FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Aplikace pro vyhodnocení splnění úkolů ve vývojových nástrojích
Student:	Anna Churina
Vedoucí:	Ing. Jiří Mlejnek
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2023/2024

Pokyny pro vypracování

Cílem práce je navrhnout a implementovat aplikaci, která umožní automatické vyhodnocení splnění definovaných úkolů v nástrojích pro podporu tvorby softwarových projektů. Aplikace bude využívána v rámci předmětu BI-IDO k rychlé kontrole stavu plnění zadaných úkolů.

1. Analyzujte požadavky a navrhněte řešení umožňující definování požadovaných úkolů a jejich automatické vyhodnocení. Typicky se bude jednat o úkoly vztažené ke stavu git repository (existence větví, commitů, tagů) a nastavení projektu v rámci nástroje GitLab. Rozsah možností, které aplikace pro definici úloh bude nabízet, konzultujte s vedoucím práce.
2. Na základě provedeného návrhu proveďte její implementaci.
3. Implementované řešení otestujte, zdokumentujte a připravte pro nasazení do pilotního provozu.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 15. září 2022 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Aplikace pro vyhodnocení splnění úkolů ve vývojových nástrojích

Anna Churina

Katedra softwarového inženýrství
Vedoucí práce: Ing. Jiří Mlejnek

10. května 2023

Poděkování

Ráda bych poděkovala vedoucímu mé práce panu Ing. Jiřímu Mlejnkovi za odborné vedení, konzultace a všechny rady. Dále bych chtěla poděkovat rodině za podporu a trpělivost.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principu při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisu. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisu, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programu, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli zpusobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelum). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným zpusobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 10. května 2023 . . . . . .

Odkaz na tuto práci

Churina, Anna. *Aplikace pro vyhodnocení splnění úkolů ve vývojových nástrojích*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2023.

Abstrakt

Tato bakalářská práce se zabývá vývojem aplikace pro vyhodnocení splnění úkolů ve vývojových nástrojích. Aplikace slouží ke kontrole a automatickému vyhodnocení úkolů, které budou splněny studenty v rámci předmětů na Fakultě informačních technologií Českého vysokého učení technického v Praze. Primárně aplikace bude využívaná pro potřeby předmětu Úvod do DevOps. Analytická čast práce se zabývá analýzou požadavků na aplikaci, návrhem způsobu definování úkolů, výběrem technologií pro implementaci a návrhem aplikace. Výstupem práce je naimplementovaná aplikace, která umožnuje evidovat splnění úkolů studenty.

Klíčová slova webová aplikace, automatické vyhodnocení úkolů, vývojový nástroj, Úvod do DevOps, Gitlab, Java

Abstract

This bachelor's thesis deals with the development of an application for evaluating the fulfillment of tasks in development tools. The application serves to monitor and automatically evaluate the tasks that will be completed by students within the subjects at the Faculty of Information Technologies of the

Czech Technical University in Prague. The application will primarily be used for the purposes of the Introduction to DevOps course. The analytical part of the thesis deals with the analysis of the requirements for the application, the design of the way of defining tasks, the selection of technologies for implementation and the design of the application. The output of the work is an implemented application that allows to record the completion of students' tasks.

Keywords web application, automatic tasks evaluation, development tool, Introduction to DevOps, Gitlab, Java

Obsah

Seznam obrázků

Seznam tabulek

Úvod

V dnešním světě se informační technologie velmi rychle rozvíjejí. Je těžké si představit život bez internetu, ukládání obrovského množství dat v elektronické podobě a používání aplikací. To vše pomáhá zkrátit čas, který může být věnován konkrétnímu úkolu nebo práci. Taková automatizace v různých oblastech činnosti člověka je v současné době velmi aktuální.

I na univerzitách je často potřeba systémů nebo aplikací, které poslouží ke zjednodušení práce učitelů a učení studentů. Děje se tak i na Fakultě informačních technologií Českého vysokého učení technického v Praze. V rámci předmětů vyučovaných na fakultě se využívají různé aplikace, například, kde student má zobrazeny informace o předmětu nebo kde má možnost přečíst a vyplnit zadání a odevzdat ho k ověření.

Aplikace vytvořená v této práci automaticky vyhodnocuje odevzdané úkoly studentů ve vývojových nástrojích systému, přičemž je kladen důraz na úkoly související s prací v git repositáři. Primárně bude využívána v rámci předmětu Úvod do DevOps. Aplikace nabízí několik funkcí. Umožňuje cvičícím nebo jiné zodpovědné osobě vytvořit úkol zabývající se daným tématem, který lze přizpůsobit potřebám daného předmětu pomocí zadání různé variace parametrů. V neposlední řadě poskytuje možnost generování reportů obsahující informace o počtu splněných úkolů a získaných bodů daného studenta.

Práce je strukturovaná do tří hlavních kapitol - analýza, návrh a implementace. První kapitola se zabývá analýzou průběhu předmětu Úvod do DevOps, definici hlavních typů úkolů a vytvoření způsobu jejich definování v rámci aplikace, analýzu požadavků a výběr vhodných technologií. Poté následuje kompletní návrh aplikace na základě provedené analýzy. A celá práce je zakončena implementací a testováním navržené aplikace.

Aplikaci je možné přizpůsobit i pro výuku dalších předmětů, které využívají verzovací systém git nebo jiné vývojové nástroje.

Kapitola 1

Cíl práce

Cílem bakalářské práce je navrhnout a implementovat aplikaci, která umožní automatické vyhodnocení splnění definovaných úkolů v nástrojích pro podporu tvorby softwarových projektů. Primárním zaměřením jsou úkoly vztažené ke stavu git repozitáře a nastavení projektu pomocí nástroje GitLab.

Aplikace bude využívána na Fakultě informačních technologií Českého vysokého učení technického v Praze a bude sloužit k rychlé kontrole stavu plnění zadaných úkolů v rámci předmětů, které jsou vyučovány na fakultě.

Dílčím cílem je zanalyzovat požadavky na aplikaci, určit způsob definování úkolů a zvolit vhodné technologie pro implementaci. Na základě provedené analýzy bude vytvořen kompletní návrh aplikace. Dalším cílem je aplikaci naimplementovat, otestovat, zdokumentovat a připravit pro nasazení do provozu.

Kapitola 2

Analýza

Tato kapitola rozebírá provedenou analýzu. Jelikož aplikace bude sloužit pro vyhodnocení úkolů, tak nejprve je potřeba zjistit jaké typy úkolů by měla aplikace podporovat. Tím se zabývá sekce 2.1. Cílem analýzy následující sekce je najít vhodný způsob definování úkolů v aplikaci. Dalším krokem je zajistit vyhodnocení úkolů, analýza tohoto procesu je popsána v sekce 2.3. Pak se nachází sekce 2.4, která se zabývá možnosti využití aplikace pro další předměty vyučované na Fakultě informačních technologií. Pro tvorbu aplikace je často potřeba vytvořit doménový model, zanalyzovat uživatelské požadavky a popsat případy užití. To je cílem sekcí 2.5 a 2.6.

2.1 Definování typů úkolů

Tato sekce se zabývá definováním typů úkolů, které by měla aplikace podporovat, to znamená, že takový typ úkolu je možné do aplikace zadat a pak vyhodnotit. V podsekcích jsou popsány analýza průběhu předmětu BI-IDO a výsledné typy úkolů.

2.1.1 Průběh předmětu BI-IDO: Úvod do DevOps

Pro zjištění, jaké typy úloh by měla aplikace podporovat, byl prostudován průběh předmětu Úvod do DevOps [1]. Předmět se zabývá tématem DevOps[2] a připravuje budoucí vývojáře a administrátory na moderní kulturu vývoje a provozu systémů a služeb. Jednou z tematických částí výuky je práce s nastrojí pro podporu vývoje softwarových projektů.

Během cvičení studenti dostávají úkoly související s prací ve vývojových nástrojích. To jsou nástroje, které slouží pro podporu vývoje softwarového projektu. Příkladem nástrojů, které jsou využívány v rámci předmětu BI-IDO, jsou GitLab [3] a SonarQube [4]. Zaměřením této bakalářské práce jsou úkoly spojené s prací ve vývojovém nástroji GitLab.

Pro úspěšné absolvovaní předmětu studenti potřebují dostat určitý počet bodů, které je možné získat po splnění bodovaných úkolů. Vyučující tyto úkoly kontrolují a zapisují studentům body. Aplikace zautomatizuje proces vyhodnocení.

Výsledkem analýzy jsou různé typy úkolů, které jsou popsány v následující části práce.

2.1.2 Typy úkolů

Jelikož úkoly, jimiž se tato bakalářská práce zabývá, souvisejí se stavem git repositáře a nastavení projektu ve vývojovém nástroji GitLab, tak je potřeba mít slovník, kde jsou uvedené názvy git komponent nebo prvků v českém a anglickém jazyce. Slovník je uveden v tabulce 2.1.

Na základě analýzy průběhu předmětu byly identifikovány komponenty nástroje GitLab, které bude aplikace kontrolovat a vyhodnocovat. Typy úloh odpovídají typům GitLab komponent.

Seznam komponent pro úkoly:

- Projekt;
- Tiket;
- Milník;
- Merge request;
- Štítek;
- Nástěnka;
- Větev;
- Commit;
- Pipeline;
- Úloha.

Kromě rozdělení úkolů podle typu komponenty jsou úkoly také rozděleny podle počtu prvků, které mají být současně kontrolovány. To znamená, že pro každý typ komponenty je možné nadefinovat úkol pro kontrolu jednoho určitého prvku nebo pro kontrolu existence více prvků.

V rámci této bakalářské práce bylo rozhodnuto vytvořit implementaci vyhodnocení úkolů, které se vztahují k těmto komponentám:

- Tiket;
- Milník;

Tabulka 2.1: Slovník pojmů

Název v češtině	**Název v angličtině**
Commit	Commit
Fáze	Stage
Merge request	Merge request
Milník	Milestone
Nástěnka	Board
Pipeline	Pipeline
Projekt	Project
Řešitel	Assignee
Štítek	Label
Tiket	Issue
Úloha	Job
Větev	Branch

- Větev;
- Commit;
- Několik tiketů;
- Několik milníků;
- Několik větví;
- Několik commitů.

2.2 Analýza způsobů zadávání úkolů do aplikace

Pro lepší strukturalizaci úkolů v aplikaci bude možné vytvářet témata, v rámci kterých budou zadávány. Jelikož je nutné mít možnost nadefinovat hodně různých úkolů, byla zvolena následující struktura úkolu.

Struktura úkolu:

1. Název
2. Povinné parametry
3. Volitelné parametry

Každý typ úkolu má definované své vlastní parametry.

Parametry pro kontrolu jedné komponenty jsou uvedené v tabulce 2.2. Pokud parametrem pro kontrolu jedné komponenty bude počet komponent jiného typu, tak kontrola se bude provádět pomocí operátoru >=. Tedy pokud komponenta obsahuje jiné komponenty, jejichž počet je větší nebo roven počtu zadanému jako parametr, tak student dostane body za splnění tohoto parametru úkolu. V dalších verzích aplikace to bude možné změnit a přidat ke každému parametru úkolu typu „počet“ atribut, který bude obsahovat operátor, podle kterého by se měla provádět kontrola. Tento způsob povolí mít několik parametrů, které označují počet komponent, v jednom úkolu, protože pro každý parametr bude zadefinován operátor, podle kterého by se mělo provádět porovnání.

Tabulka 2.2: Paramentry úkolů pro kontrolu jedné komponenty

Typ úkolu podle komponenty	**Povinné parametry**	**Volitelné parametry**
Projekt	Název projektu	Úroveň viditelnosti, metoda sloučení projektů, výchozí větev, seznam členů projektu
Tiket	Název tiketu	Řešitel, milník, seznam štítků
Milník	Název milníku	Popis, datum splatnosti, datum zahájení, počet tiketů
Merge Request	Zdrojová větev, cílová větev	Řešitel, stav sloučení
Štítek	Název štítku	Barva, popis, priorita, počet tiketů
Nástěnka	Název nástěnky, seznam štítků	
Větev	Název větve	počet commitů
Commit	Větev	Zpráva
Pipeline		Status, seznam úloh
Úloha		Seznam artefaktů, fáze

Úkoly sloužící pro kontrolu existence několika komponent budou obsahovat jako parametry počet, operátor >, >=, <, <= nebo = a nepovinný filtr nebo filtry, podle kterých se bude provádět kontrola. Filtrem může být parametr komponenty nebo název jiné komponenty, která je spojená s kontrolovanými aplikací. Seznam filtru je uveden v tabulce 2.3.

Tabulka 2.3: Paramerty pro kontrolů několika komponent

Typ úkolu podle komponenty	**Filtry**
Tiket	Milník, štítek, řešitel
Merge Request	Zdrojová větev, cílová větev, řešitel, stav sloučení
Štítek	Barva, priorita
Commit	Větev, status
Pipeline	Status
Úloha	Fáze

Pro zadaní některých parametrů bude možné použit masku „*text*“, kde místo „text“ je fixní text, který by student měl mít ve svém řešení, a místo „*“ se může nacházet libovolný text.

Dále pro znázornění různých typů úkolů jsou uvedené konkrétní příklady.

1. Vytvořte v projektu tiket, který bude splňovat následující kritéria:

   - Části názvu tiketu je „issue1“;
   - Řešitelem tiketu je uživatel s uživatelským jménem „student“;
   - Tiket patří do milníku s názvem „Iterace2“;
   - Tiket je označen štítky „TODO“ a „Priorita: High“.

   Dále je uveden příklad zadaní úkolu do aplikace:

   Příklad 1 (kontrola jedné komponenty):
   Název úkolu:
   Vytvoření tiketu
   Popis úkolu:
   V projektu vytvořte tiket, který bude splňovat zadaná kritéria.
   Parametry:
   Název tiketu: *issue1*
   Řešitel: student
   Milník: Iterace2
   Seznam štítků: TODO, Priorita: High

2. V projektu vytvořte minimálně 5 tiketů, které budou splňovat následující kritéria:

   - Tikety patří do milníku s názvem „Iterace2“;
   - Tikety jsou označeny štítkem „TODO“.

   Dále je uveden příklad zadaní úkolu do aplikace:

   Příklad 2 (kontrola několika komponent):
   Název úkolu:
   Vytvoření několika tiketů
   Popis úkolu:
   V projektu vytvořte minimálně 5 tiketů, které budou splňovat zadaná kritéria.
   Parametry:
   Počet prvků: 5
   Operátor: >
   Milník: Iterace2
   Štítek: TODO

2.3 Vyhodnocení úkolů a reporty

V této sekci jsou popsány způsob vyhodnocení úkolů a způsob zobrazení výsledků pomocí reportů.

2.3.1 Způsob vyhodnocení úkolů

Důležitou částí aplikace je poskytnout možnost zkontrolovat a vyhodnotit splnění úkolů studenty. Pro tyto účely bude možné vygenerovat reporty, na kterých budou zobrazeny výsledky studentů v bodech nebo v procentech.

U každého úkolu bude provedená kontrola autorství. Bude porovnáván vlastník projektu s uživatelem, který splnil úkol.

Maximum bodů, který může dostat student, určuje vyučující, který úkol vytvořil. Jelikož úkol může obsahovat povinné a nepovinné části, počet bodů může být i neúplný. Při nesplnění povinného parametru úkolu by student měl dostat 0 bodů.

2.3.1.1 Částečné vyhodnocení úkolů

Existují různé způsoby implementace částečného vyhodnocení úkolu.

Jeden je rovnoměrně rozdělit body mezi parametry úkolu. To znamená, že učitel zadefinuje jenom maximální počet bodů. Pak na základě celkového počtu parametrů se body rovnoměrně rozdělí. Tento způsob nepovoluje kontrolu počtu bodů pro každý parametr, ale je lehčí vzhledem k implementaci.

Druhý způsob je zadávat určitý počet bodů pro každý parametr. To znamená, že učitel by měl nadefinovat kolik bodů stojí splnění každého parametru úkolu. Výhodou je mít možnost dávat menší počet bodů za splnění nepovinného parametru a vetší za splnění povinného.

Dalším způsobem je zadefinovat maximální počet bodů za splnění úkolu a pro každý parametr zadat procento, které bude označovat podíl na celkovém počtu bodů. Výhodou tohoto způsobu je možnost nastavit prioritu parametru, bez konkrétního počítaní bodů pro každý parametr ručně.

Pro první verzi aplikace, která bude implementována v rámci této bakalářské práce, bylo rozhodnuto použit první způsob částečného hodnocení. Tedy rovnoměrně rozdělovat body mezi parametry.

2.3.2 Typy reportů

Aplikace bude poskytovat několik typů reportů pro zobrazení informací potřebných pro různé účely.

Typy reportů:

1. Report podle jedné skupiny a jednoho tématu 2.1.

   Tento report bude sloužit jako hlavní kontrola splnění úkolů. V tabulce bude možné vidět kolik bodů z maximálního počtu dostal každý student ze skupiny za určitý úkol v tématu a celkový počet bodů v rámci tématu. V reportu budou uvedeny odkazy na projekty studentů, které byli kontrolovány.

2. Report podle jedné skupiny a všech témat 2.2.

   V tomto reportu bude možné zobrazit kolik bodů dostal každý student ze skupiny za každé téma. Report bude sloužit pro zobrazení celkového hodnocení studentů za semestr. Stejně jako v prvním reportu budou uvedeny odkazy na projekty studentů.

3. Report podle všech skupin a všech témat 2.3.

   Tento report slouží k zobrazení statistik o úspěšnosti splnění úkolů studenty každé skupiny podle témat v procentech.

V rámci této bakalářské práce bylo rozhodnuto udělat implementaci reportu typu 1, tedy reportu podle jedné skupiny a jednoho tématu.

2.4 Potřeby dalších předmětů

Aplikace může být využívána i pro účely dalších předmětů vyučovaných na FIT ČVUT v Praze, které obsahují úkoly spojené s prací ve vývojových nástrojích. Každý předmět má svá specifika, a proto se můžou nástroje a úkoly

Skupina 1	Uživatelské jméno studenta	Odkaz na projekt studenta	Téma 1						Celkový počet bodů
			Číslo úkolu						
			1	2	3	4	5	6	
	username1	project link 1							
	username2	project link 2							
	username3	project link 3							
	username4	project link 4							
	username5	project link 5							
	username6	project link 6							
	username7	project link 7							
	username8	project link 8							

Obrázek 2.1: Report 1

Skupina 1	Uživatelské jméno studenta	Odkaz na projekt studenta	Název tématu						Celkový počet bodů
			Téma 1	Téma 2	Téma 3	Téma 4	Téma 5	Téma 6	
	username1	project link 1							
	username2	project link 2							
	username3	project link 3							
	username4	project link 4							
	username5	project link 5							
	username6	project link 6							
	username7	project link 7							
	username8	project link 8							

Obrázek 2.2: Report 2

		Název tématu						Procento splnění úkolů
		Téma 1	Téma 2	Téma 3	Téma 4	Téma 5	Téma 6	
Název skupiny	Skupina 1							
	Skupina 2							
	Skupina 3							
	Skupina 4							
	Skupina 5							
	Skupina 6							
	Skupina 7							
	Skupina 8							

Obrázek 2.3: Report 3

lišit. V následující části je rozebrán popis možností využití aplikace v rámci jiných předmětů.

2.4.1 BI-GIT.21: Technologie pro vývoj SW

V rámci předmětů BI-GIT.21 [5] studenti pracují s git repozitářem. Pro úspěšné zakončení předmětu byt student měl dostat minimálně 50 bodů. Během semestru má několik způsobů jak tyto body získat.

První možnost je vypracovat tři domácí úlohy za celkem 25 bodů. Úlohy jsou spojené s prací v git repozitářích na serveru gitc.cz. Po odesílaní úlohy zpět na server se provádí automatické vyhodnocení a body se posílají do systému pro zobrazení výsledků studenta ze všech předmětů Grades [6].

Další možnost jak dostat 50 bodů je udělat semestrální práci, během které studenti opravují několik git repozitářů. Odevzdaní pobíhá stejným způsobem jako u domácích úloh. Výsledný repozitář studenti posílají na server, na kterém se následně provede automatické vyhodnocení.

