Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Aplikace pro správu a udržení konzistence úložiště projektových dat

Plzeň 2023 Adam Šmucr

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2022/2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Adam ŠMUCR**
Osobní číslo:	**A20B0246P**
Studijní program:	**B0613A140015 Informatika a výpočetní technika**
Specializace:	**Informatika**
Téma práce:	**Aplikace pro správu a udržení konzistence úložiště projektových dat**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se se strukturou dat v nástrojích pro řízení projektů, s projektem SPADe a se strukturou jeho databáze.
2. Navrhněte aplikaci vhodnou pro správu vybraných aspektů dat ve SPADe databázi, která umožní pomocí grafického rozhraní uživatelské opravy způsobené nevhodnou automatickou klasifikací vybraných položek a pomůže udržovat konzistenci dat.
3. Implementujte navrženou aplikaci v desktopové nebo webové formě. Při realizaci klaďte důraz na její pozdější rozšiřitelnost.
4. Ověřte pomocí sady testů funkčnost a použitelnost aplikace.

Rozsah bakalářské práce: **doporuč. 30 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

Dodá vedoucí bakalářské práce

Vedoucí bakalářské práce: **Ing. Petr Pícha**
Katedra informatiky a výpočetní techniky

Datum zadání bakalářské práce: **3. října 2022**
Termín odevzdání bakalářské práce: **4. května 2023**

L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

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

V Plzni dne 25. října 2022

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 4. května 2023

Adam Šmucr

Abstract

The SPADe tool is used to mine project data from ALM tools in order to detect bad practices. The aim of this bachelor thesis is to design and implement an application suitable for managing selected aspects of data in the SPADe database, which will allow corrections caused by inappropriate classification of selected items through a graphical interface and will help to maintain data consistency. To achieve the goal of this thesis, an analysis of the data structure in the project management tools and the database of the SPADe tool was carried out. In the next part, an extension of the web application was designed to provide users a way to manage selected aspects of the data, which was subsequently implemented. Finally, the results of the work were verified by a set of tests.

Abstrakt

Nástroj SPADe slouží k dolování projektových dat z nástrojů pro řízení projektů za účelem detekce špatných praktik při řízení těchto projektů. Cílem této bakalářské práce je navrhnout a implementovat aplikaci vhodnou pro správu vybraných aspektů dat ve SPADe databázi, která umožní pomocí grafického rozhraní uživatelské opravy způsobené nevhodnou klasifikací vybraných položek a pomůže tak udržovat konzistenci dat. Pro dosažené cílů této práce byla analyzována struktura dat v nástrojích pro řízení projektů a databáze nástroje SPADe. V další části bylo navrženo rozšíření již existující webové aplikace o uživatelskou správu vybraných aspektů dat, které bylo následně implementováno. V závěru byly výsledky práce ověřeny sadou testů.

Poděkování

Tímto bych chtěl poděkovat vedoucímu mé práce Ing. Petru Píchovi za hodnotné rady, připomínky a především za čas, který strávil vedením této práce.

Obsah

1 Úvod

Práce na rozsáhlém softwarovém projektu obsahuje nespočet aktivit, které přímo vedou k úspěšnému dokončení projektu nebo nepřímo podporují jeho vývoj. Tyto aktivity musí proběhnout správně, aby byl zajištěn úspěch projektu. Často však dochází při vývoji k chybám v oblasti projektového řízení, které pak vedou k selhání projektu. Ve většině případů se jedná o známé chybné praktiky, které nazýváme anti-vzory. Kdybychom dokázali nevhodné praktiky detekovat v rámci řízení projektu a adekvátně bychom na ně zareagovali, mohlo by to zvýšit pravděpodobnost úspěchu projektu.

Z výše uvedených důvodů vzniká na Katedře informatiky a výpočetní techniky Fakulty aplikovaných věd Západočeské univerzity sada nástrojů s názvem Software Process Anti-Pattern Detector (SPADe). Tento systém slouží ke sběru dat z často používaných Application Lifecycle Management nástrojů (ALM), které obsahují informace o průběhu softwarových projektů. Tato data nástroj SPADe akumuluje a ukládá do unifikovaného datového modelu, kde se následně provádí automatické detekce vybraných anti-vzorů. Při dolování dat z ALM nástrojů však musí být vlivem různých koncepcí jejich datových modelů použity další analýzy pro potřeby klasifikace těchto dat, které nejsou vždy spolehlivé. Tím v databázi nástroje SPADe vznikají nekonzistence, které je třeba vyřešit.

Cílem této bakalářské práce je navrhnout a vytvořit vhodnou aplikaci s grafickým uživatelským rozhraním pro správu vybraných aspektů dat ve SPADe databázi, která by umožnila uživatelské opravy nevhodné automatické klasifikace a tím pomohla k udržení konzistence dat.

Práce je rozdělena do čtyř částí. V první části je uveden kontext práce zabývající se ALM nástroji, ze kterých jsou projektová data dolována a seznámení s projektem SPADe a jeho databází. V druhé části jsou vybrány ty aspekty databáze, které způsobují nekonzistenci dat a navržen způsob jakým by bylo možné opravit chyby způsobené např. nevhodnou automatickou klasifikací. Ve třetí části je popsána implementace navžené aplikace a v poslední části jsou uvedeny informace o tom jak byla aplikace otestována. Na závěr jsou diskutovány výsledky práce.

2 ALM nástroje a SPADe

Tato kapitola obsahuje krátký úvod do řízení softwarových projektů, především s ohledem na metodiky vývoje, na kterých staví některé koncepty použité v této práci. Dále jsou představeny nástroje pro správu a řízení projektů, jejich základní typy a zástupci s uvedením rozdílů mezi nimi z pohledu jejich datového modelu a funkcí, které nabízejí. V druhé části je popsán systém Software Proccess Anti-Pattern Detector (SPADe), který z těchto nástrojů doluje a analyzuje data o softwarových projektech. Rozšíření tohoto systému je zároveň předmětem této práce.

2.1 Softwarový proces

Tvorba softwarového produktu zahrnuje několik druhů aktivit, které je nutné v průběhu práce kvalitně splnit. Jedná se například o návrh systémů, obstarávání zdrojů, řízení osob a komunikace nebo testování. Organizovaný souhrn těchto aktivit se nazývá proces vývoje software, nebo zkráceně softwarový proces.

Obecně jde o opakovanou transformaci vstupu na výstup. V rámci řízení projektů můžeme proces rozdělit do několika subprocesů a ty pak do následujících skupin [11]:

- **iniciační procesy** – definice projektu či jeho část, shánění zdrojů,
- **procesy plánování** – stanovení plánu projektu nebo jeho části především s ohledem na řízení času, rozsahu a nákladů,
- **výkonné procesy** – koordinace osob a zdrojů, vytváření produktu,
- **procesy sledování a kontroly** – měření a monitorování postupu s porovnáním oproti plánu, zpracování zpráv o postupu práce,
- **závěrečné procesy** – akceptace projektu či jeho části, administrativní úkony, validace.

2.1.1 Základní terminologie

V této sekci jsou představeny některé základní obecné pojmy vztahující se k softwarovému vývoji.

- **Proces** – šablona pro projektové aktivity, přeměna vstupů na výstupy, které je potřeba v rámci dosažení cíle procesu splnit.
- **Projekt** – instance procesu s vlastními zdroji, kontextem a náplní vedoucí k vytvoření softwarového produktu.
- **Metodika** – ověřený a polo-standardizovaný předpis obsahující (krom jiného) proces, který zvyšuje jeho efektivitu.
- **Role** – přiřazována osobám v projektu, definuje kompetence a povinnosti. Jedna role může být přiřazena více osobám (skupinám), stejně tak osoba může zastávat více rolí.
- **Artefakt** – identifikovatelné entity, které jsou vstupem či výstupem úkolů v projektu nebo se k nim nějak vztahují (soubory, wiki stránky, infrastruktura, části kódu, apod.).
- **Aktivita** – činnost definující zaměření procesů.
- **Release** – stabilní verze produktu, která obsahuje v danou chvíli očekávatelné artefakty a je vhodná pro prezentaci zákazníkovi.
- **Konfigurace** – konkrétní množina pracovních položek souvisejících s projektem v definovaných verzích či stavech.

2.1.2 Metodiky vývoje software

Aby bylo možné začít a úspěšně dokončit projekt, je nutné použít vhodnou metodiku vývoje. Metodika definuje doporučené postupy (tj. procesní model), pravidla a nástroje, které jsou ve vývoji využity, a také určuje, na které fáze bude projekt rozdělen, jaké aktivity budou součástí vývoje a jakým způsobem se na nich bude pracovat. Výběr metodiky je velmi důležitý, protože může výrazně zvýšit efektivitu celého softwarového vývoje.

Pokud se zajímáme o strukturu dat, která jsou v rámci projektů sbírána, je důležité rozlišit přístupy těchto metodik k dělení projektu na menší části, jako například fáze, iterace nebo disciplíny.

Vodopádový model

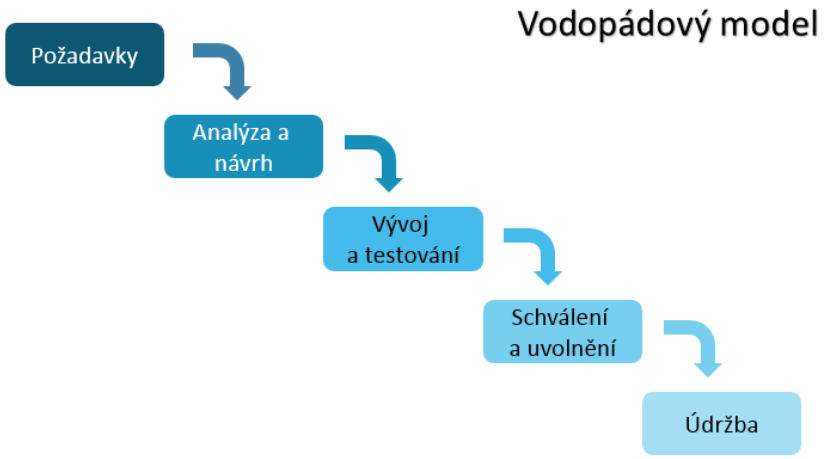
Vodopádový model je zástupcem sekvenčních modelů vývoje, který předpokládá, že jednotlivé fáze probíhají postupně podle předem známého schématu. Definované fáze jsou za sebou sekvenčně řazeny, neprobíhají současně a mají jasné pořadí. Ve vodopádovém modelu mohou být méně či více detailní fáze, ale nejčastěji se jedná o tyto [6]:

- definice požadavků systému,
- analýza, návrh systému a softwaru,
- implementace softwaru,
- integrace modulů a testování,
- provoz a údržba.

V první fázi je nutné si ujastnit požadavky klienta na požadovaný systém a analyzovat doménu, ve které se bude software tvořit. V rámci analýzy a návrhu se pak určí celková architektura systému a popíší se vztahy v jeho jednotlivých komponentách. Po návrhu se celý systém začne implementovat na základě poznatků a dokumentů vypracovaných v předchozích fázích. Poté je provedena integrace jednotlivých komponent do celkového systému a zahájí se testování. Po úspěšném předání projektu klientovi probíhá provoz systému a jeho údržba spojená s podporou.

Jelikož je projekt od začátku rozdělen do sekvenčních fází, je velice obtížné reagovat na jakékoliv změny ze strany klienta. V reálných projektech se také ukazuje, že na některé otázky či návrhové detaily je možné odpovědět až v rámci implementace, kde se již nemůžeme vrátit k předchozím fázím.

Vodopádový model je tedy málo flexibilní a vhodný spíše pro kratší úseky vývoje nebo malé projekty. Schéma vodopádového modelu je znázorněno na obrázku číslo 2.1.



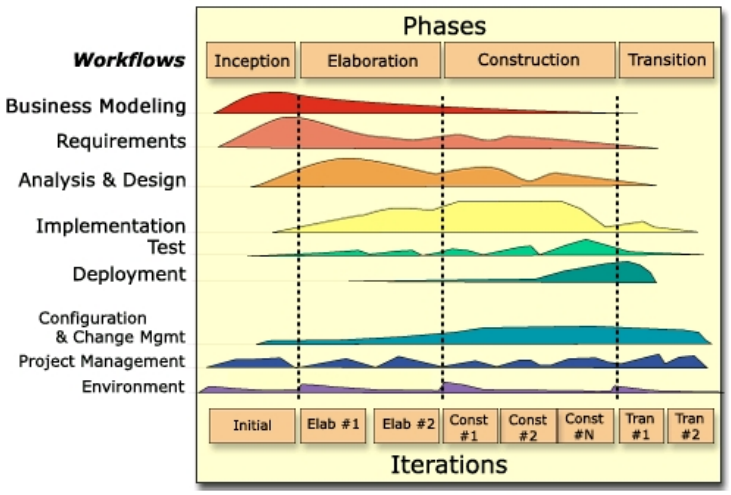
Obrázek 2.1: Schéma vodopádového modelu [6]

Unified Process

Unified Process je zástupcem iterativních a inkrementálních modelů vývoje softwaru. Model předpokládá, že požadavky na systém se budou během času měnit a tudíž využívá několika iterací v procesu vývoje. Každá iterace je pak složena z aktivit, které v rámci iterace vytvoří část plánovaného softwaru – inkrement. Tím se zkrátí čas mezi zadáním projektu a prezentováním výsledku, protože po jednotlivých inkrementech jsme schopni předvést dosavadní funkčnosti. Zároveň má zákazník možnost vyzkoušet si dosavadní produkt a na základě jeho zkušenosti konkretizovat nebo upravit své požadavky. Model rozděluje vývoj do čtyř základních fází, které jsou obecně ukončeny dosažením milníků definovaných sadou kritérií. Jde o fáze:

- zahájení (Inception),
- rozpracování (Elaboration),
- tvorba (Construction),
- předání (Transition).

