pro vytvoření nového nastavení notifikací do externí aplikace.

4. Postupně budeme ve formuláři vyplňovat hodnoty, které nastavíme následovně:

   - **URL –** Nastavíme adresu přístupového bodu, kam se mají zasílat Webhook notifikace. Pro naši nasazovanou aplikaci se jedná o adresu `<base_address>/api/v1/web`
   - **Name –** Pro naši identifikaci můžeme volitelně vyplnit název Webhook propojení.
   - **Secret token –** Tajný token pro validaci obsahu Webhook notifikace. Zde vložíme námi vygenerovaný token a dbáme na to, aby byl totožný s tím v nastavení aplikace.
   - **Trigger –** Jako trigger (spouštěč) události notifikace zvolíme „Pipeline events“ označující zasílání notifikací při změně stavu pipeline.
   - **Enable SSL verification –** Zkontrolujeme, zda je zaškrtnuté políčko pro povolení zabezpečené komunikace pomocí SSL.

   Po vyplnění všech zmíněných položek uložíme nastavení tlačítkem „Save changes“. Po tomto nastavení je webhook nastaven.

Obsah elektronické přílohy

B

B.1 Adresářová struktura

- `/`
  - `Aplikace_a_knihovny` ..... vytvořené soubory při plnění zadání
    - `Aplikace` ..... projekt webové aplikace
    - `Negativni_sada_artefaktu` ... sada artefaktů pro negativní testy validačních skriptů
      - `01_UC`
      - `02_DB`
      - `03_RQM`
      - `04_TC`
      - `05_RF-A`
    - `Wireframes` ..... drátěné modely uživatelského rozhraní
      - `img` ..... exportované obrázky
      - `src` ..... zdrojové soubory
    - `Diagrams` ..... diagramy popisující aplikaci
      - `img` ..... exportované obrázky
      - `src` ..... zdrojové soubory
  - `Poster` ..... adresář s plakátem
    - `poster.pdf`
    - `poster.pub`
  - `Text_prace` zdrojové soubory této práce (.pdf a LaTeXsoubory s obrázky)
  - `Readme.txt` ... soubor popisující adresářovou strukturu odevzdávaného archívu

Bibliografie

[Ana21] ANASTASOV, Marko. *CI/CD Pipeline: A Gentle Introduction - Semaphore* [online]. 2021. [cit. 2024-01-10]. Dostupné z: https://semaphoreci.com/blog/cicd-pipeline.

[24a] *Aplikace pro správu semestrálních prací, jejich odevzdávání a hodnocení* [online]. Západočeská univerzita v Plzni, 2024. [cit. 2024-01-30]. Dostupné z: https://is-stag.zcu.cz/opencms/napoveda/stag-v-portalu/spnew-studium_odevzdavani-praci.html.

[AS19] AREFEEN, Mohammed Shamsul; SCHILLER, Michael. Continuous Integration Using Gitlab [online]. 2019 [cit. 2024-02-20]. Dostupné z DOI: https://doi.org/10.26685/urncst.152.

[Bes24] BESCHOKOV, Mukhadin. *GraphQL protocol* [online]. Wallarm, 2024 [cit. 2024-02-12]. Dostupné z: https://www.wallarm.com/what/what-is-graphql-definition-with-example.

[BM24] BRADY, Pádraic; MARSHALL, Dave. *Mockery Docs 1.0-alpha documentation: Mockery* [online]. 2024. [cit. 2024-04-30]. Dostupné z: https://docs.mockery.io/en/latest/.

[24b] *Contributte Datagrid* [online]. Contributte.org, 2024. [cit. 2024-02-17]. Dostupné z: https://contributte.org/packages/contributte/datagrid/.

[23a] *Contributte Doctrine-dbal* [online]. Contributte.org, 2023. [cit. 2024-01-05]. Dostupné z: https://contributte.org/packages/contributte/doctrine-dbal.html#content.

[23b] *Contributte Oauth2-client* [online]. Contributte.org, 2023. [cit. 2024-04-10]. Dostupné z: https://contributte.org/packages/contributte/oauth2-client.html.