Alternativou k semestrální práci je vedení git repozitářů z předmětu Programování a algoritmizace 1 [7]. Pro získání bodů by studenti měli splnit několik různých požadavků na repozitáře spojených s jejich strukturou, formátem a uživatelem. Kontrola probíhá automaticky při odeslání repozitáře na GitLab.

Z analýzy průběhu a hodnocení předmětu BI-GIT vyplývá, že automatické vyhodnocení práce studentů už je implementováno se zaměřením na potřeby tohoto předmětu. Aplikace, která bude vytvořena v této bakalářské práci může poskytnout další funkcionalitu. Příkladem je zobrazení statistik o výsledcích studentů z různých skupin a semestrů vyučujícím nebo možnost pro studenty se podívat na všechny své výsledky. Ačkoliv taková funkcionalita by vyžadovala přidání dalších typů úkolů, případně i další vývojové nástroje.

2.4.2 BI-PA2: Programování a algoritmizace 2

Předmět BI-PA2 [8] je zaměřen na objektově orientované programování v programovacím jazyce C++ [9]. Pro získaní zápočtu z předmětu by studenti měli dostat minimálně 35 bodů, přičemž mají několik možností jak toho dosáhnout. Těmito možnostmi jsou domácí úlohy, procvičovací úlohy, znalostní testy, aktivita na cvičeních a semestrální práce.

Semestrální prací je vývoj funkční aplikaci na zadané téma splňující stanovené požadavky. Jedním z takových požadavků je použití repozitáře, který je založen pro každého studenta na fakultním GitLabu. V případe, že student nepoužíval tento repozitář, jeho hodnocení bude nižší o cca 10 %.

Aplikace, vývojem které se zabývá táto bakalářská práce, může sloužit pro rychlou automatickou kontrolu využití GitLabu skupinou studentů bě-

hem zpracování semestrální práce. Jednou z možností je zobrazení výsledků kontroly existence určitého počtu větví nebo commitů v projektu.

2.5 Doménový model

Doménový model je výkonný mechanismus pro zaznamenávání a definování byznys pojmů, které jsou identifikovány během analýzy požadavků. Poskytuje jedinou definici pojmů a jejich vztahů, na které lze odkazovat odkudkoli v rámci modelu.[10]

Doménový model je vytvořen pomocí UML diagramu, v němž jsou zobrazeny entity a vztahy mezi nimi. Na obrázku 2.4 je doménový model pro aplikaci vytvořenou v této bakalářské práci.

Dale je uveden popis entit doménového modelu.

Entita Uživatel

Entita představuje uživatele, který bude aplikaci používat. Pro každého uživatele budou uloženy následující data – uživatelské jméno, heslo, jméno a příjmení.

Entita Role

Entita představuje role, kterou může mít uživatel. Podle roli budou rozděleny práva přístupu k různým funkcionalitám, které aplikace poskytuje.

Entita Student

Entita reprezentuje studenta, tedy uživatele, který má roli student, jehož projekty se budou automaticky vyhodnocovat. Entita má atribut odkaz, který označuje odkaz na profil studenta ve vývojovém nástroji.

Entita Administrátor

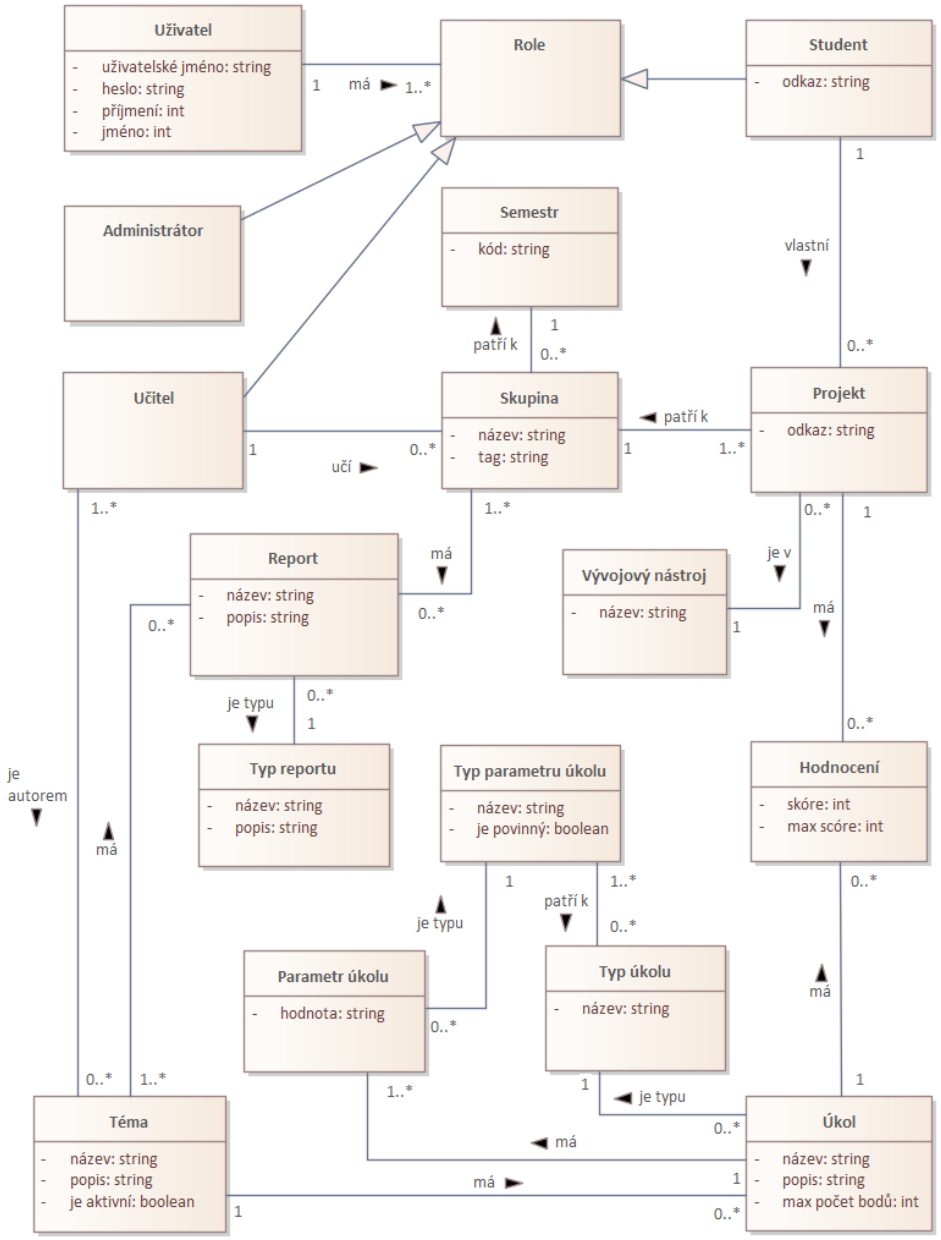
Entita reprezentuje administrátora, tedy uživatele, který má roli administrátor. Takový uživatel bude mít přistup ke všem funkcionalitám aplikace, včetně správy jiných uživatelů.

Entita Učitel

Entita reprezentuje učitele, tedy uživatele, který má roli učitel. Učitel bude moct vytvářet témata, úkoly, skupiny a generovat reporty s výsledky studentů.

Entita Projekt

Entita představuje projekt studenta. Entita má atribut odkaz, hodnotou kterého je odkaz na projekt na stránce vývojového nástroje.



Obrázek 2.4: Doménový model

Entita Vývojový nastroj

Entita představuje vývojový nastroj a slouží pro klasifikaci projektu. To znamená, že vyhodnocení úkolů pro určitý projekt bude probíhat podle názvu nástroje, který je atributem entity. Příkladem názvu může být „Gitlab“ nebo „SonarQube“.

Entita Semestr

Entita reprezentuje semestr, ke kterému budou přiřazeny skupiny. Každý semestr má svůj unikátní kód.

Entita Skupina

Entita reprezentuje skupinu projektů studentů, které se budou spolu vyhodnocovat. Atributy jsou název skupiny a tag, podle kterého se budou skupiny načítat, nebo-li je to text, který by se měl nacházet v popisu projektu.

Entita Report

Entita reprezentuje definice reportu, který je potřeba vygenerovat pro zobrazení výsledků studentů. Každá definice obsahuje název a popis a patři k určitému typu, který se zadává během jejího vytváření.

Entita Typ reportu

Entita reprezentuje typ reportu, který definuje kolik témat a skupin se budou vyhodnocovat zároveň při generovaní jednoho reportu. Entita má atributy název a popis. Název je unikátní pro každý typ.

Entita Téma

Entita reprezentuje jedno téma, do kterého může patřit několik úkolů. Atributy jsou název, popis a stav „je aktivní“, který označuje jestli téma je aktivní a úkoly v něm jsou aktuální.

Entita Úkol

Entita reprezentuje úkol, který bude v aplikace zadefinován a pak vyhodnocen v projektech studentů. Úkol obsahuje atributy název, popis a maximální počet bodů, který může student dostat za jeho splnění.

Entita Typ úkolu

Entita reprezentuje typ úkolů. Pro každý typ úkolů je zadefinován seznam parametru, které je možné do úkolu zadat. Atributem entity je unikátní název.

Entita Typ parametru úkolu

Jelikož několik typů úkolů mohou mít v sobě stejný parametr, tak bylo rozhodnuto vytvořit entitu Typ parametru úkolu. Tahle entita bude mít v sobě název typu parametrů, například „Počet tiketů“, a informaci, jestli parametr

je povinný pro splnění, tedy jestli student může dostat body bez splnění tohoto parametru úkolu.

Entita Parametr úkolu
Entita reprezentuje parametr určitého úkolů, který obsahuje hodnotu pro kontrolu. Například, pro typ parametru „Počet tiketů“ hodnotou může být číslo 3.

Entita Hodnocení
Entita reprezentuje hodnocení, které se vytvořilo po kontrole projektu studenta podle určitého úkolu. Jelikož aplikace podporuje i částečné vyhodnocení, tak entita má atribut skóre a max skóre. Skóre odpovídá skutečnému počtu bodů studenta, zatímco max skóre odpovídá maximální možnému počtu bodů, které lze získat za splnění úkolu.

2.6 Analýza požadavků

Analýza požadavků je důležitou součástí procesu vývoje softwaru. Je to proces definování očekávání uživatelů pro nový software, který se vytváří nebo upravuje [11]. Požadavky slouží pro zjištění přesnějšího odhadu pracnosti vývoje. Výsledná aplikace bude tyto požadavky splňovat.

Rozdělení požadavků do kategorií může pomoci zorganizovat proces vývoje a zajistit, aby byly vyřešeny všechny aspekty aplikace. Požadavky lze rozdělit na funkční a nefunkční. Také bylo rozhodnuto požadavky na danou aplikaci roztřídit do dvou kategorii podle priority splnění:

- Kategorie 1:

  Požadavky, které jsou kritické a musí být zahrnuty do aplikace, aby dosáhla svého hlavního cíle. Bez těchto požadavků nebo vlastností nelze aplikace považovat za úspěšný.

- Kategorie 2:

  Požadavky, které jsou důležité, ale ne kritické. Tyto požadavky nebo funkce mohou být v případě potřeby odloženy, ale stále jsou důležité pro dosažení cílů projektu.

2.6.1 Funkční požadavky

Funkční požadavky jsou specifikace funkcí (vlastností) produktu. Funkční požadavky definují, co přesně musí software dělat a jak musí systém reagovat na vstupy [12].

FP1: Definování úkolů
Kategorie: 1

Aplikace bude umožňovat nadefinovat úkoly pro práci ve vývojových nástrojích. Úkoly budou různých typů. Ke každému úkolu bude možné zadat počet bodů, který student dostane po jeho správném splnění. Aplikace bude podporovat i částečné splnění úkolů, tedy student může dostat neúplný počet bodů, které během definování zadal vyučující. Úkoly budou rozděleny do různých témat, které budou vytvořeny v aplikaci.

FP2: Vytváření skupin
Kategorie: 1

V aplikaci se automatický pomocí načtení dát z vývojového nástroje vytvoří skupiny studentů. Každá skupina bude mít svůj název a seznam projektů studentů, ve kterých budou pracovat. Bude možné zobrazit i seznam uživatelských jmen studentů. Pro vývojový nastroj GitLab vytváření skupin by mělo probíhat pomocí filtrování popisu projektu podle určitého tagu, který bude možné zadat do aplikace, s použitím masky, která je popsána v sekci 2.2.

FP4: Automatické vyhodnocení úkolů a generování reportů
Kategorie: 1

Aplikace umožní vygenerovat různé typy reportů, které jsou popsány v sekci 2.3.2. Report slouží pro zobrazení výsledků studentů, tedy kolik bodů dostal student za určitý úkol, celkový počet bodů za téma nebo procento splnění všech úkolů skrz všechna témata. Během generování reportu bude provedeno automatické vyhodnocení úkolů, které studenti splnili.

FP5: Export reportů
Kategorie: 2

Aplikace bude poskytovat možnost exportu reportů s výsledky studentů ve formátu JSON a CSV.

FP6: Kontrola autorství
Kategorie: 2

Aplikace bude porovnávat autora změn v projektu a vlastníka projektu. Cílem je zkontrolovat, jestli student dělal úkol sám.

FP7: Evidence studentů
Kategorie: 1

Aplikace bude evidovat informace o studentech. Pro každý semestr bude vytvořeno několik skupin podle paralelky. Pro jednoho studenta budou uložené osobní údaje a odkazy na jeho projekty ve vývojových nástrojích.

FP8: Ukládaní výsledků
Kategorie: 2
Aplikace bude mít uložené informace o stavu splnění úkolů studentu. Také se v aplikaci bude ukládat historie vyhodnocení, co poskytne možnost se podívat na výsledky studentů z minulých let.

FP9: Správa uživatelů
Kategorie: 1
Administrátor bude moct spravovat role uživatelů a jejich účty.

2.6.2 Nefunkční požadavky

Nefunkční požadavky určují, jakým způsobem má aplikace splňovat funkcionalitu. Pomocí nefunkčních požadavků definujeme omezení a standarty na aplikaci.

NF1: Webová aplikace
Kategorie: 1
Aplikace bude dostupná přes web.

NF2: Lokální autentizace
Kategorie: 1
Do aplikace budou mít přístup pouze oprávnění uživatelé, kteří se přihlásili pomocí uživatelského jména a hesla.

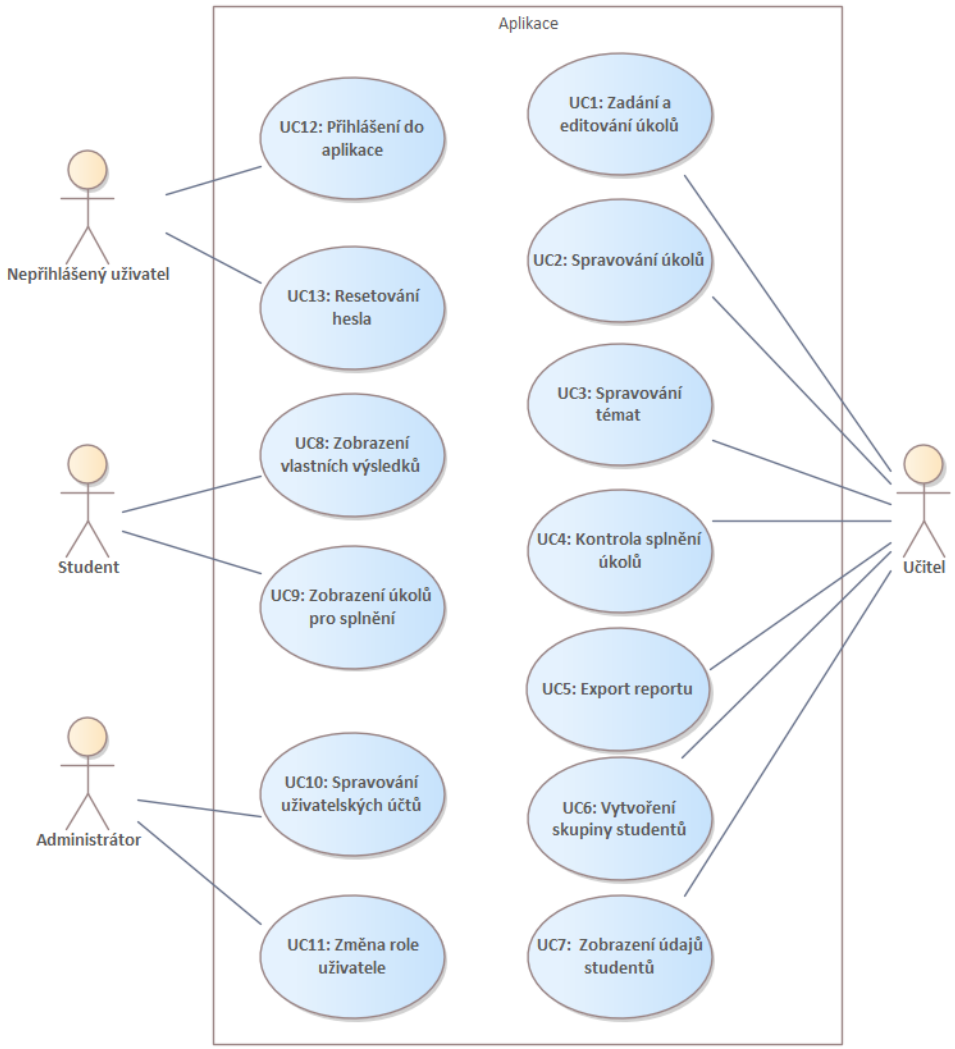
NF3: Autentizace přes SSO
Kategorie: 2
Do aplikace budou mít přístup uživatelé, kteří se přihlásili pomocí Single Sign On.

NF4: Uživatelské rozhraní
Kategorie: 1
Aplikace bude poskytovat grafické webové rozhraní.

NF5: Bezpečnost
Kategorie: 1
Pro komunikaci s aplikací bude používán šifrovaný protokol HTTPS.

2.7 Analýza případů užití

Pro detailní popis funkcionality aplikace z pohledu uživatele byl využit model případů užití(use cases) nastiňující kroky, které uživatel podnikne, interakce mezi uživatelem a systémem a očekávané výsledky. Případy užití poskytují způsob vizualizace požadavků na aplikaci a často se používají k identifi-



Obrázek 2.5: Model případů užití

kaci a stanovení priority funkčních požadavků. Model případu užití je na obrázku 2.5.

2.7.1 Aktéři

Aktér případu užití je osoba, systém nebo externí entita, která interaguje se systémem. Účelem definování aktérů je identifikovat různé entity, které budou interagovat se systémem, a porozumět jejich cílům a záměrům. Tyto informace se pak použijí k vytvoření scénářů případů užití. Dale následuje seznam aktérů pro tuto aplikaci.

- Učitel.

  Oprávněný uživatel s rolí učitel, který může spravovat témata, úkoly a skupiny studentů a vytvářet reporty pro vyhodnocení úkolů.

- Student.

  Oprávněný uživatel s rolí student, který může zobrazovat své výsledky a prohlížet úkoly.

- Administrátor.

  Oprávněný uživatel s rolí administrátor, který má přístup ke všem funkcionalitám systému včetně spravovaní uživatelských účtů.

- Nepřihlášený uživatel.

  Uživatel, který nebyl do aplikace přihlášen. Má přistup na stránku pro přihlášeni.

2.7.2 Případy užití - učitel

V této podsekci jsou popsány případy užití, hlavním aktérem kterých je učitel.

UC1: Zadání a editování úkolů

Popis: Případ užití umožňuje učiteli nadefinovat a uložit úkol do aplikace, což je jednou z jeho nejdůležitějších činností. Také povoluje už vytvořený úkol opravit.