V rámci jedné fáze je pak provedeno několik iterací obsahující disciplíny potřebné k dokončení celé fáze. Jde o tvorbu modelů, sběr a správu požadavků na systém, analýzu a návrh systému, implementace a testování a nasazení, řízení projektu a mnoho dalších. Disciplíny pak probíhají souběžně v rámci fáze nebo iterace. Na obrázku 2.2 je znázorněno rozdělení fází a orientační poměr jednotlivých disciplín v nich.

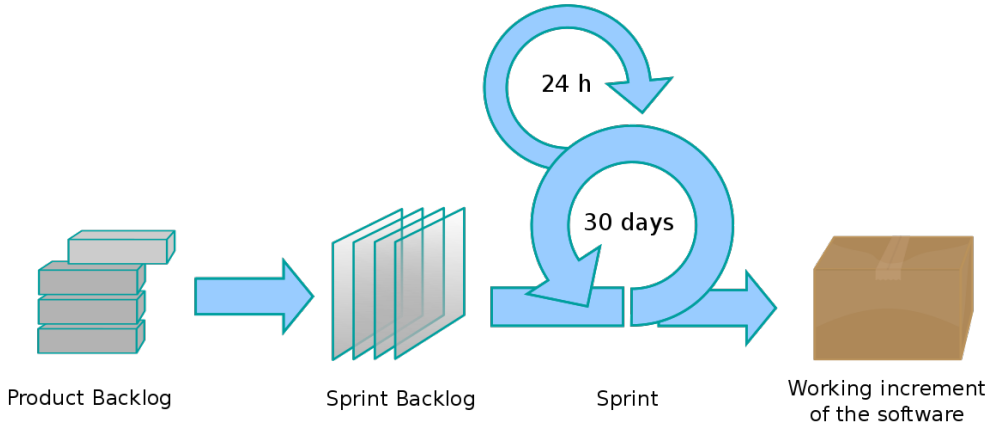


Obrázek 2.2: Schéma Unified Process [4]

SCRUM

SCRUM je zástupcem agilní metodiky vývoje, které jsou moderním přístupem v řízení softwarových projektů. Na rozdíl od vodopádového modelu, SCRUM počítá s tím, že bude docházet ke změnám požadavků zákazníka a věří, že je tuto změnu mnohem efektivnější umožnit, než jí bránit. Tomu je podřízen samotný vývoj spočívající na krátkých iteracích a inkrementech, rychlé a efektivní komunikaci mezi zákazníkem a členy vývojového týmu či automatickém testování [12].

Na obrázku 2.3 lze vidět jakým způsobem probíhají iterace (sprinty) v metodice SCRUM.



Obrázek 2.3: Schéma sprintů v metodice SCRUM [5]

2.1.3 Další metodiky vývoje

Kromě výše zmíněných metodik vývoje existují samozřejmě i další. Příkladem mohou být takzvané Lean (štíhlé) metodiky, které se zaměřují na zlepšení efektivity vývoje za pomoci maximální redukce času potřebného na vývoj. Zaměřují se tedy na finální produkt a procesy, které k němu přímo nevedou jsou potlačeny. Zástupce takové metodiky je např. Kanban.

Dalším příkladem jsou evoluční modely, které na počátku vývoje vytvoří základní verzi produktu a ta je pak upravována dle komentářů zákazníka. Použití této metodiky je vhodné především v případě, kdy cílový zákazník nedokáže definovat prvotní požadavky.

2.2 ALM nástroje

S rostoucím počtem vyvíjených softwarových projektů se také zvyšuje potřeba je efektivně řídit, aby se zvýšila šance na úspěšné dokončení projektu. Na samotném vývoji se podílí více stran zainteresovaných v projektu. Jedná se například o zadavatele, programátory, testery a projektové manažery, kteří spolu nutně musí komunikovat v rámci specifických procesů vývoje. Aby bylo možné tuto komunikaci zprostředkovat a zároveň účelně řídit procesy, využívá se Application Lifecycle Management nástrojů (ALM). Ty umožňují správu různých aspektů projetku a napomáhají s organizací jeho životního cyklu. Svými funkcemi umožňují tyto nástroje zlepšit koordinaci softwarového týmu především v oblasti řízení změn, správy verzí a chyb nebo při samotném plánování vývoje, aby bylo možné co nejlépe dosáhnout definovaných cílů.

2.2.1 Typy ALM nástrojů

Existuje celá řada komerčních či volně použitelných ALM nástrojů, které se zaměřují na různé aspekty vývoje projektu. Některé shromažďují data o změnových požadavcích, jiné se více zaměřují na správu verzí projektu v různých fázích a existují i kombinované nástroje, které pokrývají většinu procesů řízení. Obecně je lze rozdělit do několika skupin v závislosti na jejich primárních funkcích a zaměření [13].

V praxi je běžné, že se v rámci vývoje jednoho projektu využije více odlišných nástrojů, které dohromady pokryjí správu všech potřebných procesů. V sekcích 2.2.1 a 2.2.1 jsou zmíněny často používané typy ALM nástrojů zaměřující se na správu verzí a změnových požadavků. Samozřejmě existuje i celá řada dalších nástrojů, které jsou pak stručně popsány v sekci 2.2.1.

Nástroje pro správu verzí

Version Control System nástroje (VCS) umožňují řízení příspěvků více osob do jednoho zdrojového kódu a uchovávání jednotlivých verzí produktu v čase, se kterými lze následně pracovat. Pomocí těchto nástrojů lze také obnovit projekt do některé z předchozích verzí a zároveň poskytují informace o tom, kdo a kdy jaké části projektu upravoval.

Zástupcem této kategorie ALM nástrojů je především systém Git a webové služby na něm založené – GitHub[1] a GitLab[2] [1]. Za zmínku stojí také systém

[1]Webová stránka dostupná na `https://github.com`

[2]Webová stránka dostupná na `https://about.gitlab.com`

Apache Subversion[3] (SVN), který nahradil starší Concurrent Version System (CVS). Zmíněné nástroje mimo jiné umožňují vytváření více nezávislých vývojových větví, které lze následně značkovat a slučovat.

Nástroje pro správu změn

Nástroje pro správu změn slouží k řízení změnových požadavků ve vyvíjené aplikaci, které mohou představovat úpravu chování nějaké části systému nebo jeho vylepšení. Dále napomáhají při řešení a odstraňování chyb nalezených v projektu.

Vývojáři, testeři, nebo v některých případech i zákazníci mohou do tohoto nástroje nahlásit konkrétní selhání aktuální verze aplikace nebo požadavek na její změnu. Tím vzniká v nástroji seznam podnětů k úpravě produktu, tzv. tickety či issues, které jsou poté přiděleny jednotlivým vývojářům. Každý ticket má v závislosti na jeho druhu přidělenu prioritu, závažnost a další potřebné informace k jeho klasifikaci. Dle jeho popisu a zařazení pak vývojář posoudí jakým způsobem bude podnět zpracován. Jedná-li se o chybu aplikace, může být opravena nebo označena jiným odpovídajícím řešením (např. nejedná se o chybu, duplicitní, apod.)

Tímto typem nástroje je např. systém Bugzilla[4], Redmine[5] či Atlassian Jira[6]. Každý nástroj má však svá specifika a některé nabízí další odlišné funkce z oblasti řízení projektů.

Další typy ALM nástrojů

Kromě výše zmíněných typů nástrojů, existují i další, které se zaměřují na jinou oblast vývoje. Jejich příklady jsou stručně uvedeny v této sekci.

Jedním z nich jsou nástroje sloužící k nasazení či integraci aplikace a jejích částí za pomoci automatizace při důrazu na plynulost tohoto procesu. Tyto nástroje se také označují jako Continuous Integration / Continuous Delivery (CI/CD). Přičemž první část zabývající se integrací dovoluje programátorovi provedené změny automatizovaně zkontrolovat a na základě těchto testů posoudí, zda je bezpečné nový kód začlenit do programu. Druhá část se pak zabývá nasazením celé aplikace do produkce [10]. Zástupcem tohoto typu nástroje je např. Jenkins[7].

[3] Webová stárnka dostupná na `https://tortoisesvn.net`
[4] Webová stránka dostupná na `https://www.bugzilla.org`
[5] Webová stránka dostupná na `https://www.redmine.org`
[6] Webová stránka dostupná na `https://www.atlassian.com/software/jira`
[7] Webová stránka dostupná na `https://www.jenkins.io`

Dalším typem nástrojů jsou systémy pro kontrolu kvality produktu, např. Selenium[8] a Squash[9]

Pro samotnou komunikaci mezi vývojáři se pak používá nástroje typu Slack[10]. Za ALM nástroje lze považovat veškeré podpůrné aplikace, které slouží k efektivnímu řízení kolaborativního projektu vývoje softwarového produktu.

2.2.2 Datové modely ALM nástrojů

Jak již bylo řečeno, existuje velké množství nástrojů, které podporují vývoj softwarových projektů z mnoha pohledů. Každý nástroj však pracuje s obecně známými koncepty různým způsobem. Některé nabízejí větší customizaci nastavení, používají jiná názvosloví či fungují rozdílně, což zapřičiňuje odlišnosti v datových modelech různých nástrojů. Problém těchto odlišností se projeví v momentě, kdy chceme ukládat projektová data z různorodých nástrojů do unifikovaného datového modelu, který má například aplikace SPADe popsaná v sekci 2.3. Všechna data z ALM nástrojů tedy musí být uložena jednotným způsobem, což mohou níže popsané odlišnosti zesložiťovat. Proto bylo v rámci SPADe potřeba zavést další automatickou analýzu a kategorizaci dat, která však nedokáže vše analyzovat se stroprocentní přesností a mohou tak v databázi vznikat nekonzistence. Řešením je uživatelský zásah do struktury vybraných částí projektu, který umožní opravu vzniklých nekonzistencí [7]. Vybrané rozdíly v konceptech jednotlivých nástrojů jsou uvedeny v následujících sekcích.

Hierarchie projektů

Vytvářené softwarové projekty mohou být často součástí nějakého většího celku. Tento vztah některé ALM nástroje umožňují nastavit přímo při jeho vytváření, např. nástroj Redmine při definici projektu nabízí výběr rodičovského projektu. Tento koncept však není podporován ve všech nástrojích, příkladem je nástroj GitHub, který umožňuje v novějších verzích určité nastavení hierarchie repozitářů nebo tzv. submodulů pro knihovny a společné části kódu, ale tento koncept SPADe nevyužívá, protože byl vytvořen pro starší verze tohoto nástroje. Z těchto důvodů není možné ze všech použitých nástrojů získat hierarchii projektů a je tedy nutné nechat určitou část této funkcionality uživateli.

[8]Webová stránka dostupná na `https://www.selenium.dev`

[9]Webová stránka dostupná na `https://www.squashtest.com`

[10]Weboá stránka dostupná na `https://slack.com/`

Výčtové typy

Koncept kategorizace určitých částí projektů jako jsou role, priority, typy úkolů apod. je zastoupen téměř v každém issue tracking systému. Každý ale přistupuje k definici kategorií jiným způsobem a to i v rámci jednotlivých druhů výčtových typů. Zatímco je pro typy úkolů v nástrojích Jira, Assembla či Redmine definována základní množina výčtových typů (není však shodná napříč nástroji), GitHub tuto záležitost řeší pomocí labelů (značek), které si uživatel definuje sám. Podobným způsobem jsou řešeny i priortiy či stavy úkolů. Speciálním případem je pak definice vztahů mezi úkoly, kde se použité názvosloví značně liší mezi jednotlivými nástroji a např. GitHub nemá žádné možnosti jak tyto vztahy definovat. Problém klasifikace výčtových typů je tak problematičtější nejen z toho důvodu, že si sady kategorií v ALM nástrojích neodpovídají, ale zároveň jsou odlišné i u každého projektu. Příkladem může být výčtový typ pro priority, kde je u většiny nástrojů rozdílný počet klasifikačních tříd (cca 3 - 6) a zároveň mají ještě jiným způsobem namapované tzv. nadtřídy, které definují obecnější zařazení výčtového typu.

Iterace a fáze

Dále je nutné vzít v potaz, že každý projekt může být řízen pomocí různé metodiky vývoje. Některé mohou používat iterace, jiné fáze nebo určitou kombinaci. Ty jsou v nástrojích často definovány jedním společným datovým konceptem, který mezi použitou metodologií nerozlišuje. Zatímco v nástrojích pro správu chyb jsou iterace a fáze vnímány jako verze aplikace, na kterých probíhá testování, v GitHubu či Assemble jsou uloženy v datovém modelu jako milníky, kterých je potřeba dosáhnout.

Konfigurace a commity

Konfigurace obecně je množina artefaktů v projektu v dané verzi (ne nutně u všech stejné). V kontextu SPADe a jeho datového modelu je to sada změn artefaktů, úkolů nebo již existujících konfigurací provedených jedním člověkem v jeden moment (tj. jednou akcí). Jednou z možných forem konfigurace jsou commity v úložišti kódu. Konfigurace, které nejsou přímo commity, jsou pak získávány většinou z komentářů u ticketů nebo v záznamu historických změn, např. v issue trackeru nebo jiném ALM nástroji. Spřízněné s commity a VCS nástroji obecně jsou pak koncepty jednotlivých vývojových větví, tzv. branch, a tagů, které umožňují označení určitého bodu v historii repozitáře [2].

Další koncepty ALM nástrojů

Každý nástroj má nějakým způsobem definovány osoby, které pracují na projektu. Ty jsou však téměř vždy specifikovány na základě identit, díky kterým je možné osoby z ALM nástrojů dolovat.