[Cur10] CURTIS, Nathan. *Modular web design: creating reusable components for user experience design and documentation* [online]. New Riders, 2010 [cit. 2024-02-29]. ISBN 978-0-321-60135-3.

[24c] *Doctrine ORM: Getting Started with Doctrine* [online]. Doctrine Project, Doctrine Project, 2024 [cit. 2024-05-02]. Dostupné z: https://www.doctrine-project.org/projects/doctrine-orm/en/current/tutorials/getting-started.html.

[24d] *Doctrine ORM: The QueryBuilder* [online]. Doctrine Project, Doctrine Project, 2024 [cit. 2024-02-15]. Dostupné z: https://www.doctrine-project.org/projects/doctrine-orm/en/latest/reference/query-builder.html.

[GF23] GILLIS, Alexander S; FERGUSON, Kevin. *App Architecture: Dependency Injection* [online]. TechTarget, 2023 [cit. 2024-01-25]. Dostupné z: https://www.techtarget.com/searchapparchitecture/definition/dependency-injection.

[24e] *GitLab Docs: Use OpenID Connect as an authentication provider* [online]. GitLab, Inc., 2024. [cit. 2024-01-22]. Dostupné z: https://docs.gitlab.com/ee/administration/auth/oidc.html.

[24f] *GitLab Docs: Webhooks* [online]. GitLab, Inc., 2024. [cit. 2024-04-09]. Dostupné z: https://docs.gitlab.com/ee/user/project/integrations/webhooks.html.

[22] *GitLab: What is CI/CD?* [online]. GitLab, GitLab, Inc., 2022 [cit. 2024-04-03]. Dostupné z: https://about.gitlab.com/topics/ci-cd/.

[Gru23] GRUDL, David. *Nette: 7 důvodů, proč používat Nette* [online]. Nette Foundation, 2023. [cit. 2024-03-17]. Dostupné z: https://nette.org/cs/10-reasons-why-nette.

[Gru24a] GRUDL, David. *Nette Tester: Začínáme s Nette Tester* [online]. Nette Foundation, 2024. [cit. 2024-03-05]. Dostupné z: https://tester.nette.org/cs/guide.

[Gru24b] GRUDL, David. *Nette: Formuláře v presenterech* [online]. Nette Foundation, 2024. [cit. 2024-02-28]. Dostupné z: https://doc.nette.org/cs/forms/in-presenter.

[Gru24c] GRUDL, David. *Nette: Passing Dependencies* [online]. Nette Foundation, 2024. [cit. 2024-04-20]. Dostupné z: https://doc.nette.org/en/dependency-injection/passing-dependencies.

[Her22] HEROUT, Pavel. *Přednášky z OKS* [online]. 2022. [cit. 2024-03-08]. Dostupné z: https://www.kiv.zcu.cz/~herout/vyuka/oks/prednasky/oks-2022.pdf.

[Che+17] CHEN, Xianjun; JI, Zhoupeng; FAN, Yu; ZHAN, Yongsong. Restful API Architecture Based on Laravel Framework. In: [online]. 2017, sv. 910 [cit. 2024-01-18]. Č. 1. Dostupné z DOI: 10.1088/1742-6596/910/1/012016. Cited by: 24; All Open Access, Gold Open Access.

[Ják17] JÁCHYMOVÁ, Anežka. *Rozšíření funkčnosti validačního serveru a jeho testování* [online]. Plzeň: Západočeská univerzita v Plzni, 2017 [cit. 2024-02-25]. Dostupné z DOI: https://doi.org/67720. Bakalářská práce. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky. Vedoucí práce Ph.D. DOC. ING. PAVEL HEROUT.

[Kho20] KHORIKOV, Vladimir. *Unit Testing Principles, Practices, and Patterns* [online]. Simon a Schuster, 2020 [cit. 2024-04-22].

[Mar00] MARTIN, Robert C. Design principles and design patterns. *Object Mentor* [online]. 2000, roč. 1, č. 34, s. 597 [cit. 2024-04-28]. Dostupné z: https://labs.cs.upt.ro/labs/ip2/html/lectures/2/res/Martin-PrinciplesAndPatterns.PDF.