Scénář: Prvním krokem je vybrat téma, do kterého bude patřit úkol, na stránce se seznamem témat. Pak po zobrazení seznamu úkolů pro určité téma by učitel měl začít proces přidání nového úkolu pomocí přechodu do další stránky. Pro definování úkolu nejprve je nutné vybrat jeho typ. Potom následuje zadání názvu, popisu a maximálního počtů bodů za úspěšné splnění úkolů. Dalším krokem je vyplnit povinné a volitelné parametry podle zvoleného typu úkolu. Nakonec učitel úkol zkontroluje a pokusí se ho uložit. Pokud učitel nevyplnil povinný parametr, tak systém by měl zobrazit hlášku, aby učitel zadal chybějící údaje. V případě, že formát zadaných údajů neodpovídá potřebnému formátu, tak by systém měl zobrazit další hlášku pro jejich opravu. Po úspěšném uložení učitel má možnost úkol upravit podle potřeby. Postup pro editování je stejný, tedy učitel vyplní formulář, kde zadá nový název, popis nebo počet bodů a v případě, že potřebuje změnit i typ úkolů, vybere nový a zadá povinné a volitelné parametry. Učitel se upravený úkol pokusí uložit, pak systém provede potřebné kontroly zadaných údajů. V případě úspěchu se změny uloží, v opačném případě systém zobrazí pokyny pro opravu.

UC2: Spravování úkolů

Popis: Případ užití slouží pro umožnění učiteli spravovat úkoly. Učitel by měl mít možnost úkoly přidávat, upravovat a odstraňovat.

Scénář: Pro řízení úkolů by učitel měl nejprve vybrat téma ze seznamu témat a po jeho rozkliknutí přejít na stránku s úkoly, jež do něho patří. Na této stránce učitel má několik možnosti pro činnost. První je přidání a editování úkolu(UC1). Další aktivita je odstranění určitého úkolu z aplikace, což je možné udělat pokud tento úkol nemá přiřazené hodnocení, to znamená, že automatické vyhodnocení úkolu ještě nikdy nebylo spuštěno. V případě, že takové hodnocení neexistuje po potvrzení od učitele úkol bude smazán. Naposled učitel může zobrazit detaily úkolu a zkontrolovat jestli jsou v pořádku.

UC3: Spravování témat

Popis: Případ užití slouží pro umožnění učitelovi spravovat témata. Učitel by měl mít možnost témata přidávat, upravovat a odstraňovat.

Scénář: Případ užití se začíná, když učitel přejde z domovské stránky na stránku se seznamem všech témat. Dale má několik možností pro jejich spravování. První je přidání nového tématu. Učitel otevře stránku, kde bude možné zadat název, popis a stav, jestli je téma aktivní nebo ne, a pak téma uložit do aplikace. Učitel má také možnost téma odstranit. To může udělat v případě, že téma neobsahuje úkoly, které už mají přiřazené hodnocení, a po potvrzení téma bude smazáno z aplikace spolu s úkoly, které má uvnitř. Další aktivita je editování tématu, během kterého učitel má možnost změnit název, popis nebo stav konkrétního tématu a změny pak uložit.

UC4: Kontrola splnění úkolů

Popis: Případ užití umožňuje učitelovi provádět kontrolu splnění úkolů studenty. Pro zobrazení výsledků slouží reporty, během jejichž generování probíhá automatické vyhodnocení. Po generování reportu je možné ho exportovat do souboru pro další potřeby učitele.

Scénář: Případ užití se začíná, když učitel chce zkontrolovat úkoly a zobrazit výsledky studentů. Učitel přejde z domovské stránky do sekce s reporty. Po přechodu se mu zobrazí seznam už vytvořených definic reportů, které je možné otevřít a podívat se na aktuální výsledky studentů. Učitel může už existující definici smazat, pokud už ji nebude potřebovat. Pro vytvoření nové definice reportu, tedy i pro další automatickou kontrolu splnění úkolů, učitel může přejít na stránku pro přidaní reportu. Na této stránce by měl vybrat typ reportu, zadat jeho název a parametry podle typu, tedy zvolit skupiny

a témata, a definici reportu uložit. Pro automatické vyhodnocení úkolů učitel spustí proces generování reportu, během kterého se provede automatická kontrola autorství, ověření splnění úkolů, spočítaní bodů a vytvoří se report, který se pak učiteli zobrazí na další stránce.

UC5: Export reportu

Popis: Případ užití umožňuje učiteli po vygenerování reportu ho exportovat do souboru pro další potřeby.

Scénář: Případ užití se začíná, když učitel chce uložit výsledky studentů na svůj počítač. Po vytvoření nového reportu a po jeho zobrazení na stránce učitel bude mít možnost provést export reportu. Aplikace povoluje učiteli vybrat vhodný typ souboru JSON nebo CSV. Pak učitel spustí proces exportu a jeho výsledek najde ve složce pro stahovaní.

UC6: Vytvoření skupiny studentů

Popis: Případ užiti umožňuje učiteli vytvořit skupinu studentů pomocí zadání tagu, podle kterého budou načteny údaje pro skupinu, tedy projekty z vývojového nástroje.

Scénář: Případ užití se začíná, když učitel potřebuje na začátku semestru přidat do aplikace novou skupinu studentů. Učitel přejde z domovské stránky do stránky pro evidovaní skupin studentů, kde bude mít možnost pro přidání další skupiny. Pro tento účel učitel přejde na stránku, kde může zadat název skupiny, tag, který bude označovat určitou skupinu, například pro vývojový nastroj GitLab. Hodnota tagu by se měla nacházet v popisu projektu. Pak po ověření a potvrzeni uvedených údajů učitelem se podle zadaného tagu skupina načte, co znamená, že se vytvoří projekty v aplikace, které budou obsahovat odkaz na studenta a na projekt v nástroji GitLab. Údaje studentů bude také možné zobrazit v aplikaci.

UC7: Zobrazení údajů studentů

Popis: Případ užití slouží pro zobrazeni studentů a jejich údajů podle skupin. Takovými údaji mohou být uživatelské jméno, e-mailová adresa a link na studenta ve vývojovém nástroji.

Scénář: Prvním krokem pro zjištění učitelem informace o studentech je otevření sekce se skupinami v aplikaci. Dále učitel by měl vybrat semestr a skupinu, ke které patří hledaný student. Pak se zobrazí seznam studentů z této skupiny. Pro zjištění údajů určitého studenta učitel může otevřít stránku, na níž najde více detailní informace.

2.7.3 Případy užití - student

V této podsekci jsou popsány případy užití, hlavním aktérem kterých je student.

UC8: Zobrazení vlastních výsledků

Popis: Případ užití umožňuje studentovi prohlížet vlastní výsledky po splnění úkolů. Student by měl mít možnost se podívat jaké úkoly už splnil a kolik bodů za to dostal.

Scénář: Případ užití se začíná, když student přejde z domovské stránky do sekce z výsledky. Tam mu budou zobrazeny už splněné úkoly a počet bodů, které dostal po jejich kontrole, ve formě seznamu. Student bude moct otevřít detaily úkolu z tohoto seznamu a podívat se na podrobnější hodnocení, tedy kolik bodů a za jaký parametr úkolu dostal.

UC9: Zobrazení úkolů pro splnění

Popis: Případ užití umožňuje studentovi zobrazit seznam úkolů, které by měly být splněny, podle témat. Student by měl mít možnost zobrazit detaily a popis úkolu.

Scénář: Prvním krokem pro zobrazení úkolů, které potřebuje student splnit je přejít z domovské stránky do stránky se seznamem témat. Po přechodu by měl student vybrat téma, po jehož rozkliknutí se dostane na stránku se seznamem úkolů, které potřebuje splnit. Dále má student možnost zobrazit popis a ostatní detaily zadaného úkolu.

2.7.4 Případy užití - administrátor

V této podsekci jsou popsány případy užití, hlavním aktérem kterých je administrátor.

UC10: Spravování uživatelských účtů

Popis: Případ užití umožňuje administrátorovi spravovat uživatelské účty, tedy přidávat nové uživatele, odstraňovat je a upravovat údaje uživatelů.

Scénář: Prvním krokem administrátora pro změnu nebo zobrazení uživatelských účtů je přejít na stránku se seznamem účtů. Dále má několik možností. První je přidat nového uživatele – administrátor se dostane na stránku, kde by měl vyplnit údaje nového uživatele a pak je uložit do aplikace. Admi-

nistrátor má také možnost určitého uživatele odstranit nebo upravit jeho účet.

UC11: Změna role uživatele

Popis: Případ užití slouží pro změnu role uživatele. Administrátor by měl mít možnost přidat nebo odebrat role určitého uživatele.

Scénář: Prvním krokem pro změnu role uživatele je po přihlášeni otevřít seznam uživatelů. Pak administrátor vybere uživatele, u kterého potřebuje provést změnu, a přejde na stránku pro pro editovaní rolí. Potom administrátor vybere role ze seznamu existujících a stiskne tlačítko pro uložení.

2.7.5 Případy užití - nepřihlášený uživatel

V této podsekci jsou popsány případy užití, hlavním aktérem kterých je nepřihlášený uživatel.

UC12: Přihlášení do aplikace

Popis: Případ užití slouží pro autentizaci uživatele v aplikaci, po které uživatel bude mít přistup ke svému účtu a údajům.

Scénář: Případ užití se začíná, když uživatel otevře stránku aplikace pro přihlášení. Uživatel vyplní svoje uživatelské jméno a heslo a zkusí se přihlásit. Po úspěšné kontrole údajů se uživatel dostane na domovskou stránku aplikace. Pokud přihlášení neproběhlo úspěšně, tak by se měla zobrazit hláška, která obsahuje informaci o dalších pokynech, například pokud přihlašovací údaje nebyli správné, tak by je měl uživatel opravit a přihlásit se znovu.

UC13: Resetování hesla

Popis: Případ užití umožňuje uživateli provést resetování hesla pomocí e-mailu, pokud ho zapomněl.

Scénář: Pro nastavení nového hesla by uživatel mel vyplnit svou e-mailovou adresu na stránce pro resetování hesla. Pak dostane e-mail s odkazem na stránku, kde bude možné zadat nové heslo a uložit ho.

2.8 Způsob přihlášení do aplikace

Jelikož aplikace bude sloužit pro potřeby Fakulty informačních technologií je nutné se zamyslet nad vhodným způsobem přihlášení.

Jedna možnost je registrace uživatelů. Po vyplněni registračního formuláře se požadavek dostane k administrátorovi aplikace. Administrátor by měl

potvrdit registraci, a pak by byl vytvořen nový uživatelský účet. Nebo administrátor může ručně přidat nového uživatele do aplikace.

Další možnost je využit to, že uživateli aplikace už by měli mít univerzitní účet. Tedy je možné připojit do aplikace systém, který se využívá pro přihlášení do ostatních univerzitních systémů. To je Single Sign On nebo SSO [13]. „Ten systém má své výhody. Technologie SSO umožňuje uživatelům jediné přihlášení, které jim zpřístupní informační zdroje z více různých systémů bez opětovného požadavku na přihlašování. Po prvotním přihlášení do počítače uživatele další systémy rozpoznají a přihlášení do nich proběhne automaticky bez nutnosti vkládání dalších hesel.“ [14] Pro danou aplikaci využití technologie SSO dává možnost ji zapojit do světa Českého vysokého učení technického v Praze.

V rámci první verze aplikace, které se zabývá tato bakalářská práce, bylo rozhodnuto implementovat možnost přihlášení, pomoci uživatelských účtů, které budou vytvořeny administrátorem, jelikož v této verzi studenti nebudou mít přístup k dané aplikaci.

Kapitola 3

Návrh

V této kapitole je popsán návrh aplikace, který je udělán na základě analýzy. Nejprve v sekci 3.1 je návrh architektury aplikace. Potom následuje sekce 3.2, která se zabývá návrhem uživatelského rozhraní. Důležitou částí pro tvorbu aplikace je také výběr vhodných technologií, který je popsán v sekce 3.3. Pak následuje návrh způsobu zabezpečení aplikace v sekci 3.4.

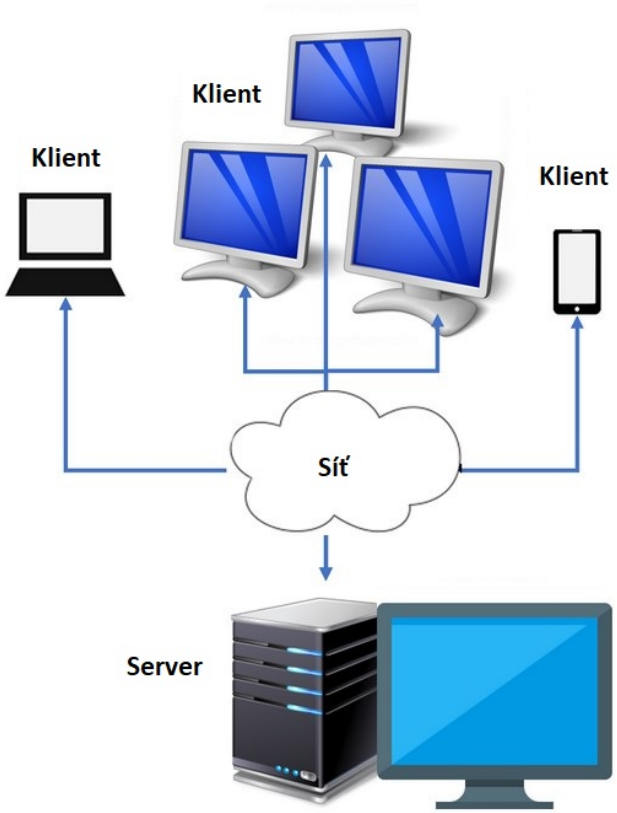
3.1 Návrh architektury

V této sekci je popsán návrh architektury celé aplikace. Z funkčních požadavků a představ vyplývá, že se bude jednat o webovou aplikaci. Webová aplikace je program, který je obvykle uložen na vzdáleném serveru a uživatelé k němu mohou přistupovat pomocí softwaru známého jako webový prohlížeč. Výhodou takového typu aplikace je přístup všemi uživateli ke stejné verzi webové aplikace. To eliminuje všechny problémy s kompatibilitou. Další výhodou je, že obvykle není nutné ji instalovat na pevný disk počítačového systému, a proto odpadají všechny problémy spojené s omezeným prostorem. [15]

3.1.1 Klient-server model

Model klient-server je struktura distribuované aplikace, která rozděluje úlohu mezi poskytovateli zdroje nebo služby, nazývané servery, a žadateli o služby, nazývané klienti [16]. V architektuře klient-server, když klientský počítač odešle požadavek na data serveru přes síť, server to přijme a doručí požadované datové pakety zpět klientovi. Model je znázorněn na obrázku 3.1

Klienty mohou být stolní počítače, notebooky, mobily nebo jiná zařízení, která mohou komunikovat se serverem přes síť a zadávat požadavky na server jménem uživatele. Klientské aplikace jsou obvykle navrženy tak, aby uživatelům poskytovaly uživatelské rozhraní. Servery mohou být aplikace, které běží



Obrázek 3.1: Klient-server model [17]

na počítači nebo clusteru počítačů a jsou navrženy tak, aby zpracovávaly více požadavků od více klientů. Serverové aplikace mohou být navrženy pro správu dat, obsluhu webových stránek nebo například poskytování e-mailových služeb.

Jednou z hlavních výhod tohoto modelu je, že díky oddělení serveru od klienta, při potřebě přidat nový typ klienta, nebude probíhat duplikování kódu serveru, tedy i kapacitu klientů a serverů lze měnit samostatně. Dalším důvodem pro zvolení tohoto modelu je, že umožňuje centralizovanou správu dat, protože data lze ukládat na centrální server a přistupovat k nim více klientů. To umožňuje lepší konzistenci a integritu dat a snižuje riziko jejích ztráty.

3.1.2 Server

V této podsekci je popsána architektura serverové části aplikace.

3.1.2.1 Třívrstvá architektura

Pro tuto aplikaci byla zvolena třívrstvá architektura. To je vzor softwarové architektury. V této architektuře jsou uživatelské rozhraní, aplikační logiku a ukládání dat rozděleny do odlišných vrstev.

Třívrstvá architektura obsahuje tři vrstvy:

1. **Prezentační vrstva:** Prezentační vrstva je uživatelské rozhraní a komunikační vrstva aplikace, pomocí které koncový uživatel komunikuje s aplikací. Jeho hlavním účelem je zobrazovat informace a sbírat informace od uživatele [18].
2. **Aplikační vrstva:** Táto vrstva je zodpovědná za implementaci business logiky aplikace. Je umístěna mezi prezentační a vrstvou pro přístup k datům (např. databáze) a je zodpovědná za zpracování a manipulaci s daty předtím, než jsou prezentována uživateli nebo uložena v databázi[19].
3. **Datová vrstva:** Datová vrstva je vrstva, která je zodpovědná za správu ukládání dat a načítání je do aplikace. Nachází se mezi aplikační vrstvou a systémem ukládání dat a poskytuje abstraktní vrstvu, která umožňuje vrstvě business logiky komunikovat se systémem ukládání dat, aniž by si byla vědoma jeho konkrétní implementace[20].

3.1.3 Klient

V této podsekci je popsána architektura klientské části aplikace.

3.1.3.1 Jednostránková aplikace

Jednostránková aplikace nebo Single Page Application (SPA) je webová aplikace, která má jednu stránku – v prohlížeči se načte jedna stránka HTML a během používání se znovu nenačítá. Když uživatelé interagují se stránkou, aplikace dynamicky přepisuje aktuální stránku místo načítání nové stránky ze serveru.[21]

Existuje i jiný typ webových aplikací, jehož název je vícestránková aplikace nebo Multi-Page Application (MPA). Vždy když aplikace tohoto typu potřebuje zobrazit data nebo odeslat data zpět na server, musí si vyžádat novou stránku, která se následně vykreslí ve webovém prohlížeči [21].

Pro účely této bakalářské práce byl zvolen typ jednostránkové webové aplikace, jelikož tento typ má několik výhod. První je, že jednostránkové aplikace rychle reagují na akce uživatelů. Dalším bodem je eliminace potřeby psaní kódu pro vykreslování stránek na serveru, tedy se proces vývoje stává efektivním a přímočarým [22].

3.2 Návrh uživatelského rozhraní

Před zahájením vývoje aplikace byl vytvořen návrh uživatelského rozhraní aplikace. Výsledkem je několik drátěných modelů obrazovek(wireframes). To jsou schémata, které nastiňují základní strukturu stránky, včetně pozice klíčových prvků, jako jsou nadpisy, obrázky, textové bloky, tlačítka a odkazy. Drátové modely pomáhají plánovat návrh aplikace před napsáním jakéhokoli vizuálního návrhu nebo kódu. To poskytuje jasné pochopení struktury, funkčnosti a obsahu výsledného produktu. Pro jejich tvorbu byl použit nástroj Balsamiq [23]. Návrh se může lišit od výsledného uživatelského rozhraní.

Všechny drátové modely, které byly vytvořeny během návrhu, se nachází v příloze B. Dale následuje jejich popis a příklady některých z nich.

Domovská stránka

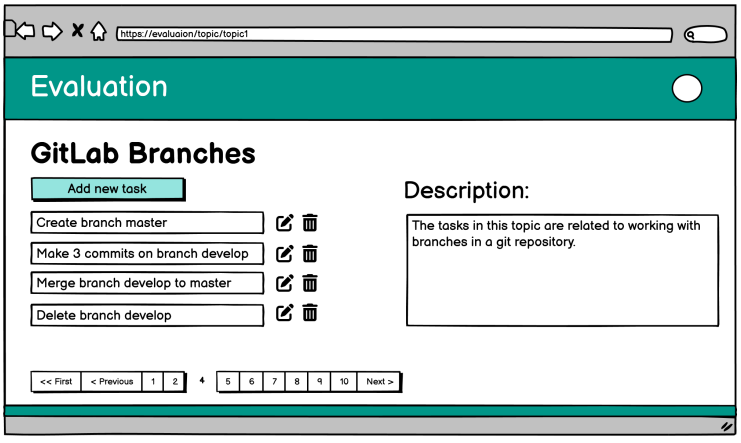
Na tuto stránku B.1 se uživatel dostane po přihlášení. Na stránce jsou tlačítka sloužící pro přechod do různých sekcí aplikace.

Témata

Na stránce B.2 se nachází seznam témat, které je možné editovat a odstraňovat pomocí tlačítek. Kliknutím na určité téma je možné přejít na stránku s informacemi o něm. Na stránce je také tlačítko na přidání nového tématu.

Téma

Na stránce 3.2 je vidět seznam úkolů, které je možné upravovat a odstraňovat pomocí tlačítek, popis tématu a tlačítko pro přidání dalšího úkolu. Kliknutím na určitý úkol je možné přejít na stránku s informacemi o něm.