Release je koncept, který je využíván k definici konfigurace, která je stabilní a použitelná v rámci prezentace zákazníkovi. Jelikož si však může vývojový tým definovat např. pomocí tagů podobná označení, není jejich dolování snadné a je nutné to provést za pomoci komentářů u pracovních položek.

Kategorie dokáží definovat zaměření pracovních úkolů a kategorizovat je do skupin. Aktivity využívají této funkce při dolování dat, jelikož definují množinu propojených úkolů právě na základě stejné nebo podobné kategorie.

2.3 Nástrojová sada SPADe

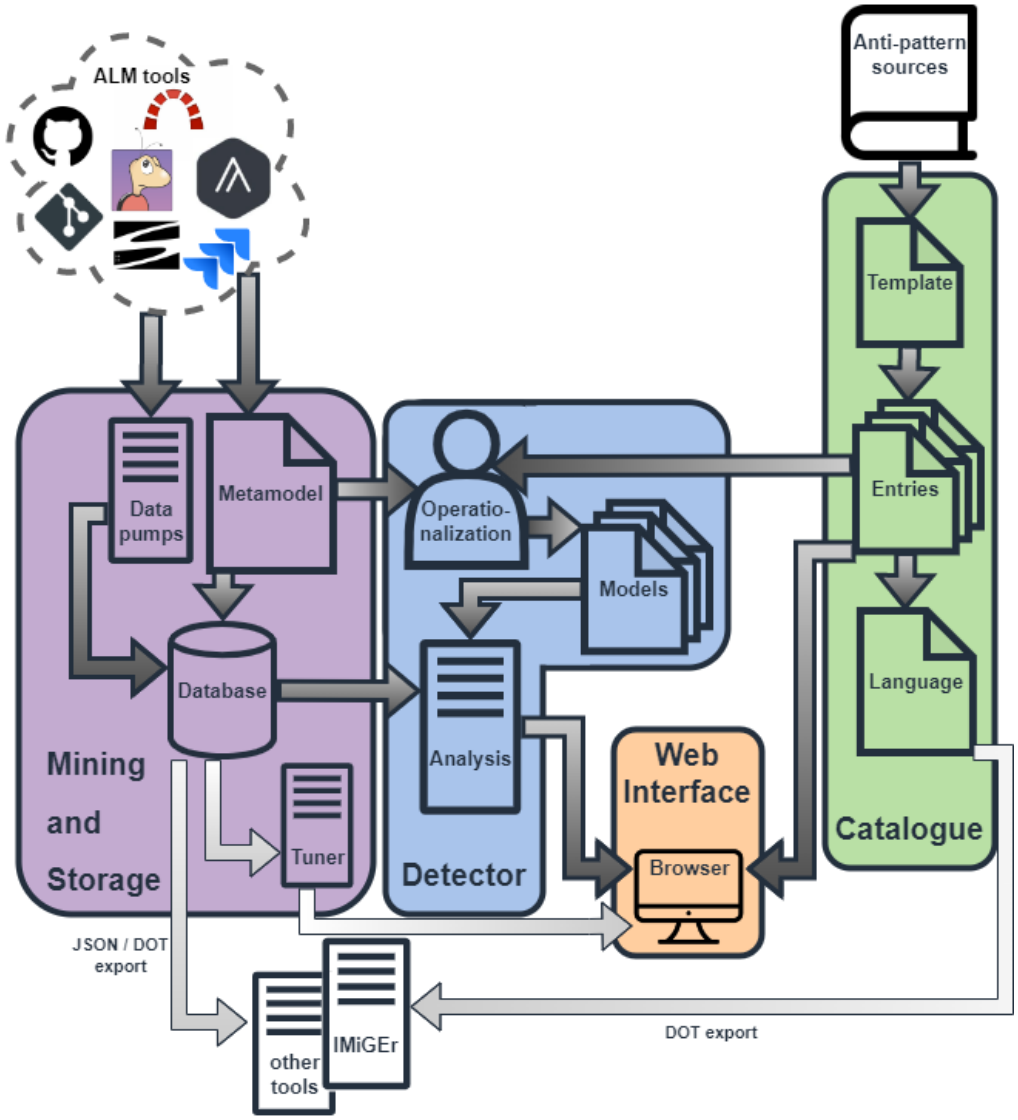
SPADe je aplikace vyvíjená na Katedře informatiky a výpočetní techniky Fakulty aplikovaných věd Západočeské univerzity v Plzni. Jejím úkolem je získávání dat z ALM nástrojů různých softwarových projektů a tato data kumulovat v jednotném datovém skladu. Nad získanými daty se poté provádí analýza s ohledem na identifikaci špatných praktik v oblasti řízení projektů, tzv. anti-vzorů [15]. Cílem tohoto nástroje je poskytnout informace projektovému manažerovi o výskytu těchto praktik a na základě vyhodnocených dat nabídnout jejich řešení. Včasným odhalením a zareagováním na přítomné anti-vzory se zvyšuje pravděpodobnost, že bude projekt úspěšně dokončen.

2.3.1 Architektura SPADe

Nástrojová sada SPADe je složena ze čtyř hlavních částí. První, nazvaná na obrázku 2.4 jako **Mining and Storage** slouží pro dolování dat a jejich uskladnění. Z vybraných ALM nástrojů se pomocí datových pump dolují data o projektech, která jsou následně ukládána do databáze s unifikovaným metamodelem. V druhé části nazvané **Catalogue** jsou na základě literatury shromážděny popisy projektových anti-vzorů, ze kterých vznikají záznamy popisující jakým způsobem v projektech danou nevhodnou praktiku detekovat. V **Detector**u se následně provede analýza, ve které se hledají definované anti-vzory v projektových datech z databáze. Výsledek je na samém konci prezentován pomocí **Web Interface**, který představuje webovou aplikaci s uživatelským rozhraním. Všechna data i katalogové záznamy o anti-vzorech lze také exportovat do formátu JavaScript Object Notation (JSON) nebo DOT, který je určen pro popis grafů. Tyto výstupy je pak možné využít v

další nástrojích, např. v Interactive Multimodal Graph Explorer (IMiGEr), který slouží pro vizualizaci grafů a usnadňuje jejich pochopení [3].

Ve schématu architektury je také možné vidět dvě barevné sady šipek, které zobrazují směr toku dat. Zatímco černě zbarvené šipky představují hlavní funkce nástroje, ty světlejší definují sekundární funkce systému, které jsou momentálně ve stádiu vývoje.



Obrázek 2.4: Architektura nástroje SPADe [8]

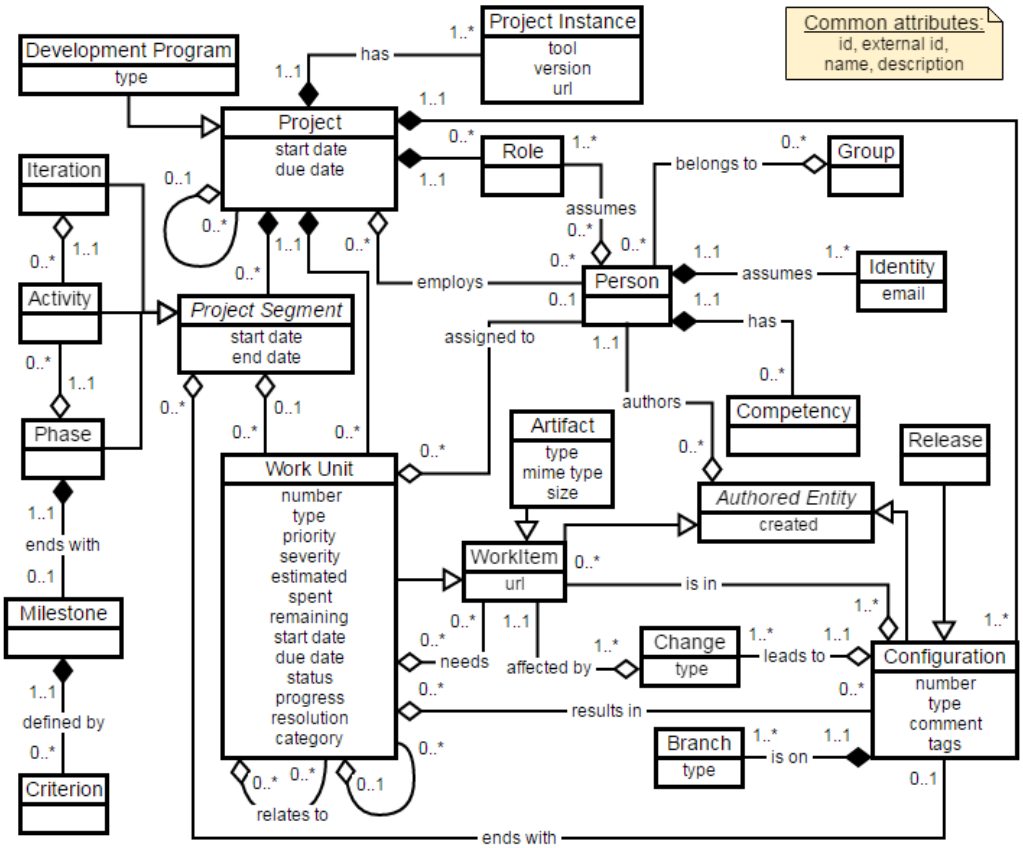
Součástí architektury je také tzv. **Tuner**, který je jakýmsi prostředníkem mezi databází s daty a webovým rozhraním. Tato část by měla pomáhat s odstraňováním nekonzistence v databázi za pomoci uživatelské úpravy urči-

tých aspektů dat, které jsou pro tuto editaci vhodné. Implementace Tuneru a navazujících služeb (rozšíření webového rozhraní, které umožňuje editaci projektu za cílem konzistence dat) je hlavním předmětem této práce.

2.3.2 Metamodel nástroje SPADe

Pro aplikaci SPADe byl implementován unifikovaný datový model, který umožňuje uložení projektových dat z různých ALM nástrojů. Kromě použití různých nástrojů se jednotlivé projekty mohou lišit také v použité metodice vývoje, na čež je datový model připraven.

Návrh modelu, který lze vidět na obrázku 2.5, byl sestaven z průniku entitních tříd vybraných nástrojů a dále musel být doplněn o jejich další specifické jednotlivosti, aby bylo možné projekty komplexněji analyzovat. Některé entity jsou namapované přímo z ALM nástrojů, jiné potřebují dodatečnou analýzu pro jejich správnou interpretaci (např. rozpoznání fáze od iterace). Vznikl tak jednotný model dat, ve kterém je možné analyzovat parametry pro anti-vzory [7].



Obrázek 2.5: Doménový metamodel nástroje SPADe [9]

Výsledná databáze vytvořená na základě modelu uvedeném v obrázku číslo 2.5 má dohromady 85 tabulek (včetně pohledů) a její schéma je možné si prohlédnout v příloze D.

Nyní budou stručně popsány entity uvedeného modelu, přičemž ty, které nejsou svým názvem či uvedeným popisem jednoznačně pochopitelné budou v následující sekci popsány podrobněji.

Základním konceptem modelu je (**Project**), který představuje konkrétní softwarový projekt v databázi. Z této entity pak vycházejí i entity další. Data o projektech jsou ve většině případů dolována z více ALM nástrojů či jiných služeb, a proto je definována entita (**Project Instance**) představující instanci daného projektu v konkrétním nástroji. Tyto entity kromě jiných informací propojují některá specifická data pro daný nástroj (např. výčtové typy). Na projektech samozřejmě pracují lidé (**Person**), kteří mohou mít definovány role (**Role**) a patřit do nějaké skupiny (**Group**). Každá osoba má své identity (**Identity**), které používá v rámci jednotlivých instancí projektu a může disponovat určitými kompetencemi (**Competency**).

Důležitou entitou je (**Work Item**), ze které dědí některé další entity. Může představovat buď nějaký specifický pracovní úkol (ticket; **Work Unit**) s detailním popisem, typem, kategorií, apod. nebo výsledek či vstup nějaké práce, zvaný artefakt (**Artifact**), vzniklý v rámci projektu nějakou konfigurací.

V rámci vývoje softwarového projektu dochází k vytváření různých verzí, tzv. konfigurací (**Configuration**), které jsou definovány nějakou změnou určitých pracovních položek v projektu (**Change**) popřípadě releasem (**Release**). Aby bylo možné rozpoznat v jaké větvi se konfigurace nachází, je definována entita (**Branch**).

Jak již bylo zmíněno, projekt je možné dělit na menší části na základě použité metodiky vývoje, viz sekce s metodikami 2.1.2. Zatímco sekvenční metodiky používají fáze definované entitou (**Phase**) často zakončené nějakým milníkem (**Milestone**) s definovanými kritérii (**Criterion**), iterativní a agilní metodiky používají iterace či sprinty (**Iteration**). Skupina úkolů se společným cílem je definována aktivitou (**Activity**), která definuje téma těchto úkolů. Jedním z využití aktivit je modelování disciplín.

2.3.3 Definice vybraných entit

V této sekci jsou popsány některé entity zanesené v modelu, které svým názvem nebo popisem v předchozí kapitole dostatečně intuitivně nevysvětlují svou podstatu.

Entita **Project instance** reprezentuje jednu instanci projektu v konkrétním ALM nástroji, přičemž jeden projekt může používat více těchto nástrojů. Instance projektů jsou přiřazeny k jednomu konkrétnímu projektu, pod který přísluší. Entita **Identity** shromažďuje všechny identity lidí podílejících se na některém projektu. Tyto identity jsou pak použity v jednotlivých instancích projektu, např. pro přihlašování se k ALM nástrojům nebo jiným aplikacím. Každá osoba může mít více identit a to i v rámci jednoho nástroje, kde představuje profily, které jsou pak sdruženy pod danou osobou.