[MEM04] MILI, Hafedh; ELKHARRAZ, Amel; MCHEICK, Hamid. Understanding separation of concerns [online]. 2004 [cit. 2024-01-15].

[Ric15] RICHARDS, Mark. *Software architecture patterns*. Sv. 4 [online]. O'Reilly Media, Incorporated 1005 Gravenstein Highway North, Sebastopol, CA …, 2015 [cit. 2024-04-05]. ISBN 978-1-09-813427-3.

[Sak+14] SAKIMURA, Nat; BRADLEY, John; JONES, Mike; DE MEDEIROS, Breno; MORTIMORE, Chuck. OpenID Connect Core 1.0 incorporating errata set 1. *The OpenID Foundation, specification* [online]. 2014, roč. 335 [cit. 2024-04-15].

[18] *Soft UI Dashboard Bootstrap* [online]. Tim, Creative, 2018. [cit. 2024-02-05]. Dostupné z: https://www.creative-tim.com/learning-lab/bootstrap/overview/soft-ui-dashboard.

[24g] *SOLID Design Principles: Object Oriented Design* [online]. 2024. [cit. 2024-02-14]. Dostupné z: https://www.oodesign.com/design-principles.

[24h] *Symfony: Logging* [online]. Symfony, Symfony, 2024 [cit. 2024-03-02]. Dostupné z: https://symfony.com/doc/current/logging.html.

[SW14] SYROMIATNIKOV, Artem; WEYNS, Danny. A Journey through the Land of Model-View-Design Patterns. In: [online]. 2014, s. 21–30 [cit. 2024-02-10]. ISBN 978-1-4799-3412-6. Dostupné z DOI: 10.1109/WICSA.2014.13.

[Val08] VALENTA, Lukáš. Praktické užití skriptovacího jazyka v Javovské aplikaci–validátor studentských prací. *Sborník příspěvků* [online]. 2008, s. 57 [cit. 2024-03-20]. Dostupné z: `https://europen.cz/Anot/33/HLAVNI.pdf#page=57`.

Seznam zkratek

API Aplication Programming Interface.

CD Continuous Deployment.

CI Continuous Integration.

CIV Centrum informatizace a výpočetní techniky.

CLI Command Line Interface.

CRON nástroj pro plánování úloh.

CRUD Create, Read, Update, Delete.

CSFR Cross Site Request Forgery.

CSS Cascading Style Sheets.

CSV Comma-Separated Values.

DB1 Databázové systémy 1.

DBAL Database Abstraction Layer.

DevOps Development and Operations.

DI Dependency Injection.

DOP Doporučený rozsah.

DQL Doctrine Query Language.

E2E End to End.

ERD Entity Relationship Diagram.

FAV Fakulta aplikovaných věd.

HTML Hypertext Markup Language.

HTTP Hypertext Transfer Protocol.

HTTPS Hypertext Transfer Protocol Secure.

IS Informační systém.

JSON JavaScript Object Notation.

KIV Katedra informatiky a výpočetní techniky.

MVC třívrstvá architektura Model-View-Controller.

MVP třívrstvá architektura Model-View-Presenter.

NEON Nette Object Notation.

OIDC OpenID Connect.

OKS Ověřování kvality software.

OOP Objektově orientované programování.

ORM Object Relational Mapper.

PHP PHP: Hypertext Preprocessor.

PMIN Minimální rozsah.

RF Robot Framework.

RFBL Robot Framework a Browser Library.

RQM Requirement.

SoC Separation of Concerns.

SOLID Single-responsibility, Open-closed, Liskov substitution, Interface segregation and Dependency inversion principles.

SQL Structured Query Language.

SSL Secure Sockets Layer.

SSO Single Sign-on.

STAG Studijní agenda.

TC Test Case.

UC Use Case.

UI User Interface.

URL Uniform Resource Locator.

UX User Experience.

WAMP Windows, Apache, MySQL, PHP.

WEB Webové aplikace.

XSS Cross Site Scripting.

YAML Ain't Markup Language.

ZPP Základy programátorské praxe.

ZČU Západočeská univerzita v Plzni.

Seznam obrázků

Seznam výpisů