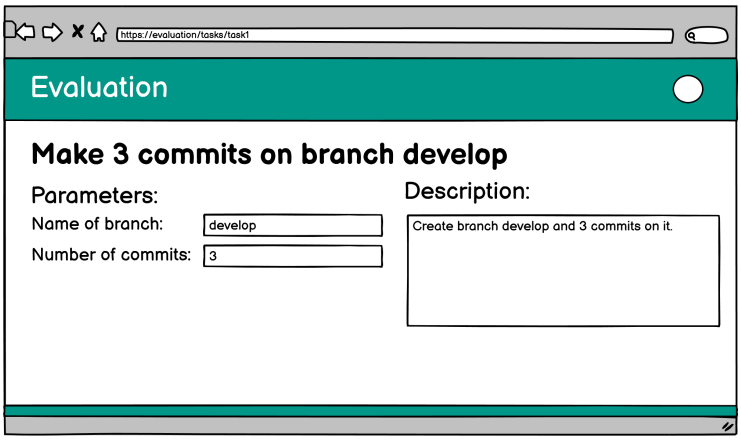
Obrázek 3.2: Téma

Přidání nebo editování tématu

Stránka B.4 slouží pro přidání nebo editování tématu. Na stránce je možné zadat název tématu a jeho popis. Pak je možné uložit ho pomocí tlačítka.

Úkol

Na stránce 3.3 jsou zobrazeny parametry úkolu a jeho popis.



Obrázek 3.3: Úkol

Přidání nebo editování úkolu

Stránka 3.4 slouží pro přidání nebo editování úkolu. Na stránce je možné zadat název úkolu, popis, jeho typ a parametry podle typu a uložit ho pomocí tlačítka.

Reporty

Na stránce 3.5 se nachází seznam uložených reportů a tlačítka pro jejich odstranění a generování nového reportu. S této stránky je možné se dostat na stránku s detaily určitého reportu.

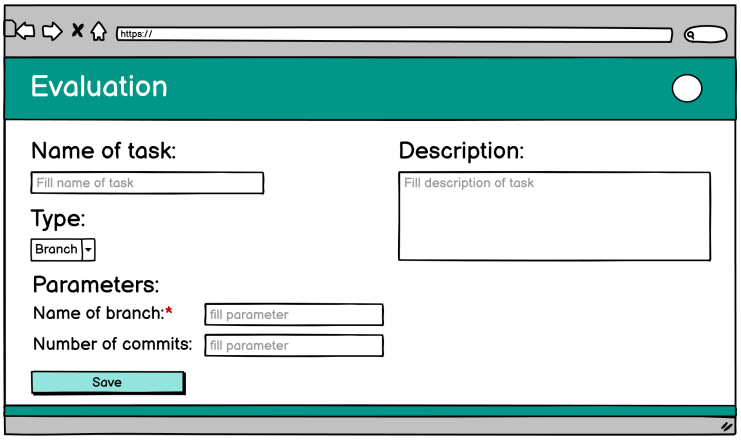
Report

Na stránce B.8 je zobrazen jeden určitý report. Pomocí odpovídajícího tlačítka je možné provést export tohoto reportu.

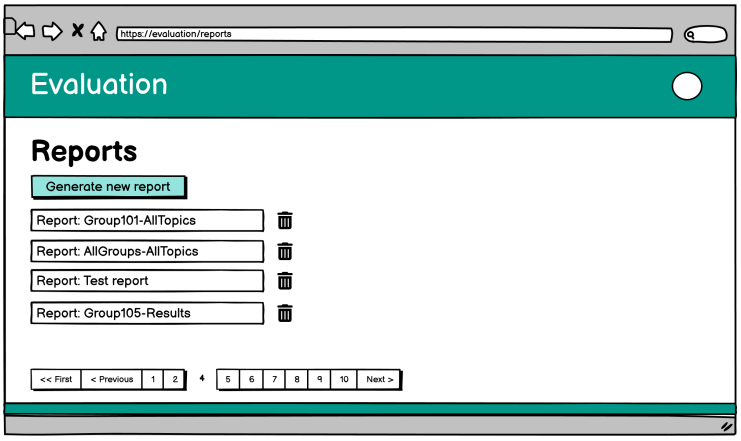
Vytvoření reportu

Stránka B.9 slouží pro vytvoření nového reportu. Je možné zadat název reportu, jeho typ a parametry podle typu.

Přidání skupiny



Obrázek 3.4: Přidání nebo editování úkolu



Obrázek 3.5: Reporty

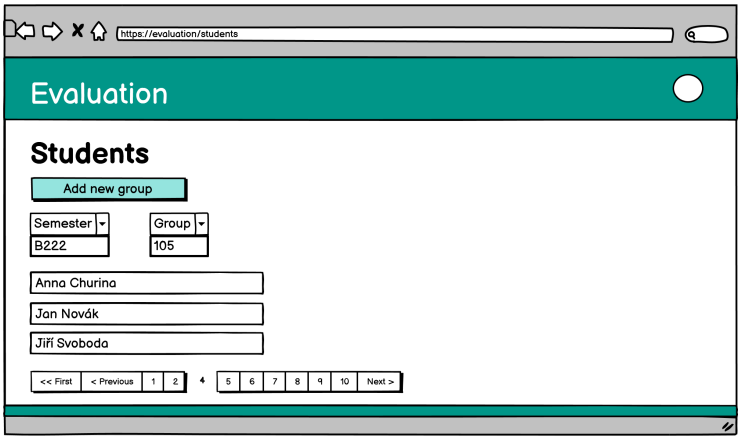
Stránka B.10 slouží pro přidání nové skupiny. Je možné zadat tag, podle kterého bude vytvořená skupina a její jméno.

Studenty

Na stránce 3.6 je možné zobrazit seznam studentů po výběru semestru a skupiny. Je tam také tlačítko pro přidání další skupiny.

Student

Na stránce B.12 jsou zobrazeny údaje studenta.



Obrázek 3.6: Studenty

3.3 Použité technologie

V této sekci jsou popsány technologie, které byly zvoleny pro tvorbu aplikace, jíž se zabývá daná bakalářská práce. Výběr technologií byl zaměřen na webovou aplikaci a vycházel z relevance technologií, ze souladu s požadavky na aplikaci a ze zkušeností autora.

3.3.1 Backend

Backend technologie označují technologie, které se používají k vývoji a správě komponent webové aplikace na straně serveru. Tyto technologie jsou zodpovědné za zpracování požadavků a odpovědí, správu dat a poskytování business logiky a funkčnosti aplikace.

3.3.1.1 Java

Existuje několik programovacích jazyků, které se běžně používají pro vývoj backendu, včetně Java, Python, PHP a Node.js. Pro tuto aplikace byl zvolen programovací jazyk Java vyvinutý společností Sun Micro systems v roce 1995 [24]. Java má jako programovací jazyk řadu výhod, které jsou popsány dále.

Java je objektově orientovaný jazyk. To znamená, že podporuje zapouzdření, dědičnost a polymorfismus. Tento fakt usnadňuje psaní modulárního a opakovaně použitelného kódu, což může zkrátit dobu vývoje a zlepšit kvalitu kódu. Další výhodou je platformní nezávislost, tedy Java aplikace může běžet na jakékoli platformě s nainstalovaným Java Virtual Machine (JVM). Java umožňuje programátorům vytvořit webovou aplikaci, aniž by se museli spoléhat na jednu sadu hardwaru nebo operačních systémů. To šetří čas a ná-

klady na vývoj a vývojáři se mohou soustředit na tvorbu webových aplikací bez ohledu na platformy, pro které je vytvářejí.

3.3.1.2 Spring Boot

Spring Boot [25] je open-source framework založený na Javě, který je navržen tak, aby zjednodušil proces vytváření a nasazování aplikací založených na Springu. Poskytuje řadu funkcí a nástrojů, které usnadňují vytváření samostatných aplikací připravených k produkci. Spring Boot se staví na platformě Spring, který je populárním frameworkem Java pro vytváření aplikací. Spring Boot však poskytuje další funkce, které usnadňují konfiguraci a nasazení aplikací.

Některé z klíčových funkcí Spring Boot jsou:

- Automatická konfigurace: Spring Boot poskytuje sadu výchozích konfigurací, které lze automaticky použít na aplikaci na základě závislostí, které jsou zahrnuty. To eliminuje potřebu, aby vývojáři ručně konfigurovali své aplikace.

- Vestavěný server: Spring Boot zahrnuje vestavěný server, jako je Tomcat nebo Jetty, který usnadňuje nasazení aplikace bez nutnosti samostatného aplikačního serveru.

Z důvodů uvedených výše byl pro vývoj této aplikace zvolen Spring Boot.

3.3.1.3 Gradle

Často pro vývoj softwaru se používají nástroje pro automatizaci sestavení, které slouží pro automatické vytváření spustitelných aplikací ze zdrojového kódu. Proces vytváření zahrnuje kompilaci, propojení a zabalení kódu do užitečné nebo spustitelné formy. Jedněmi z nejpopulárnějších takových nástrojů jsou Gradle a Maven.

Maven je nástroj pro automatizaci sestavování, který se široce používá při vývoji softwaru pro správu a automatizaci procesu sestavování projektů založených na Javě [26]. Gradle je open source nástroj pro automatizaci sestavování, který je založen na konceptu Apache Maven a Apache Ant. Je schopen vytvořit téměř jakýkoli typ softwaru [27]. Gradle používá pro své skripty doménově specifický jazyk (DSL) založený na Groovy, což umožňuje větší flexibilitu a přizpůsobení ve srovnání s konfigurací Maven založenou na XML. Gradle DSL je také stručnější a čitelnější než Maven XML.

Pro účely této bakalářské práce byl zvolen nastroj Gradle z důvodů uvedených výše a ze zkušenosti autora.

3.3.2 Frontend

Frontend technologie jsou technologie, které se používají při vývoji klientské strany webové aplikace, tedy k vytvoření uživatelského rozhraní. Jsou zodpovědní za vizuální a interaktivní aspekty aplikace.

3.3.2.1 Typescript

Existuje několik programovacích jazyků, které se běžně používají pro vývoj webových aplikací. To jsou například HTML, CSS, JavaScript nebo TypeScript. Pro vývoj frontend části této aplikace bude používán programovací jazyk TypeScript.

TypeScript je open source programovací jazyk vyvinutý společností Microsoft, který se kompiluje do JavaScriptu. Ve skutečnosti to je nadmnožina nad JavaScript, tedy veškerý platný kód JavaScript je také platným kódem TypeScript [28]. TypeScript rozšiřuje JavaScript přidáním statického typování, tříd, rozhraní a dalších funkcí. Jednou z klíčových výhod tohoto jazyka je, že pomáhá zachytit chyby v době kompilace spíše než za běhu. To znamená, že chyby lze zachytit ještě před spuštěním kódu a tím se ušetří spousta času a úsilí při ladění.

3.3.2.2 React

Jelikož byl zvolen programovací jazyk TypeScript, je potřeba najít knihovnu nebo framework pro vývoj ho podporující. Jednou z nejpopulárnějších vývojových open source JavaScript knihoven uživatelského rozhraní je React [29], který umožňuje vývojářům vytvářet opakovaně použitelné komponenty uživatelského rozhraní a efektivně spravovat stav svých aplikací.

Použití knihovny React přináší několik výhod v rámci vytváření aplikace, kterou se zabývá táto bakalářská práce. Jednou je, že React, ve srovnání s jinými populárními frontend frameworky, jako je Angular nebo Vue, je mnohem snazší se naučit, co dává možnost novým vývojářům rychle se dostat do tempa. [30] Syntaxe React je poměrně jednoduchá a snadno pochopitelná. Další výhodou je velká a aktivní komunita vývojářů, tedy existuje bohatý ekosystém knihoven a pluginů třetích stran, které rozšiřují funkčnost React.

3.3.3 Databáze

V současné době existuje velký počet řešení, jak data ukládat. Pro danou aplikaci data, které je potřeba mít v databázi, jsou znázorněny v doménovém modelu, který se nachází v sekce 2.5 této bakalářské práce. Pro výběr vhodné databáze byla provedena analýza dvou typů:

1. **Relační databáze:** Relační databáze, také nazývaná Relational Database Management System (RDBMS) nebo databáze SQL, ukládá data

v tabulkách a řádcích, které se také nazývají záznamy [31]. Relační databáze se používají pro aplikace, kde je zásadní konzistence a integrita dat, a mají výhodu v tom, že můžou podporovat velké množství záznamů a uživatelů.

2. **Nerelační databáze:** Nerelační databáze na rozdíl od relační databáze neposkytují žádné tabulky a řádky. Místo toho nerelační databáze používá model úložiště optimalizovaný pro specifické požadavky typu ukládaných dat. To mohou být například grafové databáze, které je založeny na dvou elementech – uzlech (nodes) představujících záznamy a vztazích mezi nimi (edges) [32], nebo dokumentově orientovaná databáze, kde data se ukládají v dokumentech, nejčastěji ve formátu JSON nebo XML [32]. Nerelační databáze se často používají, když je třeba organizovat velké množství složitých a různorodých dat [33].

Jelikož není potřeba v časté změně dat a v ukládaní mnoha složitých dat, pro aplikaci byla zvolena relační databáze, která poskytuje konzistence dat a lehčí přistup k nim.

Z relačních databází byla zvolená databáze PostgreSQL, protože je výkonným objektově relačním databázovým systémem s otevřeným zdrojovým kódem, který využívá a rozšiřuje jazyk SQL v kombinaci s mnoha funkcemi [34].

3.3.4 OpenAPI

Specifikace OpenAPI je jazyk specifikace pro HTTP API, který poskytuje standardizované prostředky k definování API(Application Programming Interface). [35]

Jakmile je specifikace OpenAPI definována, lze ji použít k automatickému generování dokumentace pro API. Tato dokumentace může obsahovat informace o koncových bodech, parametrech, odpovědích a dalších podrobnostech rozhraní. To usnadňuje vývojářům pochopit, jak rozhraní používat. Kromě dokumentace lze specifikaci OpenAPI použít také ke generování kódu pro API v různých programovacích jazycích. To může ušetřit čas a úsilí při vytváření API a zajistit, aby byly konzistentní a vysoce kvalitní.

Obvykle specifikace je napsána ve formátu YAML nebo JSON a obsahuje informace o koncových bodech, parametrech, odpovědích a dalších podrobnostech rozhraní API.

V rámci této bakalářské práce bylo rozhodnuto vytvořit OpenAPI specifikaci ve formátu YAML a použit ji pro dokumentaci a generování kódu pro klienta a server.

3.3.4.1 Docker

Docker je otevřená platforma pro vývoj, odesílání a spouštění aplikací. Docker poskytuje možnost zabalit a spustit aplikaci v izolovaném prostředí zvaném kontejner. Izolace a zabezpečení umožňuje provozovat mnoho kontejnerů současně na daném hostiteli. Kontejnery jsou lehké a obsahují vše potřebné ke spuštění aplikace.[36]

Pro danou aplikaci nástroj Docker bude využit pro spuštění aplikace ve vývojovém, testovacím a produkčním prostředí.

3.4 Zabezpečení

Při návrhu aplikace je důležité se zamyslet nad její zabezpečením. Dále jsou uvedeny způsoby zabezpečení, které byly vybrány pro danou aplikaci.

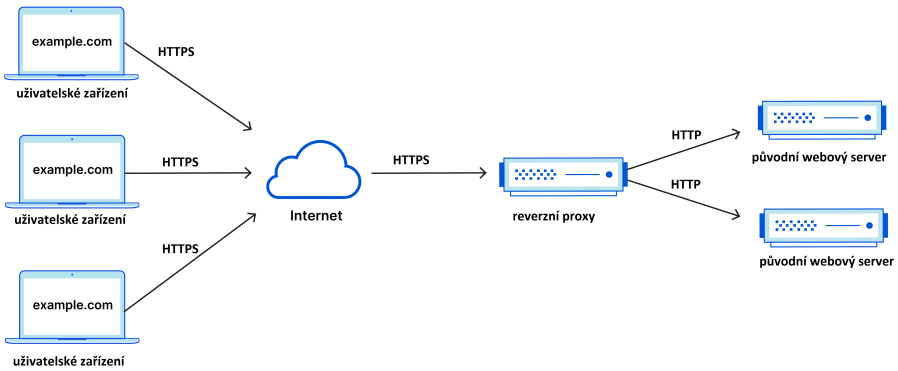
3.4.1 Reverzní proxy a HTTPS

Pro bezpečné posílaní údajů přes Internet je možné využit HTTPS (Hypertext Transfer Protocol Secure) protokol. Pro zvýšení bezpečnosti přenosu dat je protokol HTTPS šifrovaný. To je velmi důležité pro posílání citlivých dat, například hesel uživatelů. Protokol HTTPS používá k šifrování komunikace šifrovací protokol. Tento protokol se nazývá Transport Layer Security (TLS), ačkoli dříve byl znám jako Secure Sockets Layer (SSL). Tento protokol zabezpečuje komunikaci pomocí infrastruktury asymetrických veřejných klíčů. Tento typ bezpečnostního systému používá k šifrování komunikace mezi dvěma stranami dva různé klíče. Jeden se nazývá privátní klíč a druhy je veřejný klíč.[37] Privátní klíč je pod kontrolou vlastníka webové stránky a je uchováván jako soukromý. Tento klíč se nachází na webovém serveru a slouží k dešifrování informací zašifrovaných veřejným klíčem.

Pro konfiguraci HTTPS protokolu na serveru, kde běží aplikace je nutné vygenerovat privátní a veřejní klíč a nastavit použití certifikátu, což je možné udělat pomocí návodu[38].

Pro danou aplikace bylo rozhodnuto navíc použit reverzní proxy. Reverzní proxy je server, který se nachází před jedním nebo více webovými servery a zachycuje požadavky od klientů. U reverzního proxy, když klienti odesílají požadavky na původní webový server, jsou tyto požadavky zachyceny na okraji sítě (network edge) reverzním proxy serverem.[39] Reverzní proxy server pak bude odesílat požadavky a přijímat odpovědi od původního serveru. Pro posílání dat mezi rezervním proxy a webovým serverem bude použit protokol HTTP(HyperText Transfer Protocol), jelikož komunikace bude probíhat uvnitř lokální sítě, a pro posílaní dat mezi klienty a reverzní proxy bude použit protokol HTTPS pro zajištění šifrovaní dat. Obrázek 3.7 to ilustruje.

S reverzním proxy nemusí webový stránka nebo služba nikdy odhalit IP adresu svého původního serveru (serverů). Díky tomu je pro útočníky mnohem



Obrázek 3.7: Reverzní proxy [39]

těžší využít proti nim cílený útok. Místo toho se útočníci budou moci zaměřit pouze na reverzní proxy, která bude mít přísnější zabezpečení a více zdrojů na odražení kybernetického útoku.

Pro první verzi aplikace bylo rozhodnuto nastavení reverzního proxy neimplementovat. Toto by mělo být uděleno v dalších verzích aplikace a pro nasazení do pilotního provozu.

3.4.2 Autentizace a autorizace

Dalším způsobem zabezpečení je autentizace a autorizace uživatelů.

Autentizace je proces ověřování identity uživatele, tedy ověřování jestli osoba, která se snaží získat přístup k systému nebo službě, je skutečně tím, za koho se vydává. [40] Autentizace může být realizována různými způsoby, například pomocí hesla, biometrických údajů (např. otisku prstu) nebo certifikátů.

Autorizace je proces určování, jestli uživatel, který již byl autentizován, má oprávnění k přístupu k určitým zdrojům nebo operacím v aplikaci.[40] Tahle oprávnění mohou být udělena jednotlivým uživatelům, skupinám uživatelů nebo rolím.

V rámci dané aplikace bylo rozhodnuto použit Spring Security pro tyto účely.

Spring Security je část Spring framework, která poskytuje různé bezpečnostní funkce, například autentizace a autorizace, pro vytváření bezpečných aplikací v jazyce Java. Jelikož pro vyroj aplikace byl vybrán Spring Boot, tak Spring Security je vhodnější výběr.