V každém projektu vznikají úkoly a přímo využitelné produkty práce. Entita **Work Item** představuje pracovní položky projektu a je rodičovskou třídou pro další entity, které více specifikují tento obecný koncept. Jednou z nich je entita **Work Unit** definující konkrétní ticket (úkol) v projektu nebo entita **Artifact**, která představuje výsledek nějaké práce, jak bylo definováno v sekci 2.1.1. Zároveň změna pracovních položek, provedená stejnou osobou v jednu chvíli, iniciuje vytvoření nové konfigurace definované entitou **Configuration**. I tato entita může podléhat změnám, např. komentováním commitů, a je tedy rovněž oděděna od entity **Work Item**.

3 Návrh řešení

Jak bylo popsáno v předchozí kapitole, databáze projektu SPADe je zatížena nekonzistencemi, které je potřeba vyřešit uživatelským zásahem do určitých aspektů projektových dat. Ideálním způsobem jak dosáhnout tohoto cíle je implementovat uživatelské rozhraní umožňující editaci databázových entit způsobujících nekonzistenci. V rámci nástrojové sady SPADe je implementováno webové rozhraní, které zobrazuje výsledky detekce anti-vzorů. Zde se nabízí možnost tento nástroj nazvaný jako **Web Interface** rozšířit o výše zmíněné funkce, aby bylo možné pracovat v rámci jednoho rozhraní.

V této kapitole je popsána aplikace Web Interface, která bude v rámci této práce rozšiřována a navržen způsob, jakým bude uživateli umožněno vyřešit vzniklé nekonzistence v databázi.

3.1 Aplikace Web Interface

Aplikace Web Interface je součástí nástroje SPADe a zprostředkovává uživatelské rozhraní pro detekci vybraných anti-vzorů na jednotlivých projektech v databázi. Byla vytvořena v rámci diplomové práce, která se zabývala analýzou prítomnosti anti-vzorů v datech nástrojů pro řízení projektů [14]. Dále byla aplikace rozšířena na základě zpracování bakalářské práce, která rozšiřovala množinu detekovaných anti-vzorů [16].

Po rozšíření této aplikace bude výsledný nástroj poskytovat včetně podpůrného uživatelského rozhraní také funkce spojené s editací vybraných aspektů databáze.

Pro implementaci aplikace bylo využito programovacího jazyku Java 11 za použití open-source frameworku Spring Boot 2.4.4. Uživatelské rozhraní pak bylo naimplementováno pomocí šablonovacího systému Thymeleaf 3.0.12. Komunikace s MySQL 8.0.23 databází, která byla použita, je zajištěna konektorem Java Database Connectivity (JDBC).

3.1.1 Výchozí funkce aplikace

Aplikace Web Interace ve výchozím stavu funguje na principech, které jsou popsány v následujících odstavcích.

Uživatel komunikuje s aplikací přes uživatelské rozhraní, ve kterém definuje jaké detekce se budou provádět na vybrané množině projektových dat.

Požadavky uživatele pak zprácovává kontrolér, který komunikuje s ostatními moduly aplikace.

Aby bylo možné detekovat vybrané anti-vzory, je potřeba projektová data získat z datového skladu. K tomu slouží modul pro komunikaci s tímto skladem, který funguje na principu parametrizovaných SQL dotazů. Ty jsou načítány ze souboru pomocí příslušného modulu. Horní část diagramu je určena pro načítání údajů o anti-vzorech a nastavení prahových hodnot pomocí připravených konfigurací.

Získané výsledky z databáze se pak zpracovávají v modulu pro detekci anti-vzorů a jsou zpětně kontrolerem zobrazovány uživateli.

3.1.2 Balíky tříd

Celková struktura adresářů a souborů projektu je zobrazena v tabulce 3.1. Znak ∗ nahrazuje cestu `src/main` a zkratka `res` označuje složku `resources`.

Adresář	**Obsah**
∗/java	zdrojové kódy aplikace
∗/res/application.properties	nastavení aplikace
∗/webapp/res/js	javascriptové soubory
∗/webapp/WEB-INF/templates	html šablony uživatelského rozhraní
db	skripty pro nastavení databáze
pom.xml	soubor pro sestavení aplikace
docker-compose.yml	soubor pro spuštění aplikace v Dockeru

Tabulka 3.1: Struktura projektu aplikace

Třídy projektu jsou samozřejmě rozděleny do balíků podle jejich funkcionality. V hlavním balíku `cz.zcu.fav.kiv.antipatterndetectionapp` jsou uloženy specializovanější adresáře a je zde také definována hlavní třída aplikace či třída s konstantami. Vnořené balíky jsou popsány níže.

- Balík `controller` obsahuje pouze jedinou třídu představující kontroler celé aplikace. Její podstatou je získávání uživatelských vstupů, zpracování výsledků a předání těchto dat do dalších částí aplikace, kde se zobrazí uživateli.
- Balík `detecting` obsahuje všechny potřebné třídy pro detekci vybraných anti-vzorů a připojení k datovému skladu SPADe.
- Balík `exception` je balík obsahující jednu třídu, která obstarává zachytávání chyb v průběhu aplikace.

- Balík `model` obsahuje modelové třídy aplikace, představující entity v databázi nebo přepravky pro předávní dat v rámci komponent aplikace.
- Balík `repository` je balík s komponentami, které se starají o načítání a ukládání jednotlivých SQL příkazů a inicializaci detektorů anti-vzorů.
- Balík `service` je balíkem, který obsahuje třídy pracující s instancemi modelových tříd.
- Balík `spring` obsahuje konfiguraci použitého frameworku Spring, jsou zde nastaveny pozice potřebných složek a aplikační vlastnosti.
- Balík `utils` je posledním balíkem se třídami, které obsahují statické metody a mohou být proto použity na různých místech v aplikaci.

Pro rozšíření stávající aplikace o funkce spojené s editací určitých aspektů projektových dat bude určitě potřeba rozšířit balík `model` o další třídy entit z databáze a k nim navazující servisní třídy a repozitáře v balících `service` a `repository`. Aby bylo možné definovat nové části uživatelského rozhraní a reakce na požadavky uživatele, bude také potřeba rozšířit obsah balíku `controller` a složku se soubory šablon. Určitě nebude nutné zasahovat do balíků obsluhující detekci, protože ta není součástí práce.

3.2 Vybrané aspekty k uživatelské úpravě

Aby bylo možné implementovat požadované funkce, byly v rámci bakalářské práce vytipovány databázové tabulky, jejichž instance jsou zdrojem nekonzistence dat nebo je vhodné umožnit uživateli jejich editaci. Zdrojem nekonzistence mohou být především z důvodu nutnosti použití další analýzy při dolování dat z ALM nástrojů pro identifikaci jejich příslušnosti k databázové tabulce. V následujících sekcích je popsáno, proč byly vybrány konkrétní entity a navržen způsob jakým by šlo zajistit nápravu nekonzistence nebo umožnit jejich editaci.

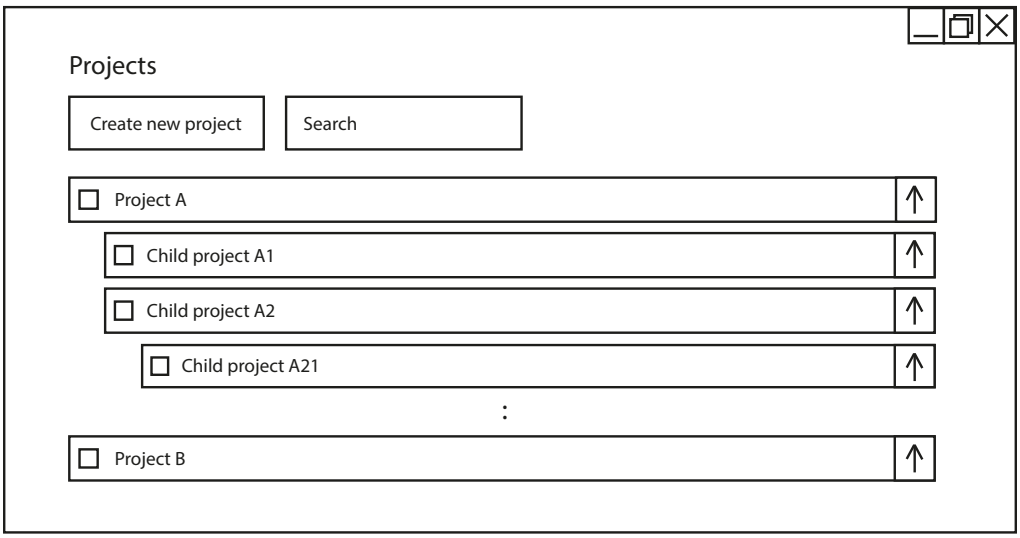
3.2.1 Projekty

Databáze nástroje SPADe obsahuje větší množství projektů, které mezi sebou mohou mít nějaký hierarchický vztah (projekt se skládá z jiných menších projektů a ty mu jsou podřízeny). Každý projekt tedy může být součástí většího celku, čímž se vytváří stromová struktura projektů. Jelikož může při

těžení dat dojít k chybné rekonstrukci hierarchie nebo ji samotný nástroj neumožňuje definovat, je nutné dovolit uživateli tyto vztahy nadřízenosti a podřízenosti vytvořit či upravit a zároveň je vhodným způsobem zobrazit. To je důležité především z toho důvodu, aby bylo možné provádět analýzy nad nadřazenými projekty nebo porovnávat výsledky těchto analýz u sourozeneckých projektů. Další část návrhu spočívá v možnosti vytvoření úplně nového projektu, který by sloužil jako kontejner pro další projekty a stal by se pro ně tzv. 'nadprojektem'. Tím by uživatel získal větší kontrolu nad logickým uspořádáním projektů a zároveň měl okamžitý přehled o tom, které projekty spolu souvisí. Pro uživatele je důležité porozumět existenci projektů a vztahům mezi nimi i z toho důvodu, že všechny následující případy užití jsou podmíněny výběrem projektu, který bude chtít uživatel upravovat.

Návrh uživatelského rozhraní

Návrh uživatelského rozhraní pro tento use case vychází z předpokladu, že se bude jednat o několik stromových struktur, jejichž kořenem budou vždy nejvýše postavené projekty, ty bez rodiče. Při návrhu je také potřeba vzít v úvahu, že každý projekt může být podřízen pouze jednomu nadprojektu a nesmí se jednat o ten samý projekt (projekt nemůže být rodičem sám pro sebe).



Obrázek 3.1: Návrh rozložení komponent pro editaci projektů

Navržené uživatelské rozhraní se skládá z manipulačního panelu a datové části. V panelu se vyskytuje tlačítko *Create new superproject* umožňující vytvoření nového nadprojektu pro všechny již vybrané projekty. Datová část

pak vizualizuje hierarchii všech projektů, kde každý řádek obsahuje informace právě o jednom projektu. Na každém řádku je navrženo v levé části zaškrtávací pole, které bude sloužit pro vybrání projektů, se kterými chce uživatel manipulovat. V pravé části se pak vyskytuje tlačítko, které vybrané projekty podřadí pod daný projekt. Návrh lze vidět na obrázku 3.1.

Připomínky k implementaci

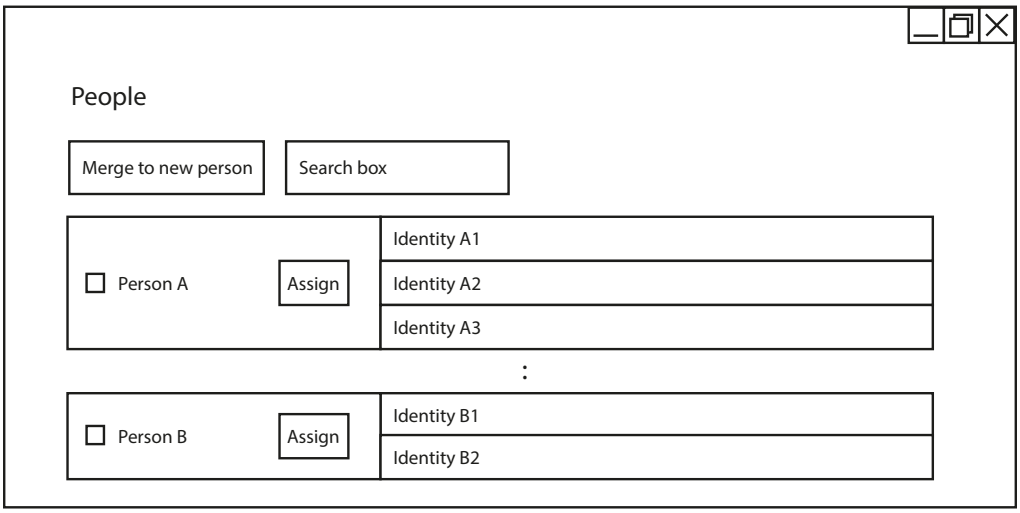
Projekty jsou mapovány z nástrojů téměř napřímo a nepotřebují žádný další zásah. Zdrojem nekonzistence tedy není samotná entita, ale její vztah k jiným entitám stejné třídy. Z tohoto důvodu se bude největší část implementace zaobírat změnou těchto vztahů, kontrolou validity požadovaných vztahů a vykreslením hierarchické struktury projektů.

3.2.2 Osoby

Na projektech se podílí osoby vlastnící jednu či více identit, které slouží pro přístup k projektovým nástrojům. Při dolování projektových dat je snaha o porovnání uživatelských jmen a emailů, ale i tak nelze vždy zaručit, že budou všechny identity sloučeny do jediné osoby. Může tak vzniknout duplicita osob (které budou mít odlišné jméno) a budou obsahovat jiné identity, které by ale měly být sloučeny. Z tohoto důvodů je pro uživatele výhodné, aby mohl více osob s jejich identitami sloučit pouze do jediné osoby. Návrh rozhraní a funkčnosti tyto požadavky zpracovává.

Návrh uživatelského rozhraní

Ve vizualizaci je nutné zobrazit vlastnictví identit nějakou osobou. Nejvíce intuitivní možností je tabulka, která by v jednotlivých řádcích obsahovala informace o identitách, přičemž jeden ze sloupců by obsahoval informaci o osobě a byl by sdílený přes všechny řádky jejích identit. Aby bylo možné slučovat osoby, je v každém řádku zaškrtávací pole, které zahrne danou osobu a všechny její identity do výběru. Pro přiřazení množiny identit, patřících k vybraným osobám, k jiné osobě se bude moci využít buď tlačítko u konkrétní cílové osoby nebo dialogové okno s formulářem vyvolané tlačítkem *Merge selected to new person*. Dialogové okno by nabízelo několik možností jak novou osobu pojmenovat. První z nich by umožňovalo zapsání jména v textové podobě, druhou možností by bylo pojmenování nové osoby podle jména jedné ze zvolených identit a třetí pak výběr jména na základě jmen vybraných osob patřící k identitám.



Obrázek 3.2: Návrh rozložení komponent pro editaci identit a osob

Aby bylo možné zorientovat se ve velkém množství osob a identit, mělo by být v horní části stránky vyhledávací pole, které by umožnilo vyfiltrovat požadované záznamy. Návrh lze vidět na obrázku 3.2.

Připomínky k implementaci

Jak již bylo zmíněno, při mapování může dojít k tomu, že některé osoby mají v databázi více záznamů s jinými identitami a ty je potřeba sloučit. V databázových tabulkách jsou relace mezi ostatními entitami definovány přes osobu, nikoliv přes identitu. Kvůli tomu není možné určit příslušnost úkolů a akcí k jednotlivým identitám a není tedy vhodné přeřazovat identity mezi osobami. Hlavním úkolem při implementaci tohoto případu užití tedy bude zajistit správný přesun a sloučení osob, propagovat tuto změny do ostatních navazujících atributů v databázi a zajistit přehlednou vizualizaci v uživatelském rozhraní.

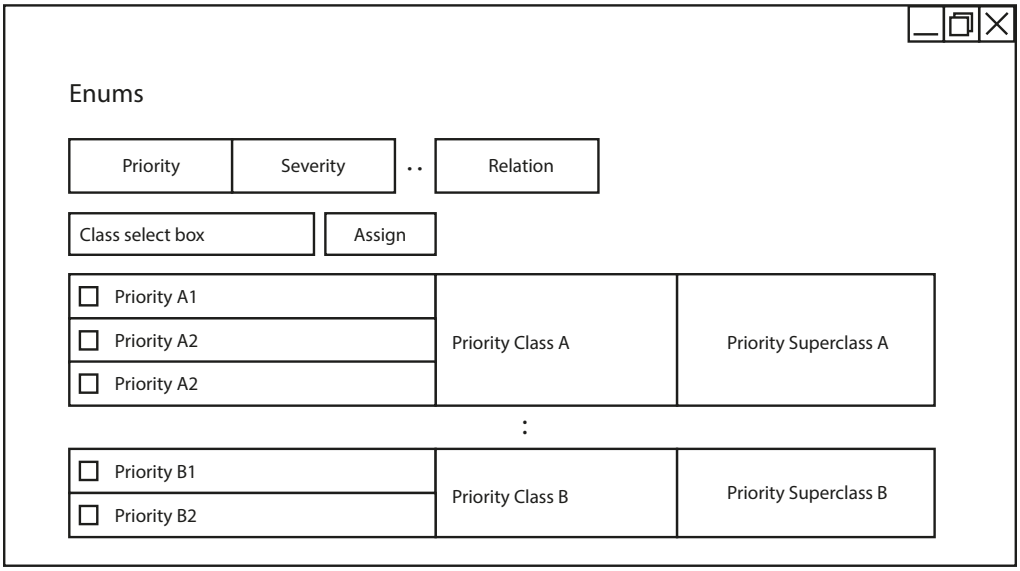
3.2.3 Výčtové typy

Pracovní úkoly a další části projektu jsou definovány pomocí určitých atributů, které mají formu výčtových typů (např. priorita, závažnost, role). Každý ALM nástroj používá jinou sadu hodnot pro atributy těchto typů a některé dokonce umožňují jejich uživatelskou customizaci. V unifikovaném modelu databáze má každý výčtový typ přiřazen klasifikační třídu, aby bylo možné alespoň hrubě kategorizovat hodnoty výčtových typů pro jednotné analýzy. Automatická zpracování při dolování dat z databáze však může způsobit, že jsou výčtové typy namapovány na klasifikační třídy nesprávně, a

proto je vhodné vyřešit tyto vzniklé nekonzistence zásahem uživatele. Téměř každý výčtový typ má také definovanou tzv. supetřídu, která je pevně namapována na klasifikační třídy a zařazuje je do obecnějšího rámce (vyjma výčtového typu `WuType`)

Návrh uživatelského rozhraní

Existuje několik výčtových typů, které jsou v různých částech projektu použity k popsání zadaných úkolů nebo vztahů. Aby bylo možné upravovat všechny výčtové typy v jednom rozhraní, obsahuje návrh princip záložek v záhlaví stránky. Každá záložka pak představuje jeden druh výčtových typů (např. priorita, závažnost, stav úkolu atd.), mezi kterými lze přepínat a upravovat tak všechny existující typy použité v projektech v rámci jednoho uživatelského rozhraní.



Obrázek 3.3: Návrh rozložení komponent pro editaci výčtových typů

Každá záložka by obsahovala seznam výčtových typů dané kategorie a její namapování na klasifikační třídu a supertřídu, které jsou definovány databází. Aby bylo možné uživatelsky měnit příslušnost ke třídě, je u každého výčtového typu zaškrtávací pole sloužící k selekci a následnému přiřazení. Samotné přiřazení by pak proběhlo vybráním klasifikační třídy ze seznamu a potvrzením změn. Návrh rozložení lze vidět na obrázku 3.3.

Mapování výčtových typů z ALM nástrojů

Jelikož každý ALM nástroj používá pro klasifikaci některých atributů rozličnou stupnici výčtových typů a zároveň nástroj SPADe doluje data z různých ALM nástrojů, musela být definována jednotná stupnice pro všechny výčtové typy, která je pak použita v nástroji SPADe.

Atributy výčtových typů jsou především použity ve třídě **Work Unit** a jedná se o entity představující řešení úkolu, jeho prioritu a závažnost, stav, ve kterém se úkol nachází, a typ úkolu. Dále se výčtové typy využijí při definici vztahů mezi pracovními položkami či při kategorizaci rolí, které se přiřazují lidem v projektu. Odpovídajícími tabulkami v databázi jsou **resolution**, **priority**, **severity**, **status**, **wu_type**, **role** a **relation**. V těchto tabulkách je vždy uložen přesný název výčtového typ z ALM nástroje v atributu `name` a v přidružené tabulce je uvedená klasifikační třída, do které výčtový typ patří. Ta byla přiřazena za pomoci jednoduché heuristické analýzy před uložením do databáze SPADe. U většiny výčtových typů je také uvedena nadtřída, která je pevně namapována na samotnou třídu a zařazuje tak typ do více obecnější třídy. Příkladem je stav úkolu, který může být rozpracovaný (`INPROGRESS`) ale je vždy ještě namapován na nadtřídu otevřený (`OPEN`).

V následujících tabulkách 3.2 – 3.8 je popsána jednotná stupnice klasifikačních tříd, která byla použita pro jednotlivé výčtové typy, reflektuje používané značení v ALM nástrojích a umožňuje jednotně ukládat tyto entity do databáze SPADe.

Výčtový typ Resolution slouží k označení, jakým způsobem byl pracovní úkol vyřešen. Třídy představují jednotlivá řešení – např. nepřiřazený, opravený, duplicitní, nebude opravováno. Nadtřídy popisují, zda je úkol vyřešen či stále otevřen pro řešení, popřípadě ještě nikomu nepřiřazen.

Třída	**Nadtřída**
UNASSIGNED	UNASSIGNED
DUPLICATE	FINISHED
INVALID	FINISHED
WONTFIX	FINISHED
WORKSASDESIGNED	FINISHED
FIXED	FINISHED
FINISHED	FINISHED
INCOMPLETE	UNFINISHED
WORKSFORME	UNFINISHED
UNFINISHED	UNFINISHED

Tabulka 3.2: Mapování výčtových typů atributu Resolution

Typ Priority klasifikuje úkoly v závilosti na tom, v jakém pořadí budou vyřešeny. Nadtřídy umožňují obecnější a méně specifický pohled na priority.

Třída	Nadtřída
UNASSIGNED	UNASSIGNED
LOWEST	LOW
LOW	LOW
NORMAL	NORMAL
HIGH	HIGH
HIGHEST	HIGH

Tabulka 3.3: Mapování výčtových typů atributu Priority

Závažnost určuje o jak závažný problém jde v rámci opravy vytvářeného softwaru. Nadtřída klasifikuje chyby s menší granularitou a poskytuje na první pohled lepší orientaci.

Třída	Nadtřída
UNASSIGNED	UNASSIGNED
TRIVIAL	MINOR
MINOR	MINOR
NORMAL	NORMAL
MAJOR	MAJOR
CRITICAL	MAJOR

Tabulka 3.4: Mapování výčtových typů atributu Severity

Výčtový typ Status přidává k úkolům definici stádia jeho životního cyklu . Nadtřídy pak shrnují zda v je v tomto stavu úkol rozpracovaný nebo uzavřený.

Třída	Nadtřída
UNASSIGNED	UNASSIGNED
NEW	OPEN
OPEN	OPEN
ACCEPTED	OPEN
INPROGRESS	OPEN
RESOLVED	OPEN
VERIFIED	OPEN
DONE	CLOSED
CLOSED	CLOSED
INVALID	CLOSED
DELETED	CLOSED

Tabulka 3.5: Mapování výčtových typů atributu Status

Výčtový typ WuType definuje o jaký typ úkolu se jedná – oprava chyby (`BUG`), vylepšení (`ENHANCEMENT`), nová vlastnost (`FEATURE`), úkol (`TASK`). Mapování na nadtřídy zde není definováno.

Výčtový typ Role definuje roli uživatele, kterou v projektu zastává. Nadtřída pak klasifikuje tyto třídy do skupin podle vztahu k projektu.

Výčtový typ Relation definuje vztahy mezi jednotlivými úkoly. Nadtřídy pak specifikují typ tohoto vztahu – např. hierarchický, temporální.

Třída
UNASSIGNED
BUG
ENHANCEMENT
FEATURE
TASK

Tabulka 3.6: Mapování výčtových typů atributu WuType

Třída	**Nadtřída**
UNASSIGNED	UNASSIGNED
NONMEMBER	NONMEMBER
MENTOR	STAKEHOLDER
STAKEHOLDER	STAKEHOLDER
PROJECTMANAGER	MANAGEMENT
TEAMMEMBER	TEAMEMBER
ANALYST	TEAMEMBER
DESIGNER	TEAMEMBER
DEVELOPER	TEAMEMBER
TESTER	TEAMEMBER
DOCUMENTER	TEAMEMBER

Tabulka 3.7: Mapování výčtových typů atributu Role

Třída	**Nadtřída**
UNASSIGNED	UNASSIGNED
DUPLICATES	SIMILARITY
DUPLICATEDBY	SIMILARITY
BLOCKS	TEMPORAL
BLOCKEDBY	TEMPORAL
RELATESTO	GENERAL
PRECEDES	TEMPORAL
FOLLOWS	TEMPORAL
COPIEDFROM	SIMILARITY
COPIEDBY	SIMILARITY
CHILDOF	HIERARCHICAL
PARENTOF	HIERARCHICAL
CAUSES	CAUSAL
CAUSEDBY	CAUSAL
RESOLVES	CAUSAL
RESOLVEDBY	CAUSAL

Tabulka 3.8: Mapování výčtových typů atributu Relation

Poznámky k implementaci

Základem pro implementaci bude vytvořit funkční a přehlednou vizualizaci výčtových typů s ohledem na jejich mapování na třídy a nadtřídy. Samotná změna příslušnosti typu ke třídě bude především otázkou validace vstupních dat a změny příslušného atributu u entity v databázi.

3.2.4 Segmenty projektu

Jak bylo řečeno v sekci 2.1.2, každý projekt v databázi SPADe může být řízen podle jiné metodiky. Některé mají definovány fáze, ve kterých vývoj probíhá, v jiných probíhají iterace a někde může dojít k určité kombinaci přístupů. V databázi je také definována entita představující kategorie úkolů. Ta umožňuje úkoly kategorizovat pomocí dalších atributů (definovaných nástrojem či vývojovým týmem) a někdy také může představovat iterace, fáze či aktivity z důvodu pohodlnosti vývojového týmu nebo protože použitý nástroj nedokáže tyto aspekty definovat jiným způsobem. Při dolování dat z ALM nástrojů se z těchto důvodů neprovádí analýza rozlišující příslušnost entity k databázové tabulce, a je toto rozhodnutí ponecháno na uživateli. Z těchto důvodů by bylo vhodné v rámci aplikace SPADe umožnit změnu iterace uložené v databázi na fáze a opačně. Uživatel by měl dostat také díky vyvíjenému nástroji možnost manuálně změnit příslušnost zmíněných entit zmíněných do správné tabulky se zachováním dalších vazeb v databázi. Třetí návrh týkající se segmentů projektu by měl umožnit hromadné přiřazování projektových aktivit k pracovním úkolům.

V následujících sekcích jsou popsány jednotlivé případy užití týkající se segmentů projektu. U každé je uveden návrh uživatelského rozhraní a připomínky, které je nutné vnímat při implementaci.