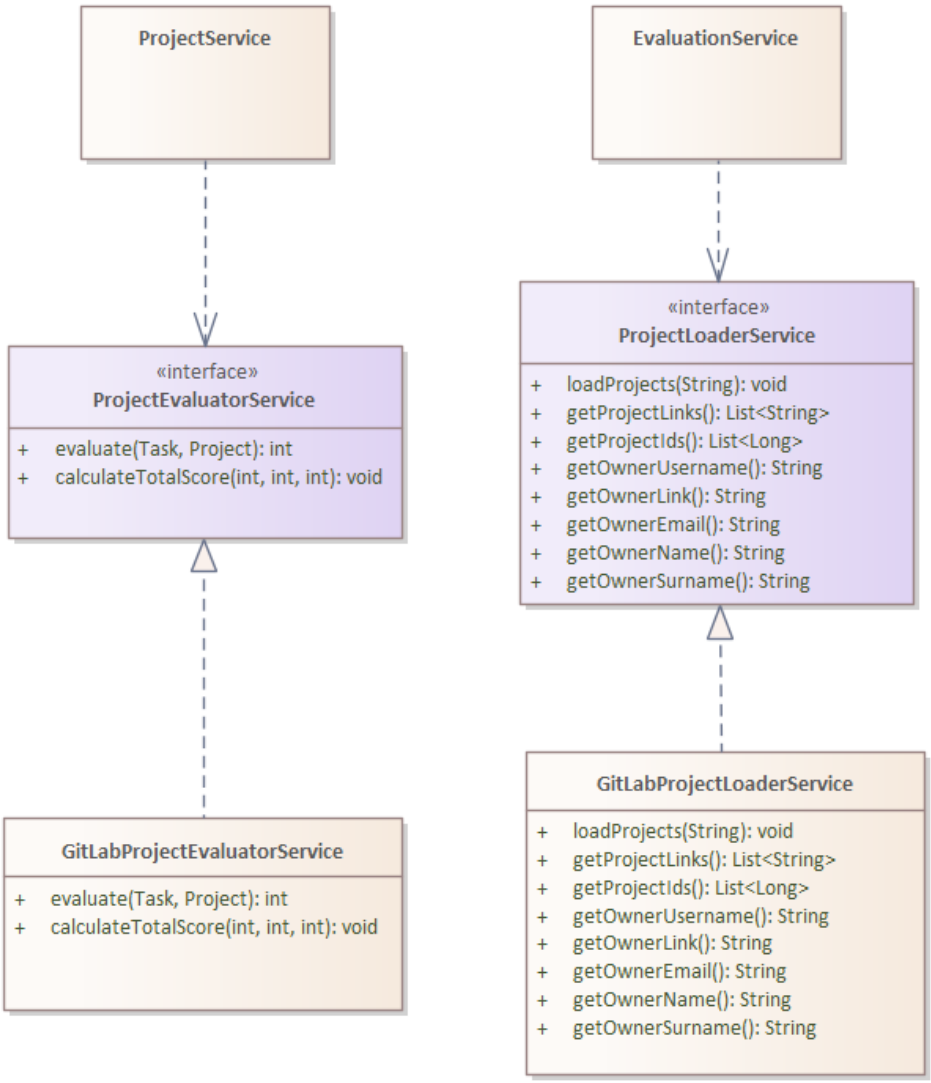
Dále je nutné vybrat autentizační mechanismus. První možnost je použit Basic Authentication. To je jednoduchý mechanismus výzvy a odpovědi, pomocí něhož si server může od klienta vyžádat ověřovací informace (ID uživatele a heslo). Klient předá serveru autentizační informace v hlavičce Authorization. Ověřovací informace jsou v kódování base-64. [41] Nevýhodou tohoto způsobu je zranitelnost vůči útokům typu replay [42], protože základní ověřování nešifruje údaje uživatele. Další možnost je použit JWT(JSON Web Token). JWT je formát pro výměnu informací v podobě tokenu.[43] Takový token je podepsán serverem a obsahuje zakódované informace o uživateli, například jeho uživatelské jméno, role a oprávnění. Tento přístup je bezpečnější než Basic Authentication, protože token je zašifrovaný a útočníci jej nemohou zachytit a změnit. Další výhodou je větší flexibilita, protože token může být uložen v místním úložišti (local storage) prohlížeče nebo v souborech cookie, a proto uživateli můžou zůstat přihlášení v různých relacích.

Pro danou aplikaci byl vybrán způsob s použitím JWT, jelikož má více výhod, nejdůležitější ze kterých je větší zabezpečení.

3.4.3 Napojení dalších vývojových nástrojů

Jelikož hlavním cílem aplikace je vyhodnocovat úkoly, které splnili studenti ve vývojových nástrojích, je nutné navrhnout vhodnou architekturu pro přidání dalších vývojových nástrojů do aplikace.

Bylo rozhodnuto vytvořit dva rozhraní, které budou zodpovídat za načtení skupin, tedy projektu studentů, a vyhodnocení úkolů pomocí dotahování dat z vývojového nástroje po připojení k němu. Přidání nových vývojových nástrojů se bude spočívat v implementaci rozhraní `ProjectEvaluatorService` a `ProjectLoaderService`. Diagram 3.8 to ilustruje.



Obrázek 3.8: Návrh rozhraní pro práci s vývojovými nástroji

Kapitola 4

Implementace

Tato kapitola se zabývá implementací aplikace na základě analýzy a návrhu, které jsou uvedeny v předchozích kapitolách. Kapitola obsahuje seznam využitých vývojových nástrojů v sekci 4.1, popis struktury výsledné aplikace v sekci 4.2, včetně backend a frontend částí. Dále jsou popsány důležité detaily implementace spojené s využitím GitLab API v sekce 4.3, dokumentace v sekci 4.4 a příprava k nasazení v sekci 4.5.

4.1 Použité vývojové nástroje

V této sekci jsou popsány vývojové nástroje, který byli použity pro implementaci aplikace.

4.1.1 GitLab

Během vývoje softwaru pro spravování změn ve zdrojovém kódu a dalších projektových souborech v průběhu času a zároveň udržování historii těchto změn se používají verzovací systémy. Jedním z takových systému je verzovací systém Git.

Pro vývoj teto aplikace byl zvolen nastroj GitLab, který má v sobě řešení pro správu git repozitáře. Jedním z důvodu jeho použití je možnost pro další studenty lehce se zapojit do vývojů, protože FIT ČVUT v Praze poskytuje vlastní verzovací systém v rámci tohoto nástroje.

4.1.2 IntelliJ IDEA

Jako vývojové prostředí bylo zvoleno IntelliJ IDEA od společnosti pro výrobu programového vybavení JetBrains [44], které podporuje programovací jazyky jako Java, Kotlin, Groovy, Scala a další.

IntelliJ IDEA poskytuje programátorům různé nástroje a funkce, které zvyšují produktivitu a usnadňují vývoj software. Mezi tyto funkce patří na-

příklad inteligentní návrhy kódu, refaktoring, automatizované testování a integrace s verzovacími systémy.

4.2 Struktura aplikace

Při vývoji aplikace je důležité mít pochopitelnou a logicky správně rozdělenou strukturu. Pro danou aplikaci struktura vypadá následující:

- `evaluation-server` ..... adresář se zdrojovými kódy backend části
  - `ci` ..... adresář s nastaveními pro GitLab CI/CD
    - `application.properties` . soubor obsahující vlastností pro CI-CD
  - `src` .....
    - `main` ..... zdrojové kódy implementace
      - `api` ..... zdrojové kódy pro kontroléry a výjimky
      - `data` ..... zdrojové kódy pro modely
      - `security` ..... zdrojové kódy pro bezpečnost
      - `service` ..... zdrojové kódy pro business logiku
      - `EvaluationServerApplication.java` ..... třída pro spuštění aplikace
    - `test` ..... zdrojové kódy pro testování
  - `build.gradle` ..... nastavení pro Gradle
  - `settings.gradle` ..... nastavení pro Gradle
  - `Dockerfile` ..... soubor pro vytvoření docker kontejneru pro server
  - `README-SERVER.md` ..... informace o serveru
- `web` ..... adresář se zdrojovými kódy frontend části
  - `public` ..... adresář pro statická data
  - `src` .....
    - `assets` ..... adresář pro styly a obrázky
    - `components` ..... adresář pro react komponenty
    - `generated-sources` ..... adresář pro openapi vygenerovány kód
    - `pages` ..... adresář pro stránky aplikace
    - `utils` ..... adresář pro další funkce
    - `App.tsx` ..... hlavní komponenta aplikace
    - `index.tsx` ..... hlavní vstupní bod aplikace
  - `openapitools.json` ..... nastavení pro openapi generátor
  - `package.json` ..... nastavení pro projekt včetně závislostí
  - `specification.yaml` ..... openapi specifikace
  - `Dockerfile` ..... soubor pro vytvoření docker kontejneru pro klienta
  - `README-CLIENT.md` ..... informace o klientu
- `.gilab-ci.yaml` ..... nastavení GitLab CI
- `docker-compose.yml` ..... soubor pro spuštění celé aplikace přes docker
- `docker-compose.local.yml` ... soubor pro lokální spuštění celé aplikace přes docker
- `README.md` ..... informace o projektu

Aplikace je rozdělena na dva velké moduly – jeden pro serverovou část aplikaci a jeden pro klientskou část. Detaily jejich implementace jsou popsány v dalších sekcích kapitoly.

4.3 Využití GitLab API

Jelikož aplikace, která je vytvářená v rámci této bakalářské práce, se primárně vztahuje k vyhodnocení úkolů spojených s vývojovým nástrojem GitLab, bylo nutné najít způsob se k němu připojit. Pro tyto účely během implementace byly využity dvě knihovny, které poskytují rozhraní k GitLab API, které umožňuje aplikacím Java interagovat s instancemi GitLab programově. To jsou knihovny `java-gitlab-api` [45] a `gitlab4j-api` [46].

Využití `gitlab4j-api`:

1. Připojení k Gitlab.

   Pro připojení k nástrojů Gitlab pomocí této knihovny byl použit osobní přístupový token (personal access token).

   Příklad kódu:

   ```
   private static final String GITLAB_URL =
       "https://gitlab.fit.cvut.cz";

   private final GitLabApi gitLabApi;

   public GitLabProjectLoaderService(User user) {
      this.gitLabApi = new GitLabApi(GITLAB_URL,
          user.getGitLabToken());
   }
   ```

   Ukázka kódu 4.1: Připojení k GitLab API pomocí gitlab4j-api

2. Načtení skupin do aplikace.

   Pro vytváření skupin byla použita metoda, která vrací GitLab projekty podle existence určitého textu v jejich názvu nebo popisu. Pak byla implementována filtrace načtených projektů s využitím masky pro tento text(tag), která je popsala v sekce 2.2. Výsledné projekty obsahují následující údaje – odkaz na projekt v GitLab, jeho identifikační číslo(id) a informace o vlastníkovi, včetně jeho jména, uživatelského jména, odkazu na profil a emailu. Tyto údaje pak byly přidaný do tříd modelů aplikace. Dale je uveden příklad kódu, který to znázorňuje.

```
List<Project> gitLabProjects =
    gitLabApi.getProjectApi().getProjects(findTag);
for (Project project : gitLabProjects) {
   if (project.getDescription() != null &&
   NameValidator.validateWithReg
   (tag,project.getDescription())) {
      projects.add(project);
   }
}
```

Ukázka kódu 4.2: Načtení a filtrování projektů pomocí gitlab4j-api

3. Vyhodnocení úkolů

   Knihovna byla použita pro načtení tiketů a milníků určitého projektu z GitLab podle id projektu, s cílem kontroly existence takových komponent, které splňují požadavky nadefinované v úkolů. Dále jsou příklady z kódu:

```
List<Milestone> milestones;
try {
   milestones = gitLabApi.getMilestonesApi()
      .getMilestones(project.getDevToolId());
} catch (GitLabApiException e) {
   log.warn("exception while working with gitlab", e);
   return Collections.emptyList();
}
```

Ukázka kódu 4.3: Načtení milníků pomocí gitlab4j-api

```
List<Issue> issues;
try {
   issues = gitLabApi.getIssuesApi()
      .getIssues(project.getDevToolId());
} catch (GitLabApiException e) {
   log.warn("exception while working with gitlab", e);
   return Collections.emptyList();
}
```

Ukázka kódu 4.4: Načtení tiketů pomocí gitlab4j-api

Využití `java-gitlab-api`:

1. Připojení k Gitlab.

   Pro připojení k nástrojů Gitlab pomocí této knihovny byl použit stejný způsob, tedy pomocí osobního přístupového tokenu (personal access token).

Příklad kódu:

```
private final GitlabAPI gitlabAPI;

public GitLabProjectEvaluatorService(User user) {
   this.gitlabAPI = GitlabAPI.connect(GITLAB_URL,
       user.getGitLabToken());
}
```

Ukázka kódu 4.5: Připojení k GitLab API pomocí java-gitlab-api

2. Vyhodnocení úkolů

   Pro načtení větví a commitů určitého projektu z GitLab podle id projektu nebo ještě názvu větve(v případě práce z commity) byla použita táto knihovna, protože nabízí vyhovující metody na to. Po načteni se provádí kontrola těchto komponent. Dále jsou příklady kódu:

```
List<GitlabBranch> branches;
try {
   branches = gitlabAPI.getBranches(project.getDevToolId());
} catch (Exception e) {
   log.warn("exception while working with gitlab", e);
   return Collections.emptyList();
}
```

Ukázka kódu 4.6: Načtení větví pomocí java-gitlab-api

```
Map<String,GitlabCommit> commits = new HashMap<>();
try {
   if ("".equals(branch)) {
      List<GitlabBranch> branches =
          getProjectBranches(project);
      for (GitlabBranch gitlabBranch : branches) {
         List<GitlabCommit> gitlabCommits = gitlabAPI
            .getAllCommits
               (project.getDevToolId(),
                   gitlabBranch.getName());
         for (GitlabCommit commit : gitlabCommits) {
            commits.put(commit.getId(), commit);
         }
      }
   } else {
      List<GitlabCommit> gitlabCommits = gitlabAPI
         .getAllCommits(project.getDevToolId(), branch);
      for (GitlabCommit commit : gitlabCommits) {
         commits.put(commit.getId(), commit);
      }
   }
```

```
        } catch (Exception e) {
            log.warn("exception while working with gitlab", e);
            return Collections.emptyMap();
        }
```

Ukázka kódu 4.7: Načtení commitů podle větve pomocí java-gitlab-api

Hlavním důvodem využití obou knihoven bylo poskytování jimi různých metod, které využívají GitLab API. Na začátku implementace byla použitá jenom knihovna `gitlab4j-api`, která dává možnost využívat více způsobů dotazování na GitLab. Například knihovna poskytuje metodu pro načtení tiketů, patřících do určitého milníku, jenom pomoci id projektu a id milníku, což druhá knihovna nemá a což je potřebné pro implementaci vyhodnocení úkolu spojených s milníky. Ale knihovna `gitlab4j-api` na rozdíl od knihovny `java-gitlab-api` neposkytuje možnost rychle dostat větve a commity do nich patřící, která je důležitá při vyhodnocování úkolů spojených s větvemi git repozitáře.

V budoucích verzích aplikace v případě potřeby je možné udělat svou implementaci volání GitLab API místo použití knihoven. To by mohlo byt užitečné, jelikož bude možné definovat metody přímo podle požadavků na vyhodnocení specifických úkolů.

4.4 Dokumentace

Pro dokumentaci aplikace byli využity komentáře ve javadoc formátu. Dale je uveden příklad:

```
/** Method for loading students and their projects
 *  to create new group
 *
 * @param devToolName name of development tool from which projects
 *                  will be loaded
 * @param group sample group without projects in it,
 *                   but with name, tag and semester
 * @return new group which was loaded
 * @throws IllegalStateException
 */
@Override
public Group load(String devToolName, Group group) throws
    IllegalStateException {
   . . .
   return group
}
```

Ukázka kódu 4.8: Komentář v javadoc formátu

REST(Representational state transfer) rozhraní aplikace bylo zdokumentováno pomocí Open API specifikace. Dokumentace ve formátu JSON je výsledkem volání endpointu „hostname/api-docs“.

Návod pro spuštění aplikace je uveden v souborech README.md pro celou aplikaci, README-SERVER.md pro backend část aplikace a README-CLIENT.md pro frontend část aplikace.

4.5 Příprava k nasazení

V rámci přípravy k nasazení aplikace do pilotního provozu bylo provedeno nasazení do testovacího provozu na virtuální stroj poskytnutý FIT ČVUT v Praze na platformě CloudFIT. Dale je popsán proces nastavení a nasazení aplikace na server.

4.5.1 Využití nástroje Docker

Pro aplikaci podle návrhu v sekce 3.3.4.1 bylo rozhodnuto použít kontejnery Docker pro její spuštění. Byly vytvořeny Dockerfile pro server a Dockerfile pro klienta. Tyto soubory obsahují řadu příkazů používaných k vytvoření Docker image. Docker image je samostatný a spustitelný balíček, který obsahuje vše potřebné ke spuštění aplikace, včetně kódu, závislostí, knihoven a konfigurací. Pro spuštění několika kontejnerů spolu se používá nástroj Docker Compose.

Pro danou aplikaci byly vytvořeny soubory `docker-compose.local.yml` a `docker-compose.yml` pro spuštění spuštění serveru, klienta a databázi Postgres. Soubor `docker-compose.local.yml` slouží k lokálnímu spouštění a testování, kde se používají Docker image přímo sestavené z aktuálních zdrojových kódů. Soubor `docker-compose.yml` slouží pro nasazení image, které se pomocí GitLab CI nahrávají do Container Registry.

4.5.2 Nastavení virtuálního stroje

Jako první krok na virtuálním stroji byla provedena instalace Docker, jelikož aplikace by měla být spuštěná pomocí nástroje Docker Compose.

Dalším krokem bylo vytvoření nového uživatele `ci-deploy` a přidaní ho do skupiny `docker`, aby měl možnost Docker spouštět. Do repozitáře tohoto uživatele byl přidán `docker-compose.yml` a skript, který načte nové Docker image a znovu spustí běžící kontejnery. Také bylo provedeno přihlášení do GitLab Container Registry pomocí přístupového tokenu. Dalším krokem bylo vygenerovat SSH klíč, který pak byl využit pro vzdálené spuštění skriptu během úloh GitLab CI/CD, a vložit privátní klíč do `home` repozitáře do `/.ssh/authorized_keys`.

4.5.3 Nastavení GitLab CI

Pro automatické nasazování aplikace na server bylo využito Countinuous Integration / Continuous Deployment pomocí služby GitLab CI. V tomto repozitáři byl vytvořen soubor `.gitlab-ci.yml`, který obsahuje potřebné konfigurace pro pipeline, která umožňuje vývojářům definovat a spouštět řadu kroků spojených s vývojem, testováním a nasazením aplikace, které se spouštějí automaticky po každém příkazu `git push` s commitem do GitLab repozitáře nebo jiné události dle konfigurace.

Pipeline pro tento projekt má 4 fáze:

1. **build-local**

   V této fázi se spustí dvě úlohy. První se jmenuje *Build server (local)* a provede sestavení serveru. Výsledkem je artefakt, který ma v sobě spustitelné *.jar soubory. Druhá úloha je *Build client (local)* a provede sestavení klienta, balíček výsledků kterého je také artefaktem. Tato fáze ověřuje, že lze obě části aplikace v pořádku sestavit.

2. **test**

   Táto fáze má jednu úlohu *Run tests*, která provede automatické spuštění Unit testů serverové častí aplikace.

3. **build-docker**

   V rámci této fázi se spustí dvě úlohy: *Build server (Docker)* a *Build client (Docker)*. Během těchto úloh se provede sestavení Docker image pro server a pro klienta a jejich nahrání do GitLab Container Registry, kde pak budou uloženy. Každý Docker image má unikátní tag spojený s krátkou (zkrácená) verzí identifikátoru commitů(commit hash) v systému Git, například `sha-028d96d0-client`.Navíc se přidávají image s tagem podle větve. Před nahráním se provede přihlášení do Container Registry.

4. **deploy**

   Táto fáze má dvě úlohy: *Deploy dev (server)* a *Deploy dev (client)*. Tyto úlohy jsou manuální, tedy se nespustí automaticky. Během jejich spuštění se provede přidaní aktualních Docker image do Container Registry s tagem `deploy-dev-server` nebo `deploy-dev-client`. Pak pomocí SSH protokolu se spustí skript na virtuálním stroji, který načte nové Docker image a znovu spustí kontejnery pomocí `docker-compose.yml souboru`, který se nachází na virtuálním stroji.