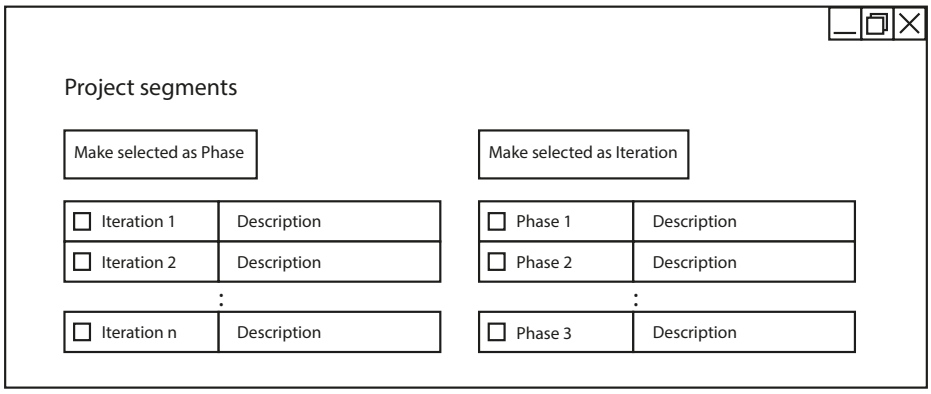
Fáze a iterace

Aby bylo možné zaměnit mezi sebou fázi a iteraci, je nutné uživateli zobrazit záznamy z obou těchto tabulek. Proto je v návrhu uživatelské rozhraní rozděleno na dvě poloviny – jedna je určena fázím a druhá iteracím. Obě poloviny se skládají z tabulky, která obsahuje informace o daných entitách. V případě, že chce uživatel provést záměnu, vybere vyhovující entity a stiskne tlačítko nad tabulkou. Návrh rozložení lze vidět na obrázku 3.4.

V datovém modelu mají fáze a iterace téměř stejné atributy a tudíž nebude problém transformovat tyto entity mezi sebou. Velká část implementace se bude zaobírat důsledky této transformace v jiných databázových tabulkách, nejvíce pak ve entitě WorkUnit, která iterace a fáze využívá a bude tedy třeba vykonanou změnu promítnout i do instancí této entity.

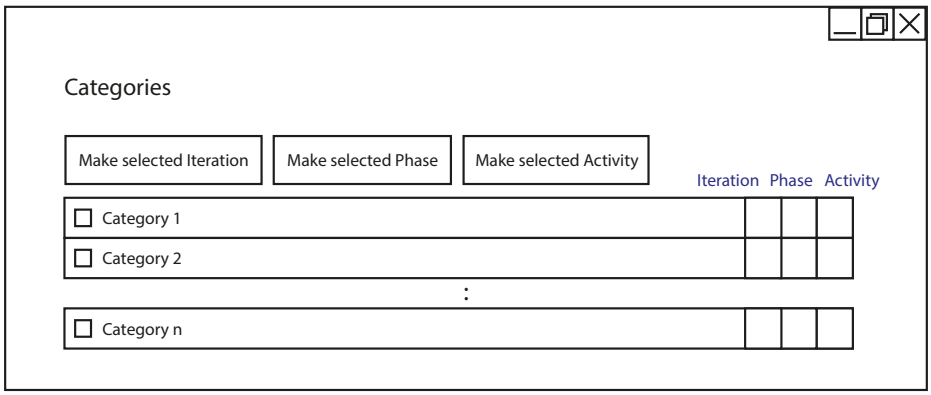
Kategorie a jejich transformace

V návrhu jsou kategorie uspořádány do tabulky, ve které má každá záznam. Řádek s kategorií může být vybrán pomocí zaškrtávacího pole a v horní části přímo přiřazen do jiné tabulky pomocí tlačítek. Prvek *Make selected*



Obrázek 3.4: Návrh rozložení komponent pro výměnu iterací a fází

as Iteration změní vybrané kategorie na iterace. Obdobná akce se vykoná i při stiknutí sousedních tlačítek, které jsou určeny pro fáze a aktivity. Další možností přeřazení jednoho konkrétního záznamu je stisknutí tlačítku u dané kategorie, které symbolem odpovídá doměné, kam chce uživatel kategorii zařadit. Návrh rozložení lze vidět na obrázku 3.5.



Obrázek 3.5: Návrh rozložení komponent pro změnu kategorie na jiné entity

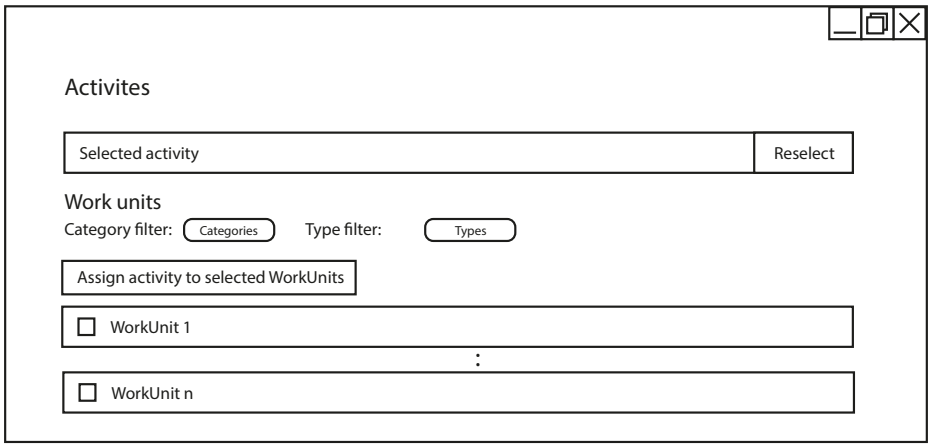
Jak již bylo řečeno v předchozím případu užití, velkou část implementace bude potřeba věnovat nalezení všech vazeb s entitami, které jsou předmětem změny a provést konzistentní úpravy dotčených atributů.

Aktivity

Aby bylo možné přiřadit nějaké podmnožině pracovních úkolů určitou aktivitu, musí být nejdříve vybrána aktivita, která bude přidělena. Uživateli tedy bude nabídnut seznam všech aktivit projektu v podobě tabulky s detailními informacemi a po vybrání jedné z nich se zobrazí tabulka se všemi

pracovními úkoly. Úkoly bude moci uživatel filtrovat v závislosti na kategorii a typu. Z určené podmnožiny pak lze ručně vybrat jednotlivé úkoly nebo označit všechny zobrazené. Pomocí tlačítka pak dojde k přiřazení aktivity k úkolům.

V rámci implementace bude zřejmě nejsložitější částí vymyslet konkrétní realizaci filtrování jednotlivých úkolů a výběr aktivit. Samotné přiřazení aktivity je už pak pouze změnou atributu.



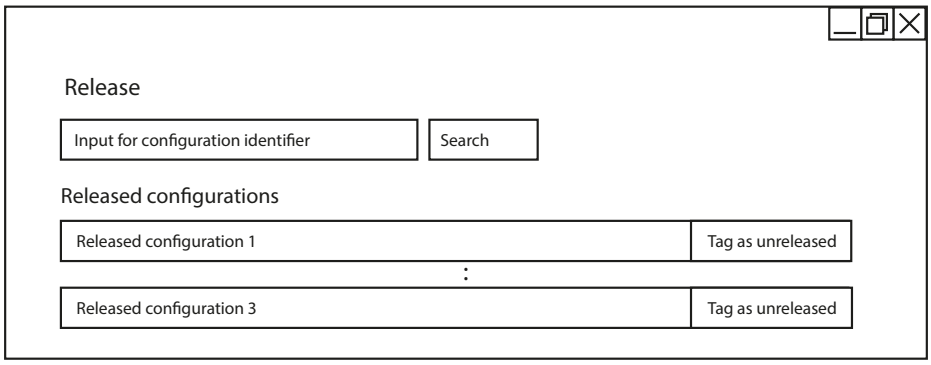
Obrázek 3.6: Návrh rozložení komponent pro nastavení aktivity úkolům

3.2.5 Release

Commitnuté konfigurace mohou být označena za release. Pak se většinou jedná o verze, které jsou stabilní a lze je použít např. pro ukázku zázkaznímkovi. V databázi sady nástrojů SPADe je také definována entity `Tag`, která umožňuje značkování verzí aplikace včetně označení release. V rámci SPADe je vhodné umožnit uživateli „povýšit" tag commitnuté konfigurace na release, protože v rámci dolování dat není vždy jednoznačené, který tag konfigurace označuje právě release (vývojový tým si může označit podobným tagem jakoukoliv konfiguraci).

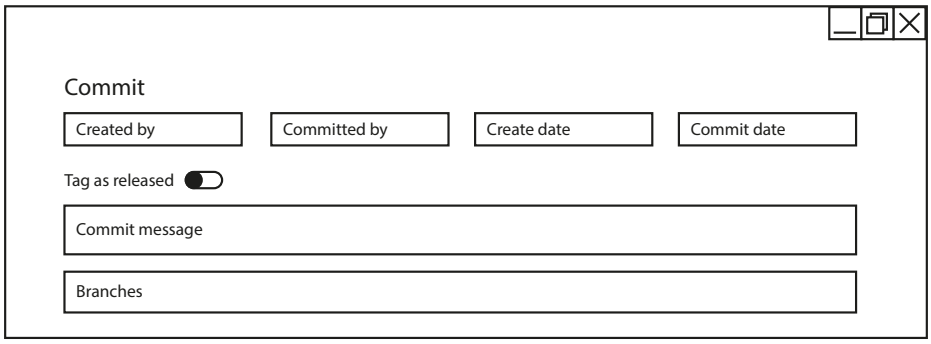
Návrh uživatelského rozhraní

Pro uživatele je z informačního hlediska vhodné zobrazit pro vybraný projekt konfigurace, které již mají nastavený příznak release a poskytnou možnost snadno degradovat release na obyčejný commit. Následně by uživatel mohl na základě identifikátoru vyhledat specifickou konfiguraci, tu si zobrazit se všemi podstatnými informacemi a po uvážení ji označit jako release. Návrh této části rozhraní lze vidět na obrázku 3.7.



Obrázek 3.7: Návrh rozložení komponent pro vyhledaní konfigurací

V tomto rozhraní by uživatel viděl všechny základní informace o konfiguraci a také by zde bylo možné přepínačem změnit příznak označující release. Návrh rozložení komponentů lze vidět na obrázku 3.8.



Obrázek 3.8: Návrh rozložení komponent pro konfigurace

3.3 Omezení implementace a další rozšíření

Implementace navrhnutých funkcionalit je součástí již existující aplikace Web Interface a musí tedy respektovat již použité technologie. Dále musí být dodržena struktura její databáze, což je částečným omezením implementace.

Další funkcionalitou, která může být v budoucnu implementována je rozšíření manipulace s částmi projektů, která by umožňovala transformaci i těch segmentů, jejichž pracovní úkoly mají již stejný typ segmentu definován. Tato funkce nebude v této práci implementována na základě dohody s vedoucím práce. Dalším doporučeným rozšířením je implementace pseudonymizace projektových dat pro účely prezentace výsledků aplikace veřejnosti.

4 Implementace

Tato kapitola popisuje, jakým způsobem byly navržené funkcionality implementovány a jak se změnila struktura původní aplikace.

4.1 Rozšíření aplikace Web Interface

Implementace dbala na původní architekturu aplikace a dodržovala zavedené balíky. Především byly rozšířeny balíky obsahující kontrolery, repozitáře a servisní třídy, aby byla možná implementace nových funkcí. Nové kontrolery jsou uspořádány v podbalíku `management` a třídy, které se zabývají úpravou výčtových typů, jsou v podbalících `enums` různých balíků. U ostatních tříd nebylo vhodné zavádět další změnu hierarchie projektu z důvodu konzistentní struktury aplikace a soudržnému členění tříd. Rozšíření jednotlivých balíků je popsáno v následujícím seznamu:

- Balík `controller` – balík rozšířen o adresář `management`, ve kterém jsou všechny třídy kontrolerů aplikace použité pro implementaci funkcionalit.
- Balík `model` – balík rozšířen o adresář `enums` obsahující modelové třídy výčtových typů a `interfaces` s definovanými rozhraními pro obecnou práci s modelovými třídami. Dále byly přidány třídy, jejichž instance odpovídají záznamům v databázi. Tyto třídy jsou popsány u jednotlivých funkcionalit v následujících kapitolách.
- Balík `repository` – balík rozšířen o adresář `enums` obsahující repozitáře tříd výčtových typů. Další repozitáře byly vytvořeny k jednotlivým modelovým třídám.
- Balík `service` – balík rozšířen o adresář `enums`, který stejně jako v předchozích případech obsahuje servisní třídy pro výčtové typy. Dále jsou v balíku implementovány nové třídy související s vytvořenými modely databáze. Každá modelová třída má jedno servisní rozhraní definující její metody a následně třídu implementující toto rozhraní.
- Balík `utils` – zde byla rozšířena třída `Utils` o jednu metodu, která pracuje s relacemi.

- Složka `resources` – definována v `/src/main/webapp/resources` a byl do ní přidán adresář `management` s javascriptovými soubory, které jsou použité v rámci nových funkcí.
- Složka `templates` - definována cestou z hlavního adresáře `/src/main/webapp/WEB-INF/templates` a byla do ní přidána šablona pro zobrazení hlavní stránky managementu databáze. Dále byla rozšířena o adresář `management`, který obsahuje všechny další šablony. V adresáři `fragments` přibyly soubory `manage-navbar.html` a `project-selection.html`.

V uživatelském rozhraní výsledné aplikaci jsou všechny implementované funkce pod záložkou `Manage`, která je k nalezení v navigačním menu. Mimo popsané změny byly provedeny nezbytné úpravy i v ostatních třídách, které ale sloužili pouze pro integraci nové funkcionality.