Kapitola 5

Testování

V této kapitole je popsán průběh testování aplikace a jeho výsledky. Nejdřív jsou popsaný jednotkové testy v sekce 5.1. Sekce 5.2 se zabývá uživatelským testováním.

5.1 Jednotkové testy

Testování kódu je klíčové pro zajištění kvality softwarových aplikací a snížení rizika chyb, které mohou vést k neočekávanému chování aplikace. Testování umožňuje vývojářům ověřit, zda kód funguje podle očekávání, a zajistit, že nové změny neovlivní již existující funkcionalitu. Testování kódu také pomáhá odhalovat a opravovat chyby co nejdříve během vývojového procesu. Testování kódu může mít různé formy, jako jsou například jednotkové testy, integrační testy nebo testy uživatelského rozhraní. V této sekci je kladen důraz na jednotkové testy(unit tests).

Jednotkové testování je proces vývoje softwaru, ve kterém jsou nejmenší části aplikace, nazývané jednotky, individuálně a nezávisle kontrolovány, jestli správně fungují.

V rámci této bakalářské práce byli napsány jednotkové testy pro třídy a metody, které se zabývá načtením údajů z vývojového nástroje GitLab a vyhodnocením úkolů, což je primárním cílem aplikace.

Pro napsaní testů byla použitá knihovna JUnit Jupiter Params, která dává možnost používat parametrizované testy. Tedy je možné napsat jeden test a předávat do něho různé parametry, například vstupy a očekávané výstupy metod. Dále v rámci ukázky kódu 5.1 je uveden jednotkový parametrizovaný test pro ilustraci.

Také pro testování metod, které potřebují připojení do GitLab, bylo použito mockování. To je metoda testování softwarových aplikací, která umožňuje vývojářům simulovat určitou funkčnost nebo chování systému nebo jeho částí.

```
@ParameterizedTest
@MethodSource("provideNames")
void testNameValidation (String value, String elementName,
    boolean result){
   Assertions.assertEquals(result,
       NameValidator.validateWithReg(value, elementName));
}

private static Stream<Arguments> provideNames() {
   return Stream.of(
   Arguments.of("name","name", true),
   Arguments.of("*name", "name", true),
   . . .
   Arguments.of("*name*", "AbcAbc", false)
   );
}
```

Ukázka kódu 5.1: Parametrizovaný test

To umožňuje testovat aplikaci izolovaně od dalších součástí systému, které nejsou připraveny nebo nejsou dostupné pro testování.

Jako Mock byli označeny následující třídy: GitLabApi, IssuesApi, MilestonesApi, GitlabAPI, protože jejich metody vyžadují připojení k GitLab, ale to se nepředpokládá v rámci jednotkového testování. Pak v testech je možné zadefinovat vystup určité metody Mock třídy. Pro ilustraci je dále uveden příklad využití mockování v existujících testech:

```
@Mock
private GitLabApi mockGitLabApi;
@Mock
private IssuesApi issuesApi;
. . .

@ParameterizedTest
@MethodSource ("provideIssues")
void testEvaluationOfTicketTask (List<Issue> issues, int
    expectedScore) throws GitLabApiException {
   // settings of parameters
   . . .

   Mockito.when(mockGitLabApi.getIssuesApi())
   .thenReturn(issuesApi);
   Mockito.when(mockGitLabApi.getIssuesApi()
   .getIssues(project.getDevToolId()))
   .thenReturn(issues);
```

```
        int result = gitLabProjectEvaluatorService.evaluate(task,
            project, user);

        Assertions.assertEquals(expectedScore, result);
    }
```

Ukázka kódu 5.2: Využití mockování pro GitLabApi a IssuesApi

V rámci první verze aplikace byli napsány jednotkové testy, které pokrývají nejdůležitější logiku. Tedy bylo otestováno načtení skupin z vývojového nástroje GitLab a vyhodnoceni úkolů, včetně použití masky pro porovnání textu, která je popsaná v sekce 2.2 kapitoly Analýza. V následujících verzích rozsah pokrytí kódu testy je možné zvětšit pomocí testování další logiky, například testování vytváření témat nebo úkolů.

5.2 Uživatelské testování

Uživatelské testování je metoda testování softwarových aplikací, která se zaměřuje na ověřování, jestli aplikace splňuje potřeby a očekávání uživatelů. Cílem uživatelského testování je získat zpětnou vazbu od reálných uživatelů aplikace, aby bylo možné identifikovat případné nedostatky nebo problémy v uživatelském rozhraní, použitelnosti nebo celkovém designu aplikace. Během uživatelského testování obvykle se používají scénáře, které popisují, jaké konkrétní úkoly nebo činnosti budou uživatelé provádět při testování. Scénář se obvykle skládá z několika kroků, které uživatel musí provést, aby úkol dokončil a je vytvořen tak, aby co nejvíce simuloval skutečné situace, které uživatelé budou při používání aplikace zažívat.

Pro uživatelské testování byli zvoleny 2 studenta FIT ČVUT v Praze a jeden známý autora práce, který nemá svůj obyčejný život spojený s informačními technologiemi. Volba byla založena na tom faktu, že studenty fakulty by mohli navrhnout věcí ke změně z pohledu funkcionality a možných rozšíření spojených s tím a třetí tester by mohl otestovat srozumitelnost a vzhled uživatelského rozhraní.

Uživatelské testování se proběhlo podle následujících scénářů:

1. **Vyhodnocení úkolů**

   To je komplexní scénář, podle kterého je možné nasimulovat celý proces, který by mohl provést učitel pro vyhodnocení splnění úkolů studenty. To znamená nejdřív vytvoření tématu a úkolů, pak načtení skupiny studentů s jejich projekty, vytvoření definice reportu a jeho generování, po kterém by se měli zobrazit výsledky studentů.

   Prvním krokem scénáře je se přihlásit jako uživatel s role učitel.

   Pak uživatel by měl přejít na stránku Topics, na které by měl uvidět seznam už existujících témat a tlačítka pro jejich editování a odstranění.

Kliknutím na tlačítko Add new topic uživatel by měl být přesměrován na stránku, kde se nachází formulář pro zadání informace o novém tématu, pak po zadání údajů a jejich uložení by se měla zobrazit stránka s informacemi o vytvořeném tématu. Na této stránce jsou tlačítka pro vracení na stránku Topics a pro jeho editování. Uživatel by mohl je vyzkoušet. Pak uživatel by měl přidat nový úkol do tématu pomocí tlačítka Add new task. Po kliknutí se zobrazí stránka s formulářem, kde uživatel by mohl vyplnit údaje o úkolů podle vybraného typu. Na této stránce se také nachází příručka, kterou uživatel by měl přečíst. Pak kliknutím na tlačítko Create task uživatel by měl úkol vytvořit a tím přejít na stránku s informacemi o novém úkolů, kde by měl uvidět tlačítka pro editování úkolů a pro přechod zpátky do tématu a je vyzkoušet.

Dalším krokem je vytvořit novou skupinu. Uživatel by měl přejít na stránku Groups a vybrat tam vyhovující mu semestr. Pak by měl uvidět seznam už existujících skupin s možnosti je opravit nebo odstranit, co by také měl vyzkoušet. Pak uživatel klikne na tlačítko Add new group, pomocí kterého přejde na stránku s formulářem pro vytvoření nové skupiny. Uživatel by měl zadat jméno skupiny, tag, podle kterého se skupina načte, a vybrat vývojový nástroj. Pak kliknutím na tlačítko Create group, se vytvoří skupina a uživatel bude přesměrován na stánku s její údaje, také na stránce jsou tlačítka pro vracení na stránku Groups a pro editování skupiny, které by uživatel mohl vyzkoušet.

Následující krok je vytvoření definice reportu. Pro tento účel uživatel by měl přejít na stránku Reports, kde se nachází seznam už existujících reportů s tlačítky pro editovaní, odstranění a generování. Uživatel může to vyzkoušet. Pak by měl přidat nový report pomocí tlačítka Add new report. Na stránce pro přidaní reportu uživatel by měl vyplnit potřebné údaje podle vybraného typu reportu a stisknout tlačítko Create report. Pak bude přesměrován na stránku s informacemi o reportu. Pak uživatel by měl stisknout tlačítko Generate report, které ho přesměruje na další stránku, kde budou zobrazený výsledky studentů, což je hlavním cílem tohoto scénáře.

2. **Zobrazení témat a úkolů pro studenta**

   Tento scénář slouží pro ověření očekávaného zobrazení témat a úkolů pro uživatele, který má jenom roli student.

   První krok po přihlášení je otevřít stránku Topics. Pak by se měl zobrazit seznam témat bez tlačítek pro jejich editování, přidání a odstranění. Pak kliknutím na řádek s tématem uživatel by se měl dostat na stránku, kde se zobrazí informace o tématu a seznam úkolů bez tlačítek pro jejich přidání, editování nebo odstranění. Další krok je kliknutí na řádek s úkolem, po kterém se otevře stránka s informacemi o úkolu bez tlačítek pro jeho editování nebo odstranění.

3. **Přihlášení a odhlášení**

   Tento scénář slouží pro otestovaní funkčností přihlašovaní a odhlasovaní z aplikace.

   Nejprve se pokusit přihlásit s chybnými údaji. Měla by se zobrazit chybová hláška a uživatel by zůstal nepřihlášený. Pak uvést správné údaje pro přihlášení jako student. Měla by se zobrazit domovská stránka s odkazy na Topics, Profile a FAQs. Další krok je odhlášení pomocí tlačítka Log out. Po odhlášení by se měla znovu zobrazit stránka pro přihlášení. Pak uživatel by se měl pokusit přihlásit jako učitel nebo administrátor. Po přihlášení by se měla zobrazit domovská stránka odkazy na Reports, Topics, Groups, Profile a FAQs.

4. **Zobrazení a editování profilu**

   Tento scénář slouží pro ověření správného chovaní aplikace pro zobrazení uživatelských údajů a jejich editování. Scénář byl vyzkoušen pod několika účty: pro studenta, učitele a administrátora.

   Nejprve po přihlášení uživatel by měl otevřít svůj profil. Pak by měl být přesměrovány na stránku Profile, kde by měl vidět svoje jméno a příjmení, uživatelské jméno, email a role. Pak uživatel by měl otestovat přidání GitLab tokenu do aplikace a editování svého profilu, během kterého by měl vyzkoušet zadat nesprávné údaje, například emailovou adresu bez @, a ověřit jejich uložení pomocí přechodu zpět na Profile stránku.

Během uživatelského testování byly odhaleny překlepy v názvech a několik nefungujících tlačítek, což bylo pak opraveno. Podle komentářů testerů bylo změněno písmo textu pro lepší přehlednost. Celkový dojem testeru od aplikace byl kladný.

Kapitola 6

Možnosti dalšího rozvoje

V této kapitole jsou popsaný možnosti dalšího rozvoje aplikace, tedy způsoby zlepšení existující verze aplikace a přidání nových funkcionalit nebo rozšíření.

6.1 Přidání dalších typů úkolů

Jednou možností rozšíření aplikace je přidání implementace automatického vyhodnocení pro typy úkolů, které jsou popsané v sekce 2.1.2 v rámci kapitoly Analýza. Kromě toho je možné nadefinovat nové typy úkolu podle potřeby, která může vzniknout během využívání aplikace.

6.2 Přidání dalších reportů

Přidání implementace generování a zobrazení dalších typů reportu by mohlo být ještě jedním způsobem zlepšení aplikace. Je možné vytvořit implementaci pro typy reportů, které jsou nadefinovaný v sekce 2.3 kapitoly Analýza nebo nadefinovat nový typ a udělat jeho podporu.

6.3 Přidání podpory dalších vývojových nástrojů

Aplikace je navržena takovým způsobem, že přidaní dalšího vývojového nástroje by nemělo být problémem. V takovém případě by bylo nutné přidat název vývojového nástroje do aplikace a implementaci načtení skupin projektů studentů a vyhodnocení úkolů spojených s tímto nástrojem. Příkladem dalších vývojových nástrojů by mohli být nástroje GitHub nebo SonarQube.

6.4 Přidání přihlášení pomoci SSO

Pro více pohodlné použití aplikace na fakultě by bylo možné přidat přihlášení uživatelů přes fakultní Single Sign On systém. To by odstranilo nutnost vytváření účtů pro každého uživatele.

6.5 Kontrola autorství

Dalším rozšířením je přidání kontroly autorství během automatického vyhodnocení úkolů. To by umožnilo udělat hodnocení studentů více poctivějším a správnějším.

6.6 Historie reportů

Jednou z možností rozšíření je přidaní podpory pro ukládání už vygenerovaných reportů, tedy vytvoření jejich historie. To by mohlo být užitečným pro zobrazení různých statistik splnění úkolů studenty nebo z časových důvodu, například aby učitel nečekal na vygenerovaní reportu a otevřel už hotový.

6.7 Export reportů

Další možná užitečná je export vygenerovaného reportu do souboru na počítači učitele. Takovým způsobem učitel by mohl uložit údaje o výsledcích studentů, a pak je použit například pro importování do jiné aplikace nebo pro tisk pro studenty.

6.8 Evidování skupin, témat a reportů podle učitele

V následující verzi aplikace by se mohlo změnit zobrazení témat, skupin a reportů pro učitele. Tedy pro určitého učitele by se měli zobrazovat jenom témata, skupiny a reporty, které byly jim vytvořeny. To by zkrátilo čas na hledání mezi všemi tématy, skupinami a reporty.

6.9 Podpora aktivních témat

Další možnost rozvoje je využití atributu isActive u témat, který označuje jestli je téma aktivní nebo ne. To by mohlo být užitečné pro filtrování témat během vytváření definic reportu nebo pro zobrazení studentům jenom potřebných témat.

6.10 Administrátorská sekce

V následující verzi aplikace by mohla být přidaná do uživatelského rozhraní administrátorská sekce. K této sekce by měl mít přistup jenom uživatel, který ma roli administrátor. V ní by bylo možné spravovat uživatelské účty, včetně přidání a odebrání rolí.

6.11 Studentská sekce

Studentská sekce v uživatelském rozhraní je další možnost rozšíření aplikace. V této sekce studenti by mohli vytvářet reporty pro kontrolu splnění úkolů a zobrazení svých výsledků.

6.12 Validace parametrů úkolů podle typu hodnoty

Jedním ze způsobů zlepšení aplikace je přidání validace parametrů úkolů podle jejich typu v klientské části aplikace. To by povolilo se vyhnout možným chybám, které jsou spojeny s formátem dat. Také tahle změna by udělala uživatelské rozhraní pohodlnějším, například pro zadaní data vybrat ho z kalendáře je většinou lepší než psát to ručně.

Závěr

Cílem této bakalářské práce bylo navrhnout a vytvořit implementaci aplikace, která umožní automatické vyhodnocení splnění definovaných úkolů v nástrojích pro podporu tvorby softwarových projektů.

Nejprve na základě cíle byla provedena analýza. V její rámci byl prozkoumán průběh předmětu Úvod do DevOps a proběhly konzultace s vyučujícími předmětu. Výsledkem analýzy je určení typů úkolů, které by měla aplikace podporovat, způsob jejich definovaní v aplikaci a způsob jejich automatického vyhodnocení, včetně zobrazení výsledků. Také byl sestaven doménový model, zadefinovány funkční a nefunkční požadavky a popsány případy užití.

Poté byl proveden návrh aplikace, který zahrnoval volbu architektury a vhodných technologií a návrh uživatelského rozhraní a způsobu zabezpečení. V práci je popsán proces implementace aplikace, její dokumentace, příprava pro nasazení do pilotního provozu a průběh testování.

Výsledná aplikace umožňuje vytvořit téma, v rámci kterého je možné nadefinovat sadů úkolů vztažených ke stavu git repositáře. Aplikace podporuje načtení skupin studentů podle popisu jejich projektů z vývojového nástroje GitLab. Úkoly pak mohou být automaticky vyhodnoceny pro skupiny podle tématu. Pro zobrazení výsledků slouží reporty, které aplikace povoluje nadefinovat a vygenerovat.

Všechny cíle stanovené v bakalářské práci byly splněny a vytvořená aplikace může být použitá pro výuku předmětů na FIT ČVUT v Praze.

Literatura

[1] *FIT CTU Courses, BI-IDO* [online]. January 2023 [cit. 23.01.2023]. Dostupné z: https://courses.fit.cvut.cz/BI-IDO

[2] *DevOps, Wikipedia* [online]. January 2023 [cit. 23.01.2023]. Dostupné z: https://en.wikipedia.org/wiki/DevOps

[3] *GitLab About* [online]. January 2023 [cit. 23.01.2023]. Dostupné z: https://about.gitlab.com

[4] *SonarQube* [online]. January 2008 [cit. 23.01.2023]. Dostupné z: https://www.sonarsource.com/products/sonarqube

[5] *FIT CTU Courses, BI-GIT* [online]. January 2023 [cit. 23.01.2023]. Dostupné z: https://courses.fit.cvut.cz/BI-GIT

[6] *FIT klasifikace* [online]. January 2023 [cit. 23.01.2023]. Dostupné z: https://grades.fit.cvut.cz

[7] *FIT CTU Courses, BI-PA1* [online]. February 2023 [cit. 12.02.2023]. Dostupné z: https://courses.fit.cvut.cz/BI-PA1

[8] *FIT CTU Courses, BI-PA2* [online]. February 2023 [cit. 12.02.2023]. Dostupné z: https://courses.fit.cvut.cz/BI-PA2

[9] *C++, Wikipedia* [online]. January 2023 [cit. 23.01.2023]. Dostupné z: https://en.wikipedia.org/wiki/C%2B%2B

[10] *UML modeling tools for Business, Software, Systems and Architecture* [online]. January 2003 [cit. 23.01.2023]. Dostupné z: https://sparxsystems.com/enterprise_architect_user_guide/15.2/model_domains/create_a_domain_model.html

[11] *Visual Paradigm, Requirement Analysis Techniques* [online]. January 2022 [cit. 23.01.2023]. Dostupné z: https://www.visual-paradigm.com/guide/requirements-gathering/requirement-analysis-techniques

[12] *Functional vs. Non-Functional Requirements: Why Are Both Important?, Mobile and web app development company — Uptech* [online]. January 2016 [cit. 23.01.2023]. Dostupné z: https://www.uptech.team/blog/functional-vs-non-functional-requirements

[13] *Single Sign On, IST, České vysoké učení technické v Praze* [online]. March 2019 [cit. 23.01.2023]. Dostupné z: https://ist.cvut.cz/nase-sluzby/single-sign-on

[14] *Zjednodušte si přihlašování do aplikací pomocí Single Sign-On (SSO), TZB-info* [online]. February 2023 [cit. 23.01.2023]. Dostupné z: https://www.tzb-info.cz/facility-management/128377-zjednoduste-si-prihlasovani-do-aplikaci-pomoci-single-sign-on-sso

[15] *What is Web Application - Javatpoint* [online]. February 2011 [cit. 18.02.2023]. Dostupné z: https://www.javatpoint.com/web-application

[16] *Client-Server Model, GeeksforGeeks* [online]. December 2022 [cit. 18.02.2023]. Dostupné z: https://www.geeksforgeeks.org/client-server-model

[17] *Client-Server Architecture, Itelon* [online]. February 2023 [cit. 18.02.2023]. Dostupné z: https://itelon.ru/blog/arkhitektura-klient-server

[18] *What is Three-Tier Architecture, IBM* [online]. January 2023 [cit. 18.02.2023]. Dostupné z: https://www.ibm.com/topics/three-tier-architecture

[19] *Business-Logic Layer, A computer science portal for geeks, GeeksforGeeks* [online]. January 2023 [cit. 18.02.2023]. Dostupné z: https://www.geeksforgeeks.org/business-logic-layer

[20] *Data-Access Layer, A computer science portal for geeks, GeeksforGeeks* [online]. January 2023 [cit. 18.02.2023]. Dostupné z: https://www.geeksforgeeks.org/data-access-layer

[21] *Single Page Applications: A Powerful Design Pattern for Modern Web Apps, Medium – Where good ideas find you.* [online]. January 2023 [cit. 18.02.2023]. Dostupné z: https://medium.com/a-lady-dev/single-page-applications-a-powerful-design-pattern-for-modern-web-apps-ec3590bb7e7a#:~:text=A%20SPA%20is%20a%20web,new%20page%20from%20the%20server