Seznam nově vytvořených modelových tříd

Původní aplikace byla rozšířena o tyto modelové třídy, které lze nalézt v balíku `model`:

`Activity`, `Branch`, `Category`, `Commit`, `Configuration`, `CommittedConfiguration`, `ConfigurationPersonRelation`, `Identity`, `Iteration`, `Person`, `Phase`, `Project` (pouze rozšířena), `ProjectInstance`, `Tag`, `WorkItem` a `WorkUnit`.

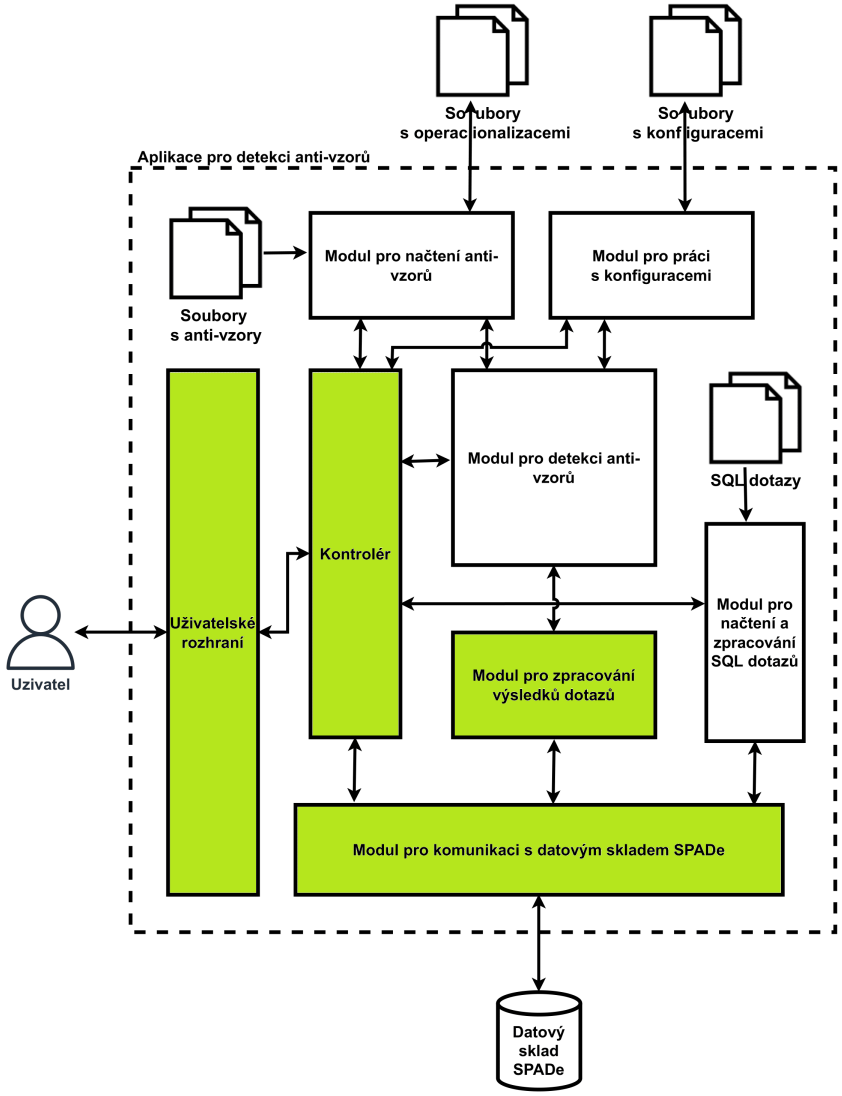
Moduly, které byly v rámci implementace rozšířeny jsou označeny ve struktuře aplikace zelenou barvou.

Změny atributů v databázi

V rámci implementace budou v databázi probíhat změny na úrovni vytváření nových instancí či změnách atributů. Tyto změny jsou znázorněny ve výřezu datového modelu, který obsahuje entity, kterých se změny budou týkat 4.2. Červeně označené entity jsou ty, jejichž instance budou v rámci programu vytvářeny. Zeleně označené atributy pak budou předmětem změny.

4.2 Základní stránka

Aby bylo možné jednoduše přepínat mezi navrhnutými případy užití, byla implementována hlavní stránka (vůči novým funkcionalitám), která bude sloužit jako rozcestník pro všechny funkce. Tuto stránku lze nalézt v menu pod názvem *Manage*. Tato položka má také rozbalovací menu, které nabízí



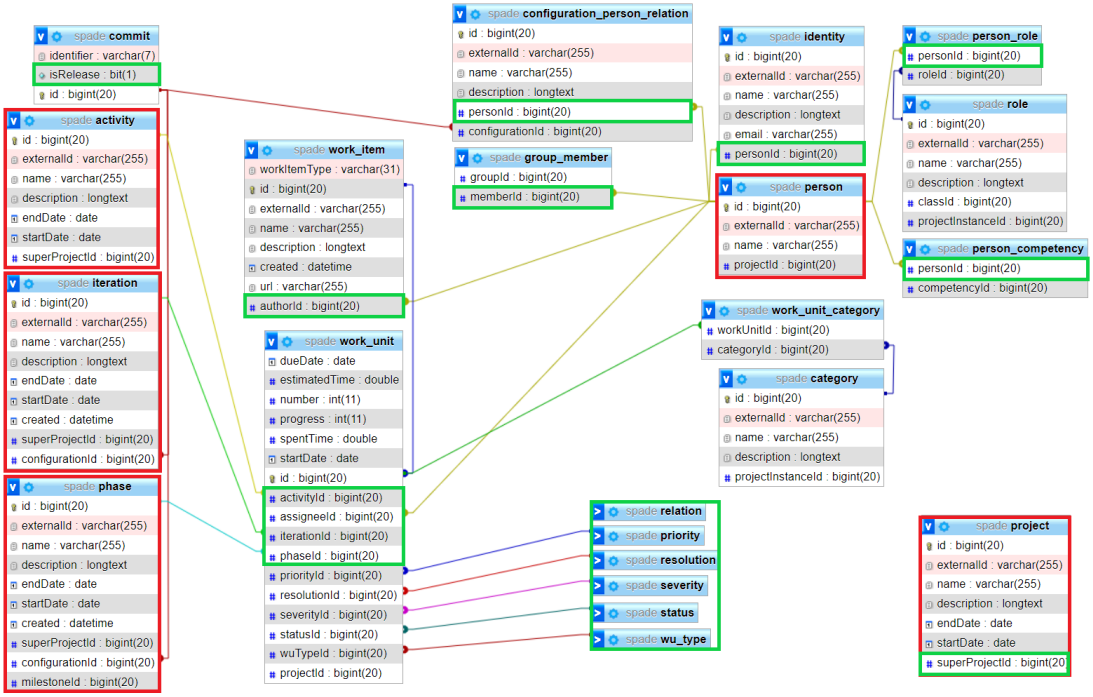
Obrázek 4.1: Rozšířené moduly v aplikaci Web Interface [16]

rychlé přesměrování na požadovanou funkci. Výsledný vzhled uživatelského rozhraní lze vidět na obrázku 4.3.

4.3 Projekty

Implementace funkčního návrhu, který lze vidět na obrázku 3.1 je složena ze dvou částí. První se zabývá sestavením a vizualizací hierarchie projektů představující několik nezávislých stromových struktur a ta druhá pak manipulací s těmito strukturami a přidáváním dalších projektů.

Pro potřeby navržených funkcionalit byla vytvořena třída `ProjectController.java` z balíku `controller/management`, která



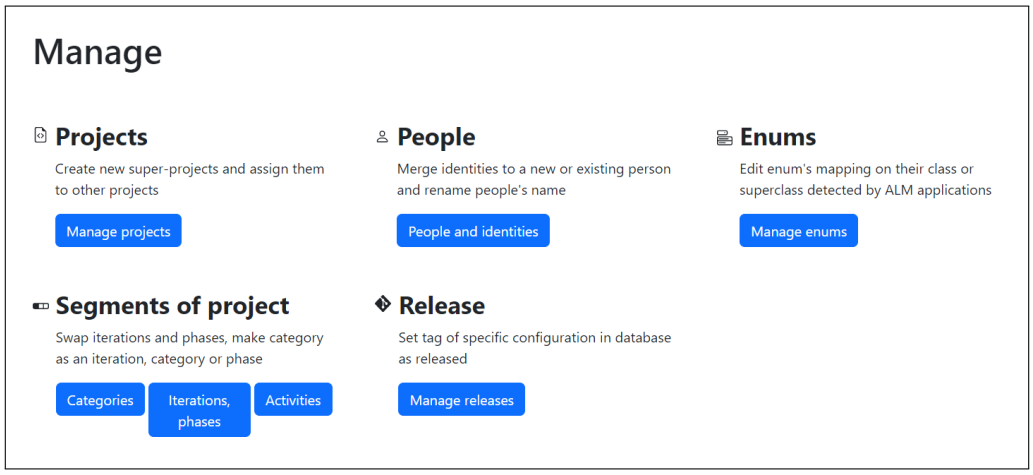
Obrázek 4.2: Změny prováděné v databázi nástroje SPADe

bude zodpovídat za zpracování uživatelských dat, změnu modelu dat a aktualizaci pohledu, který vizualizuje projekty. Tato třída obsahuje čtyři metody, které popsanou funkčnost zajišťují.

4.3.1 Sestavení hierarchie projektů

Aby bylo možné vizualizovat hierarchii projektů, je nutné nejdříve data o projektech získat z databáze. K tomu slouží třída `Project` v balíku `model` reprezentující entitu pro projekty v databázi, která obsahuje potřebné atributy, především pak identifikátor a jméno. Pro napojení entity s databází je využita třída `ProjectRepository` z balíku `repository` a pro manipulaci s tímto repozitářem existuje třída `ProjectService` z balíku `services`.

Kontroler obsahuje metodu `projects(Model model, ...)`, která je namapována na GET metodu HTTP requestu s odpovídající stránkou zobrazující hierarchii projektů. Šablona této stránky je popsána v sekci 4.3.2. Ve zmíněné metodě se ze získaných dat generují datové struktury představující strom, které jsou následně zobrazeny v šabloně. Pro vybudování zmíněných struktur s projekty byla vytvořena v balíku `model` třída `Node`, která představuje jeden vrchol ve stromě. Instance této třídy vlastní odkaz na projekt, který má daný vrchol reprezentovat a kolekci dalších instancí třídy `Node` ob-



Obrázek 4.3: Příklad uživatelského rozhraní rozcestníku

sahující přímé potomky tohoto vrcholu. Vytvořením několika instancí této třídy se postupně může budovat celá hierarchie.

```
1    public class Node {
2        public Project project;
3        public ArrayList<Node> children;
4    }
```

Listing 4.1: Implementace třídy Node

Pomocí servisní třídy projektů jsou z databáze nejdříve získány všechny rodičovské projekty, což jsou projekty, které nemají žádného rodiče a jsou tedy kořenem stromové struktury. Pro každý takový projekt se inicializuje instance třídy `Node`, která bude reprezentovat vrchol ve stromě a pro jeho podřízené projekty se vytvoří nový prázdný seznam. Tímto procesem vznikne několik kořenů různých stromů, kde každý bude finálně představovat jinou skupiny projektů. Dále je pro každý vrchol stromu zavolána metoda `calculate(Project p)`, popsána v 4.2, jejímž úkolem je vytvořit zbytek stromové struktury pomocí rekurzivního volání. V této metodě se opět pomocí repozitáře získá seznam projektů, které jsou přímými potomky aktuálního projektu a všechny jsou přidány do kolekce nadřazeného vrcholu. Pro získání dalších potomků je použita rekurze. Do modelu, který je později využit připravenou šablonou stránky je pak vložen seznam rodičovských instancí třídy `Node`, který je opět rekurzivně procházen a zobrazován v uživatelském rozhraní.

```
1  private ArrayList<Node> calculate(Project p) {
2      ArrayList<Node> nodes = new ArrayList<>();
3      List<Project> projects = projectService.
           getSubordinateProjectsTo(p.getId());
4      for(Project project : projects) {
5          Node n = new Node();
6          n.project = project;
7          n.children = calculate(project);
8          nodes.add(n);
9      }
10     return nodes;
11 }
```

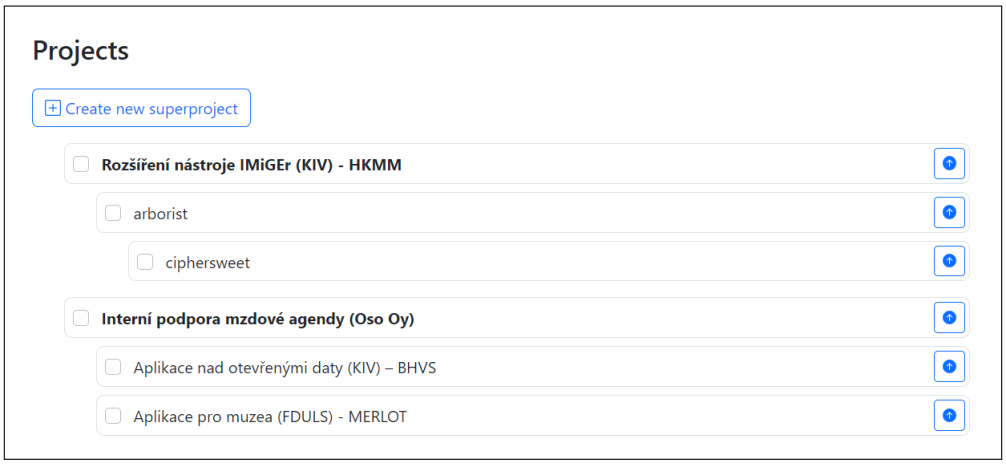
Listing 4.2: Metoda pro sestavení hierarchie projektů

4.3.2 Pohled pro zobrazení projektů uživateli

Výsledný pohled, který uvidí uživatel ve webovém rozhraní je definován v souboru `projects.html` ve složce `templates`. Ten se skládá z formuláře obsahující tlačítko, které otevírá dialogové okno s možností vytvořit nový nadprojekt pro již vybrané projekty. Poté následuje segment s vykreslenou hierarchickou strukturou projektů. Pro znázornění hierarchie je použit klasický položkový seznam, který je kaskádovými styly upraven do přehlednější a uživatelsky přívětivější podoby. Jak již bylo zmíněno, definovaná šablona získá z modelu seznam rodičovských vrcholů stromových struktur a následně pro každý takový kořen vygeneruje položku v seznamu. Položka obsahuje zaškrtávací pole pro výběr projektu při následné manipulaci a tlačítko pro přiřazení všech vybraných projektů pod tento projekt. Další položky jsou pak vygenerovány jako nový vnořený seznam pomocí rekurizvního vkládání definovaného fragmentu. Tímto způsobem se vykreslují všechny podřízené projekty, dokud není kolekce s potomky všech vrcholů prázdná. Příklad výsledné vizualizace je zobrazena na obrázku 4.4

4.3.3 Manipulace se stromovou strukturou

Pro změnu struktury musí uživatel nejdříve vybrat projekty, kterým chce změnit rodiče. Pokud je vybrán některý projekt, zablokují se zaškrtávací pole u podřízených projektů, protože ty budou automaticky změněny se svým rodičem v korespondenci se stávající strukturou. Pokud je ve výběru zahrnut alespoň jeden projekt, odblokují se tlačítka u jednotlivých projektů a tlačítko na vytvoření zcela nového. Popsané změny stavů u ovládacích prvků zajišťuje skript `project.js` ve složce `webapp/resources/js/management`. Stisknutím některého z těchto tlačítek dojde k odeslání formuláře do kontroleru, ve kterém je definována metoda `projectChange(Model model,`



Obrázek 4.4: Příklad uživatelského rozhraní projektů

`...`) zpracovávající získána data.