[22] *Single Page Application Design Pattern for Modern Web Apps. Rapidops* [online]. January 2023 [cit. 18.02.2023]. Dostupné z: https://www.rapidops.com/blog/single-page-application-design

[23] *Balsamiq. Rapid, Effective and Fun Wireframing Software* [online]. January 2023 [cit. 07.02.2023]. Dostupné z: https://balsamiq.com

[24] *Why Choose Java For Web Application Software Development?. Digital IT Services & Development Service Provider, Infiraise* [online]. January 2022 [cit. 18.02.2023]. Dostupné z: https://www.infiraise.com/why-choose-java-for-web-application-software-development/#:~:text=Java%20allows%20you%20to%20create,they%20are%20making%20them%20for

[25] *Spring Boot - Introduction.* [online]. January 2023 [cit. 18.02.2023]. Dostupné z: https://www.tutorialspoint.com/spring_boot/spring_boot_introduction.htm

[26] *Apache Maven - Wikipedia.* [online]. January 2021 [cit. 18.02.2023]. Dostupné z: https://en.wikipedia.org/wiki/Apache_Maven

[27] *Gradle Tutorial - Javatpoint.* [online]. January 2021 [cit. 18.02.2023]. Dostupné z: https://www.javatpoint.com/gradle

[28] *What Is TypeScript? Pros and Cons of TypeScript vs. JavaScript, STX Next* [online]. January 2023 [cit. 18.02.2023]. Dostupné z: https://www.stxnext.com/blog/typescript-pros-cons-javascript

[29] *React (JavaScript library) - Wikipedia.* [online]. February 2023 [cit. 18.02.2023]. Dostupné z: https://en.wikipedia.org/wiki/React_(JavaScript_library)

[30] *React vs Angular: Which One is Best for Your Next Front-end Project?, Radixweb* [online]. February 2015 [cit. 18.02.2023]. Dostupné z: https://radixweb.com/blog/react-vs-angular

[31] *Relational vs. Non-Relational Databases, Pluralsight LLC* [online]. February 2023 [cit. 18.02.2023]. Dostupné z: https://www.pluralsight.com/blog/software-development/relational-vs-non-relational-databases

[32] *Relační databáze vs. nerelační databáze: Čím se liší?, MasterDC* [online]. February 2023 [cit. 18.02.2023]. Dostupné z: https://www.master.cz/blog/relacni-databaze-nerelacni-databaze-jake-jsou-rozdily

[33] *What Is A Non-Relational Database?, MongoDB: The Developer Data Platform* [online]. February 2023 [cit. 18.02.2023]. Dostupné z: https://www.mongodb.com/databases/non-relational#:~:text=

Non%2Drelational%20databases%20are%20often,history%20and%20credit%20card%20information

[34] *PostgreSQL: About, PostgreSQL: The world's most advanced open source database* [online]. February 1996 [cit. 18.02.2023]. Dostupné z: https://www.postgresql.org/about

[35] *What is OpenAPI?, OpenAPI Initiative* [online]. February 2022 [cit. 18.02.2023]. Dostupné z: https://www.openapis.org/what-is-openapi

[36] *Docker overview, Docker Documentation* [online]. February 2013 [cit. 18.02.2023]. Dostupné z: https://docs.docker.com/get-started/overview

[37] *What is HTTPS?, Cloudflare* [online]. February 2023 [cit. 18.02.2023]. Dostupné z: https://www.cloudflare.com/learning/ssl/what-is-https/

[38] *Enabling HTTPS on your servers, web.dev* [online]. February 2023 [cit. 18.02.2023]. Dostupné z: https://web.dev/enable-https/

[39] *What is a reverse proxy?, Cloudflare* [online]. February 2023 [cit. 18.02.2023]. Dostupné z: https://www.cloudflare.com/learning/cdn/glossary/reverse-proxy

[40] *Authentication vs Authorization: What's the Difference?, Javatpoint* [online]. February 2011 [cit. 18.02.2023]. Dostupné z: https://www.javatpoint.com/authentication-vs-authorization

[41] *HTTP basic authentication, IBM Corporation* [online]. February 2017 [cit. 18.02.2023]. Dostupné z: https://www.ibm.com/docs/en/cics-ts/5.4?topic=concepts-http-basic-authentication

[42] *Replay attack, Wikipedia* [online]. January 2023 [cit. 18.02.2023]. Dostupné z: https://en.wikipedia.org/wiki/Replay_attack

[43] *JSON Web Token Introduction, jwt.io* [online]. February 2023 [cit. 18.02.2023]. Dostupné z: https://jwt.io/introduction

[44] *IntelliJ IDEA – the Leading Java and Kotlin IDE, JetBrains: Essential tools for software developers and teams* [online]. February 2023 [cit. 20.04.2023]. Dostupné z: https://www.jetbrains.com/idea

[45] *java-gitlab-api 4.1.1 javadoc (org.gitlab), Free Java Doc hosting for open source projects - javadoc.io* [online]. February 2023 [cit. 20.04.2023]. Dostupné z: https://javadoc.io/doc/org.gitlab/java-gitlab-api/latest

[46] *gitlab4j-api 6.0.0-rc.1 javadoc (org.gitlab4j), Free Java Doc hosting for open source projects - javadoc.io* [online]. February 2023 [cit. 20.04.2023]. Dostupné z: `https://javadoc.io/doc/org.gitlab4j/gitlab4j-api/latest`

Příloha A

Seznam použitých zkratek

API Application Programming Interface

CD Continuous delivery

CI Continuous integration

CSS Cascading Style Sheets

ČVUT České vysoké učení technické v Praze

FIT Fakulta informačních technologií

HTML HyperText Markup Language

HTTP HyperText Transfer Protocol

HTTPS HyperText Transfer Protocol Secure

JSON JavaScript Object Notation

JWT JSON Web Token

REST Representational state transfer

SSH Secure Shell

SSO Single Sign On

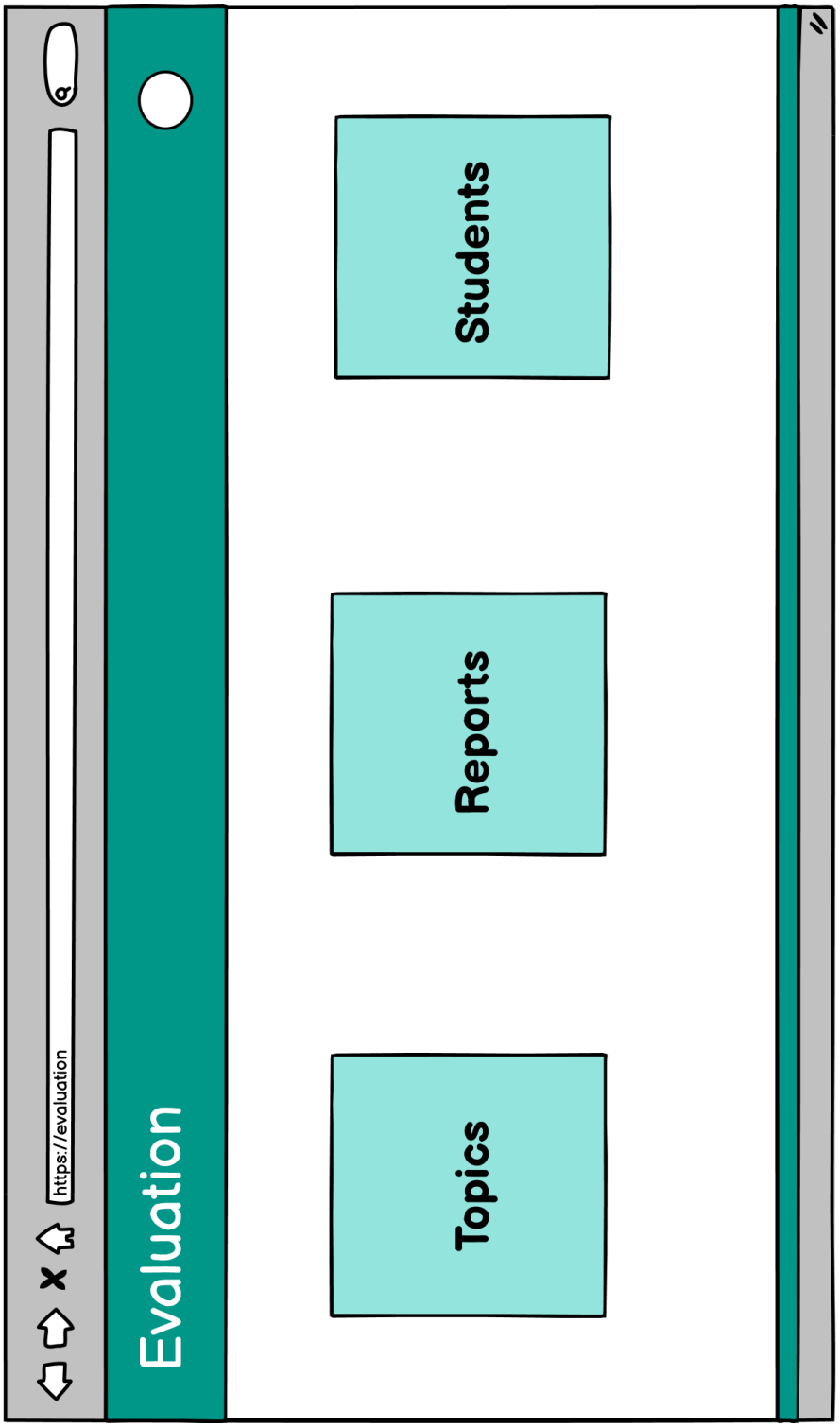
UC Use Case

UML Unified Modeling Language

XML Extensible Markup Language

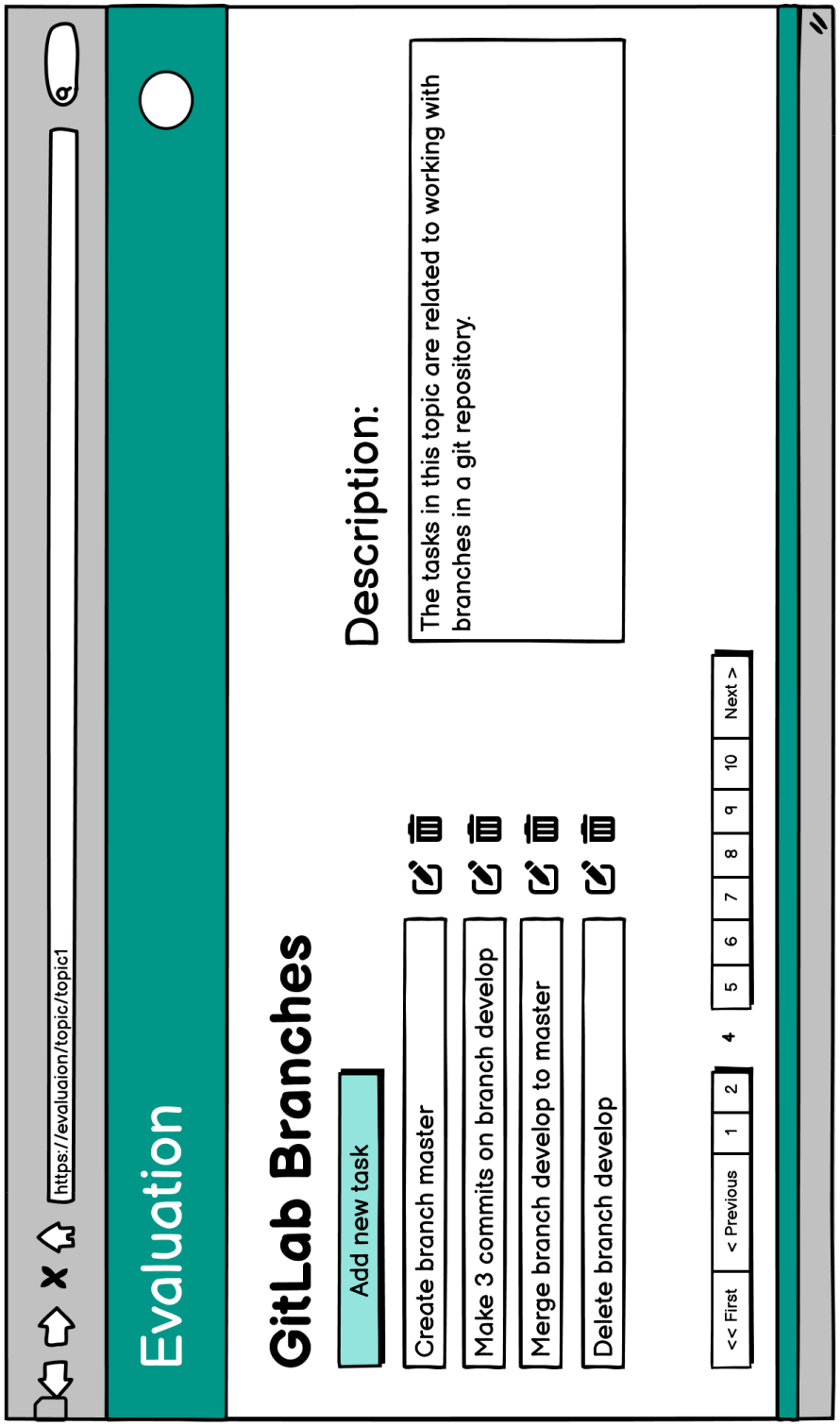
Příloha B

Drátěné modely obrazovek aplikace

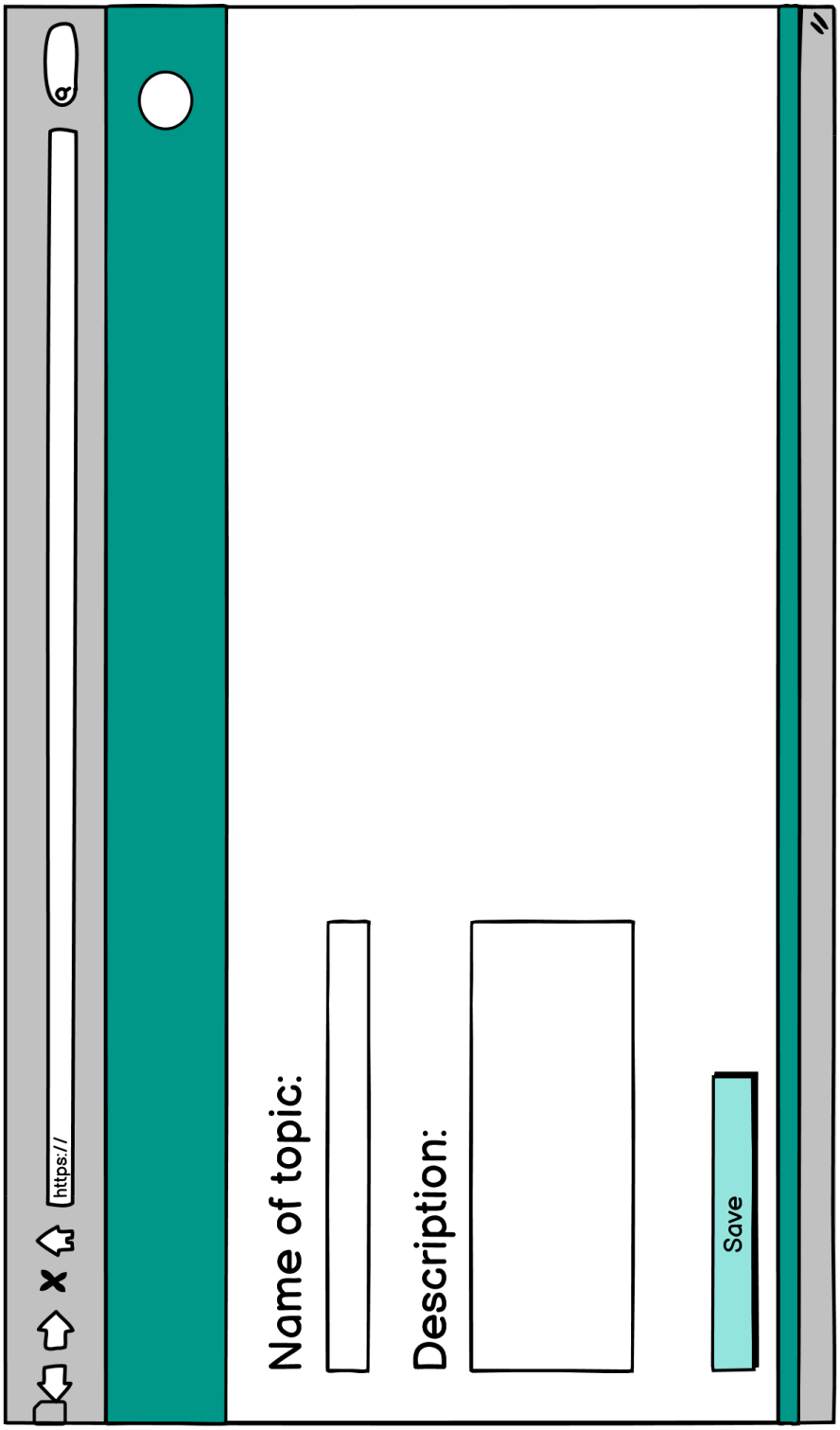


Obrázek B.1: Domovská stránka

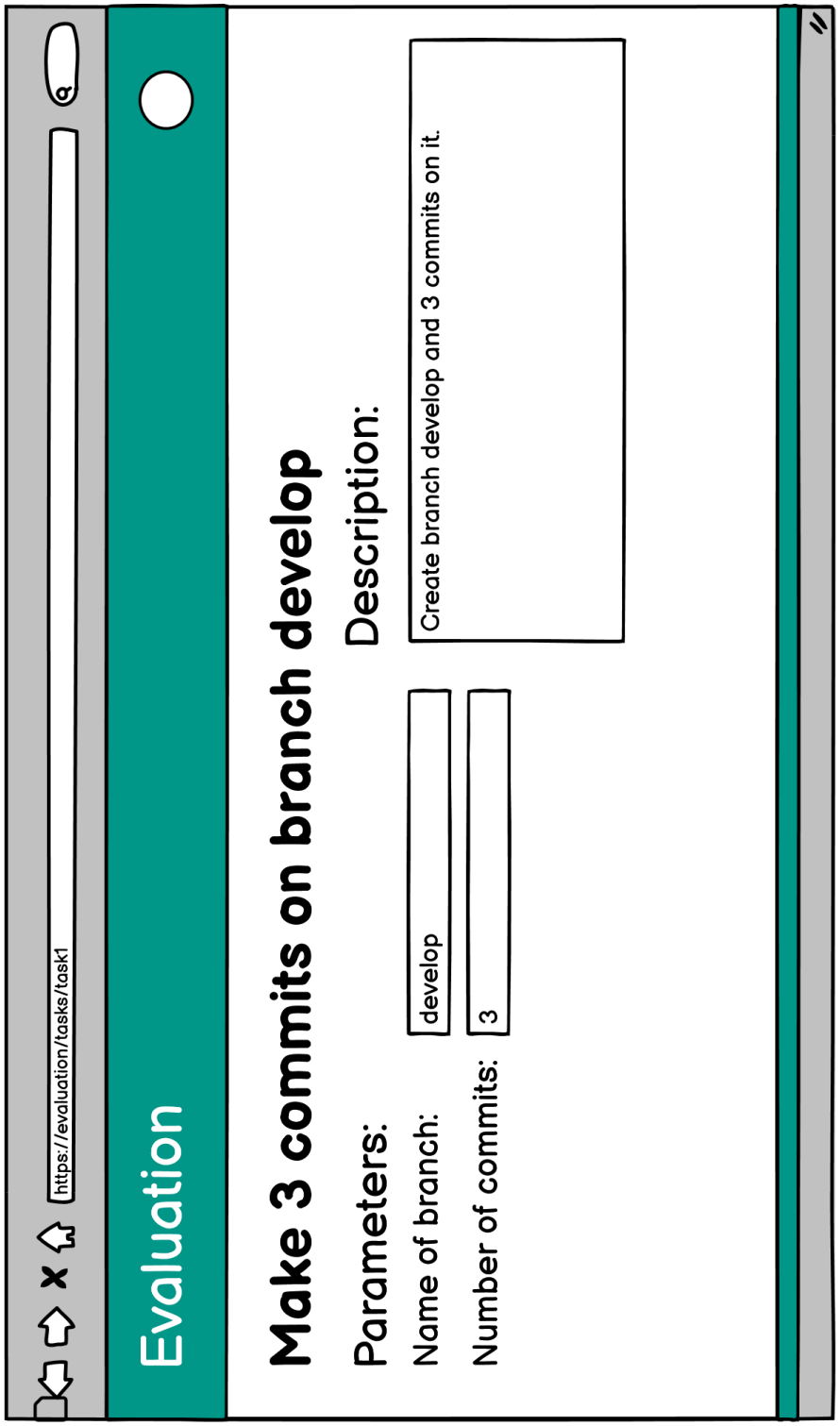
Obrázek B.2: Témata



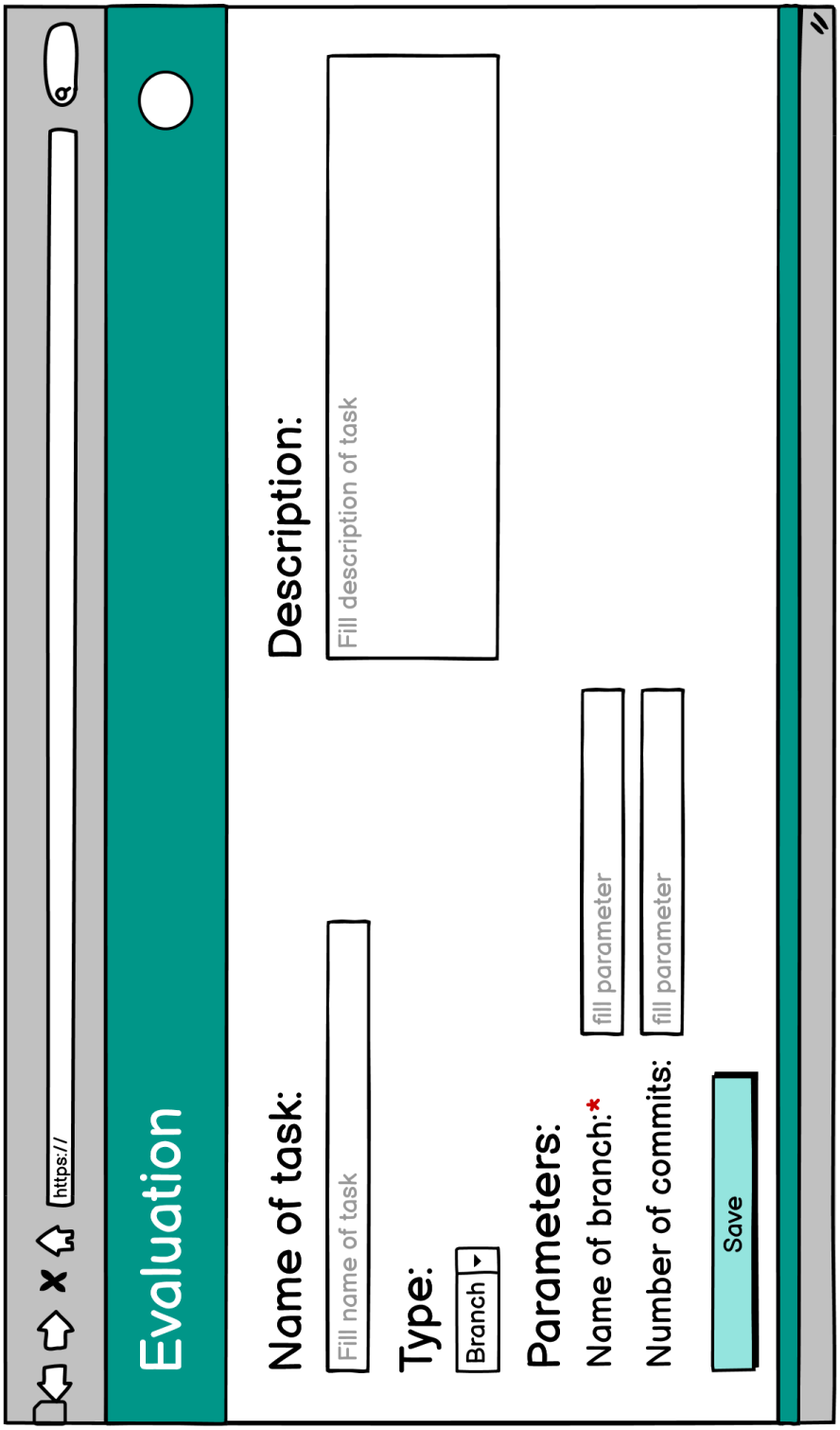
Obrázek B.3: Téma



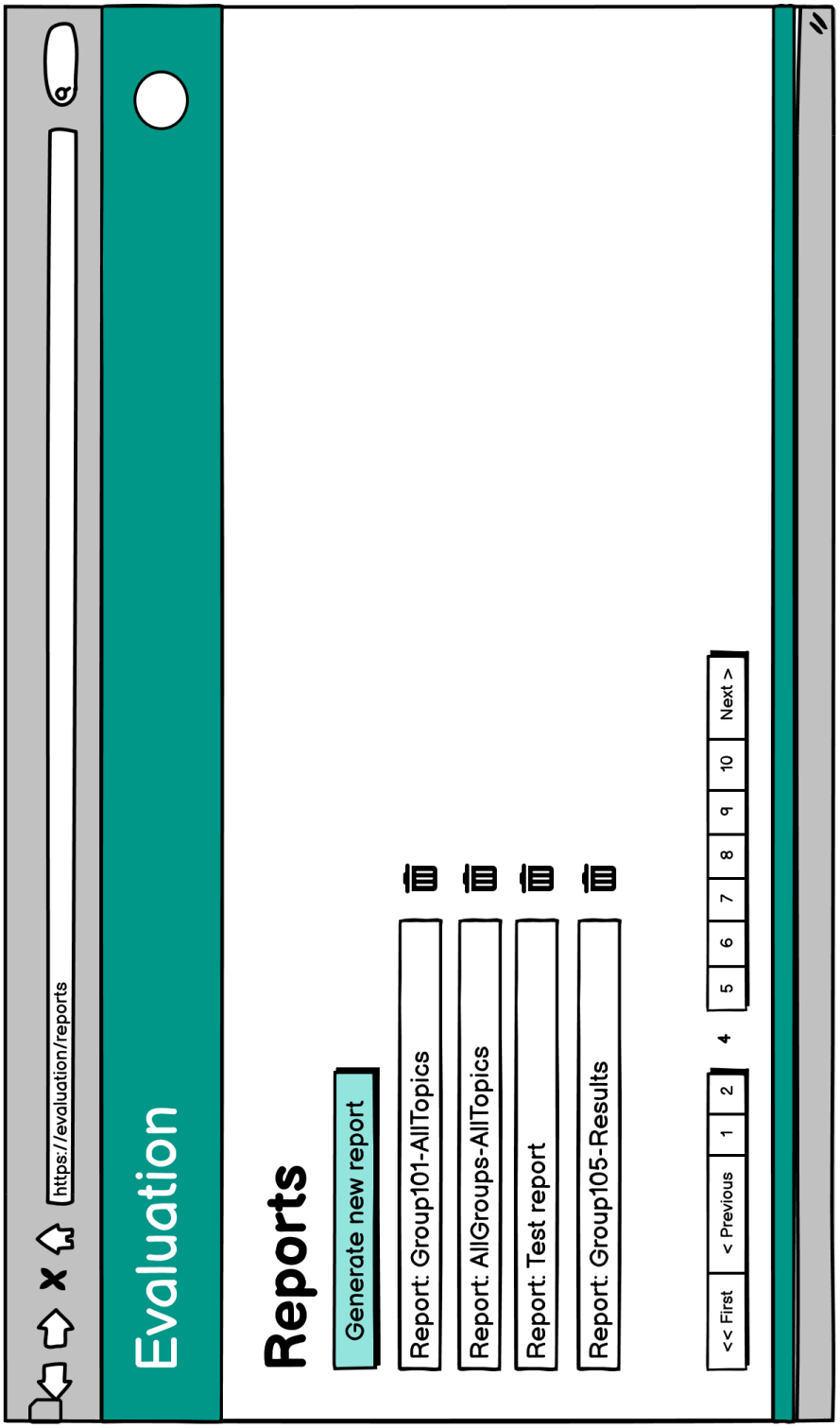
Obrázek B.4: Přidání nebo editování tématu



Obrázek B.5: Úkol

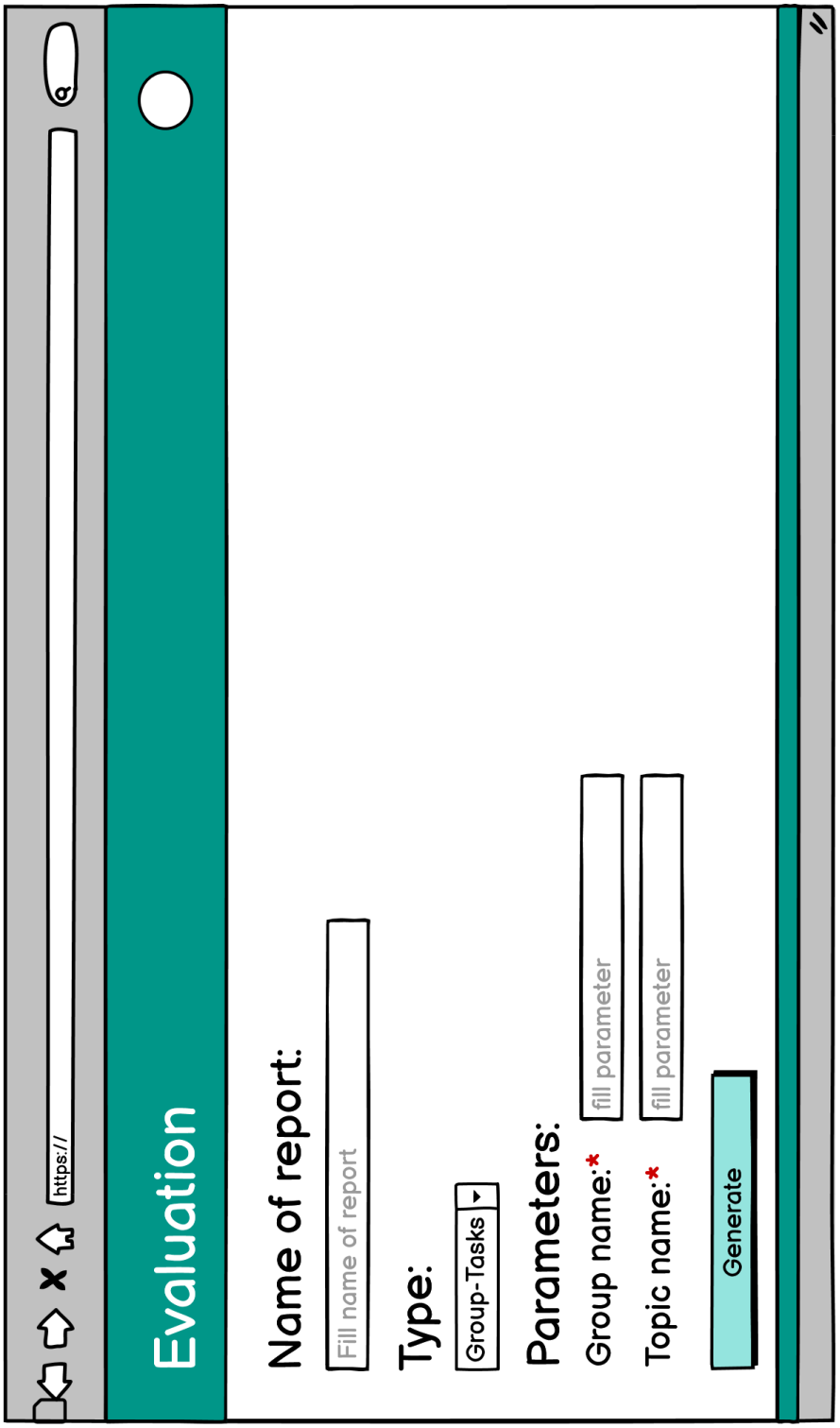


Obrázek B.6: Přidání nebo editování úkolu

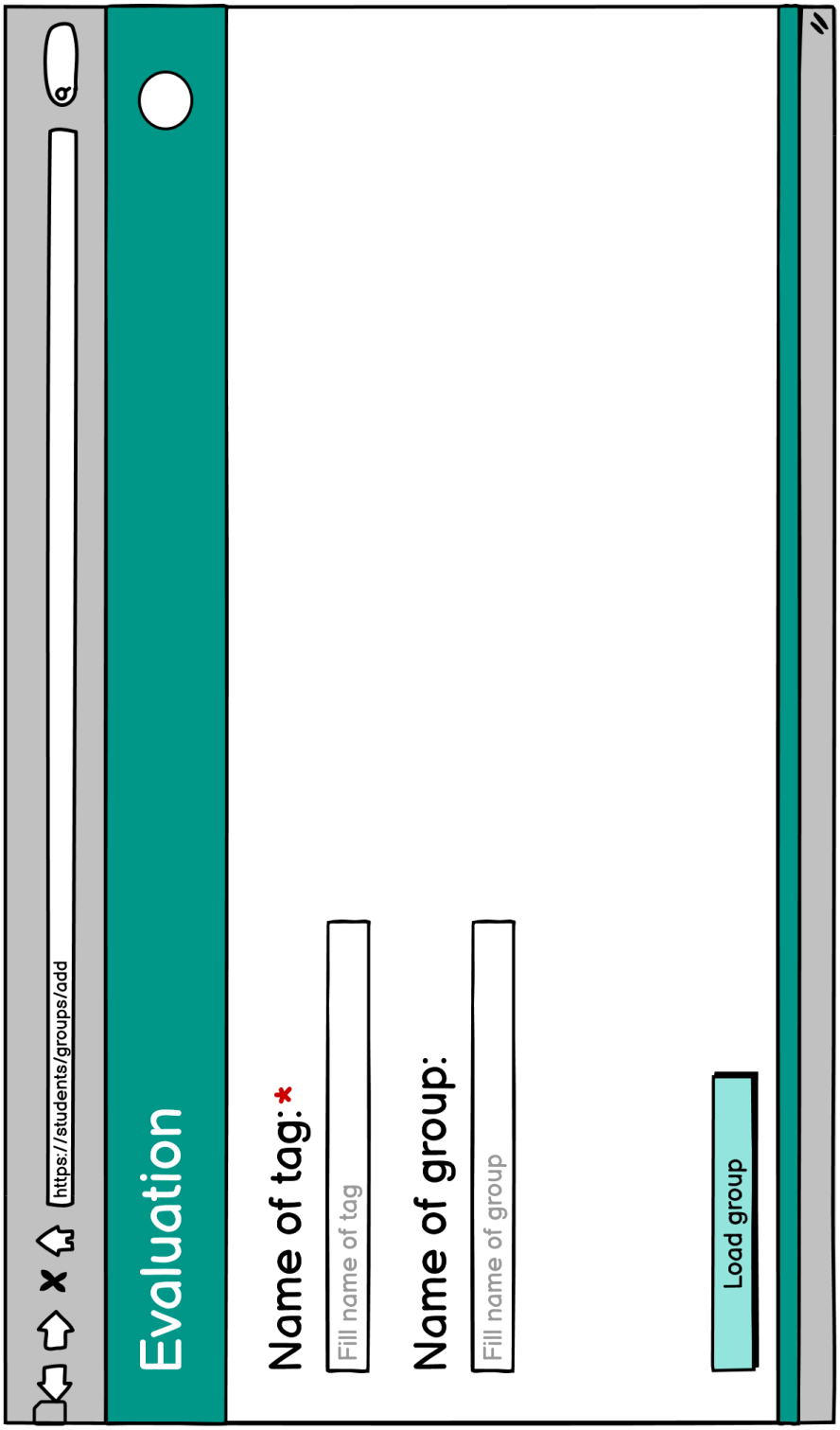


Obrázek B.7: Reporty

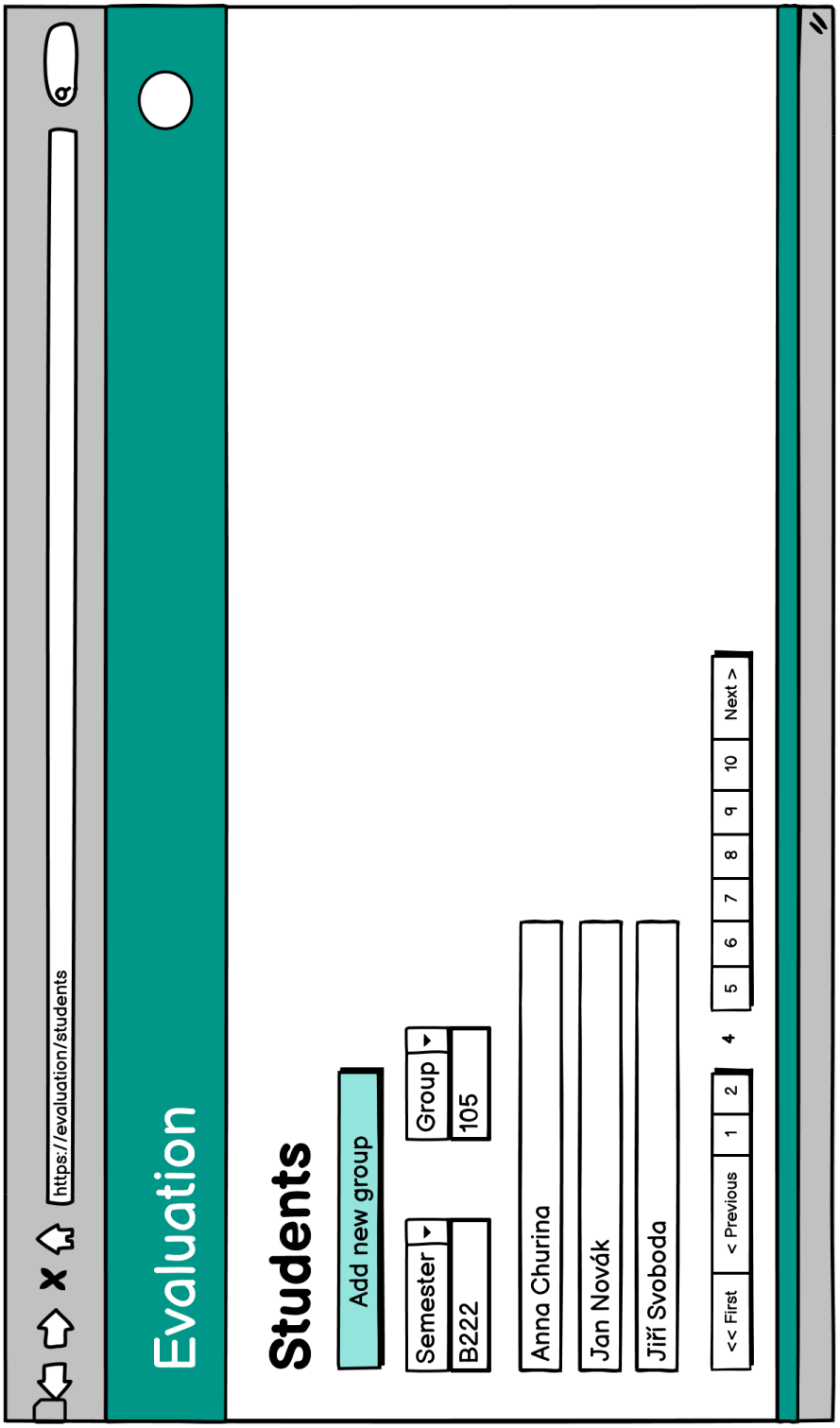
Obrázek B.8: Report



Obrázek B.9: Vytvoření reportu



Obrázek B.10: Přidání skupiny



Obrázek B.11: Studenty

Obrázek B.12: Student

Příloha C

Obsah odevzdané přílohy

- `readme.txt` ..... stručný popis obsahu přílohy
- `src`
  - `implementation` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma textu práce ve formátu LaTeX
- `text`
  - `thesis.pdf` ..... text práce ve formátu PDF