Validace dat

V metodě nejdříve proběhne validace dat, při které se zkontroluje existence vybraných projektů a logický aspekt úkonu (např. uživatel nemůže projekt označit za nadřazený sám sobě nebo nemůže vytvořit smyčku – projekt nemůže být nadřazeným pro svého rodiče).

Vznik cyklu lze ověřit následujícím způsobem. Vytvoří se seznam, který se naplní všemi projekty, které budou změněny (ty vybrané uživatelem i jejich potomky). Poté se otestuje, zda-li tento seznam obsahuje projekt, pod který budou vybrané projekty přiřazeny. Pokud ano, vznikla by smyčka, která není ve stromě žádoucí. Zjednodušeným algoritmem výše popsaného se také testuje, zda uživatel nechce přiřadit stejný projekt sám pod sebe.

Provedení změny hierarchie

Poté co jsou data úspěšně zvalidována, dojde ke změně vytvořené hierarchie. Každému projektu, který byl v uživatelském rozhraní vybrán, se přenastaví atribut `superProjectId`. Jde o atribut, který má každá instance třídy `Project` a který definuje unární vazbu ve smyslu podřízenosti dalšímu projektu. Na základě této operace dojde ke změně příslušnosti celého podstromu, jejímž vrcholem je právě vybraný projekt. Změněné entity se následně uloží do databáze a uživateli je znovu načtena výchozí stránka s již upravenou strukturou.

Vytvoření nového nadprojektu

Kromě přiřazení entit pod již existující projekt, může uživatel vytvořit zcela nový nadprojekt. Uživateli stačí vybrat projekty, které bude chtít přiřadit, stisknout tlačítko v horní části rozhraní, vyplnit jméno nového projektu a vytvořit ho. Stejně jako v předchozím případě se data z formuláře odešlou do zmíněné metody kontroleru.

Pokud je jméno projektu vyplněno, vytvoří se nová instance projektu, která je pomocí servisní třídy a metody `saveProject(Project p)` uložena do databáze. Tato instance má nastaveno pouze jméno. Následně je provedena změna atributu stejným způsobem, který je popsán v sekci výše, ale již není potřeba kontrolovat přítomnost cyklu, který při vzniku nového projektu není možné vytvořit.

4.4 Osoby

Návrh uživatelského rozhraní, které bylo implementováno je popsáno na obrázku 3.2. Pro logickou část tohoto případu užití byla vytvořena třída `PersonController.java` v balíku `controller/management` zodpovídající za manipulaci s uživatelskými daty a propagováním změn do modelu. Další třídy a jejich funkce jsou popsány v následujících podkapitolách.

4.4.1 Výběr editovaného projektu

Aby bylo možné modifikovat aspekty projektů, je nejdříve potřeba vybrat jeden, který bude momentálně uživatelsky upravován. Z tohoto důvodu jsou v metodě, která je namapována na HTTP request metodou GET vybrány všechny projekty z databáze pomocí servisní třídy projektů. Tento seznam je následně uložen do modelu a zobrazen v uživatelském rozhraní jako rozbalovací menu. Po výběru ze seznamu se odešle kontroleru formulář se zvoleným projektem a ten je poté vložen jako atribut do probíhající relace. Tím je zaručeno, že pokud bude chtít uživatel editovat více aspektů projektu (osoby, segmenty, výčtové typy) nebude muset vždy vybírat projekt znovu. Dále se také načte zbytek stránky, který již implementuje konkrétní funkcionalitu. Tento přístup se používá na většině stránek s uživatelským rozhraním.

4.4.2 Vizualizace osob s vazbou na identity

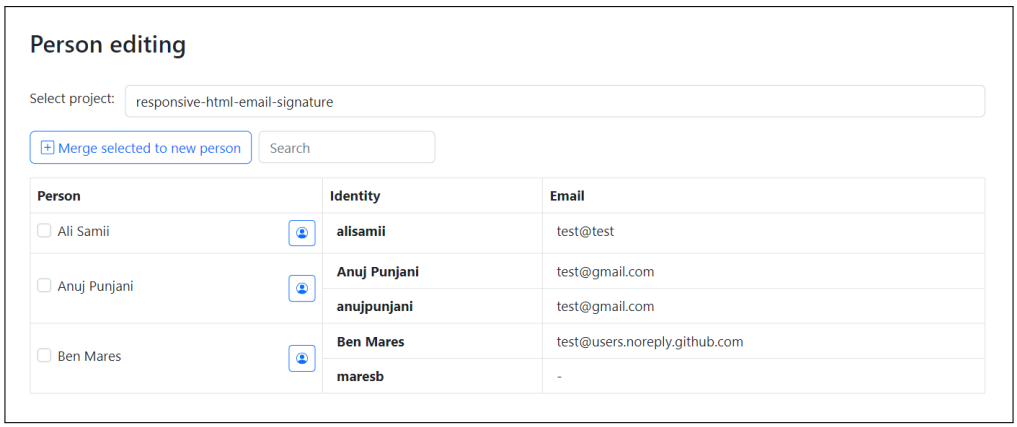
Po vybrání projektu se v metodě `people(Model m, ...)` přidá do modelu atribut se všemi zainteresovanými osobami, které mají nějakou identitu.

Instance třídy `Person` v balíku `model` představují jednotlivé osoby v databázi projektu a k jejich namapování slouží rozhraní `PersonRepository` z balíku `repository`. S instancemi lze poté manipulovat prostřednictvím třídy `PersonServiceImpl`, která poskytuje metody na získání všech osob a úpravu relací, kde figurují osoby.

Každá osoba vlastní nějaké identity, které je potřeba v rámci vizualizace zobrazit, aby měl uživatel přehled o tom, s jakými osoby manipuluje a kterých identit se změna dotkne. Třída z balíku `model` s názvem `Identity` v modelu definuje instance identit osob a v databázi má s osobami relaci many-to-one. Každá identita tedy může patřit pouze jedné osobě, ale osoba může vlastnit více identit. Tato vazba je také implementována v entitních třídách a je tedy možné pomocí metody získat všechny identity osoby.

Osoby jsou předány atributem do modelu, který zpracovává šablona `person.html` uložená ve složce `templates/management`. V ní je nejdříve vygenerován formulář s ovládacími prvky, jako je tlačítko pro vytvoření nové osoby nebo vyhledávací pole jehož princip fungování je popsán v sekci 4.4.3. Dále je na základě atributu s osobami vygenerována tabulka se dvěma sloupci, přičemž v prvním z nich je jméno osoby a v druhém jsou všechny její identity získané pomocí metody `getIdentities()`. V závislosti na nástroji, ze kterého identita pochází, je její jméno v databázi uvedeno buď v atributu `name` nebo `description`. V implementaci se tedy testuje, který z těchto atributů je neprázdný a ten se zobrazí.

Sloupec s osobou je vždy vygenerován pouze u její první identity a je rozšířen přes všechny řádky identit pomocí atributu `rowspan` s hodnotou odpovídající velikosti kolekce s identitami. Tento způsob implementace zaručí, že bude pro uživatele jednoznačně identifikovatelné, se kterými prvky v tabulce pracuje. Příklad výsledné vizualizace lze vidět na obrázku 4.5



Obrázek 4.5: Příklad uživatelského rozhraní s osobami

4.4.3 Zpracování uživatelských vstupů

Uživatel může sloučit vybrané osoby dvěma způsoby. První z nich spočívá ve výběru osob a následným stisknutím tlačítka u osoby, do které chceme výběr sloučit. V druhém případě uživatel vytvoří novou osobu vyplněním jejího jména v dialogovém okně a do této osoby se pak sloučí ty vybrané.

Ať už se jedná o jakýkoliv způsob odeslání formuláře, získaná data se vždy předají do kontoleru, přesněji do metody `mergeIdentities(Model model, ...)`. V ní se nejdříve musí zjistit, jestli byly vybrány nějaké osoby a jaký způsob odeslání formuláře byl použit. To lze zjistit pomocí existence atributu `submitId`, který by byl formulářem předán v případě, že byla zvolena již existující osoba. V opačném případě zase budou data obsahovat atribut `personName`, který obsahuje jméno nové osoby.

Pokud se jedná o novou osobu, vytvoří se nová instance dle zadaného jména, která je poté uložena do databáze. Na nově vzniklou instanci si metoda ponechá odkaz. V případě přiřazení výběru k existující osobě, se získá z databáze požadovaná instance. Staré instance osob, které budou sloučeny do té nové je potřeba po dokončení všech operací smazat. Aby to bylo možné, je potřeba najít všechny vazby, které souvisejí se starými osoby a změnit je na osobu novou.

Relační vazby s osobou

Jak bylo popsáno výše, je potřeba nalézt všechny vazby s původní osobou, aby bylo možné zastaralou osobu smazat. V tabulce 4.1 je uveden seznam tabulek, které takovou vazbu mají definovanou.

Tabulka	Atribut	Popis relace
identity	personId	Identita patří jedné osobě
work_unit	assigneeId	Pracovní úkol je přiřazen jedné osobě
work_item	authorId	Osoba je autorem úkolu, artefaktu nebo konfigurace
configuration_person_relation	personId	Osoba má vztah s konfigurací, např. ji commitnula
person_role	personId	Osoba má určitou roli v projektu
group_member	memberId	Osoba je členem nějaké skupiny
person_competency	personId	Osoba má definovány kompetence

Tabulka 4.1: Relační vazby s osobou

Pro změnu výše uvedených vazeb je v servisní třídě osob implementována metoda `updatePersonRelations()`, která využívá SQL dotazů za použití DML příkazů a změní tak příslušné atributy na hodnoty identifikátoru nové osoby. Po této změně jsou staré osoby, jejichž identity byly sloučeny, odebrány z databáze pomocí metody `deletePerson()`.

Funkce vyhledávání osob

Jelikož může mít každý projekt velké množství osob s identitami, bylo implementováno vyhledávání v souboru `search.js`. Vyhledávací pole při každé změně svého vstupu zavolá funkci `findEntries()`, která všechny položky v tabulce skryje přidáním atributu `display` s hodnotou `none`. Poté na základě zadaného vstupu prohledá všechny řádky tabulky a ty, které daný text obsahují opět zobrazí. Jelikož tabulka obsahuje atribut `rowspan` u sloupců s osobami, skrývání pouze některých řádků by znehodnotilo vizualizaci. Z toho důvodu je zobrazena vždy celá osoba se všemi identitami, i kdyby hledaný text obsahovala pouze jedna její identita.

4.5 Výčtové typy

Výčtové typy jsou součástí každé instance třídy `WorkUnit`, která také představuje tabulku v databázi. Také se vyskytují u osob pracujících na projektu, pokud definujeme jejich role. Zpracování požadavků uživatele na zobrazení těchto typů a následná manipulace s nimi je podstatou třídy `EnumController.java` z balíku `controller/management`. Implementace je složena z dvou částí. První se zabývá získáním dat o výčtových typech z databáze a následným vytvořením datových struktur, které by pomohly tato data vizualizovat. Druhá část následně zpracovává požadavky na změnu příslušnosti nějakého výčtové typu ke své třídě. Popis tříd výčtových typů je popsán v sekci 3.2.3.

4.5.1 Třídy a rozhraní výčtových typů

Každý druh výčtového typu má v aplikaci svou třídu, která jej reprezentuje. Stejně tak i entita představující klasifikace těchto výčtových typů. Pro všechny tyto třídy ještě existují servisní třídy a rozhraní repozitáře. Aby bylo možné se všemi těmito třída jednotně pracovat, každý druh výčtového typu implementuje rozhraní `EnumType`, kde jsou definovány metody pro získání názvu výčtového typu a jeho klasifikační třídy. Jejich servisní třídy pak implementují rozhraní `EnumService`. Třídy reprezentující klasifikace v databázi implementují rozhraní `Classification`, které poskytuje získání názvu klasifikační třídy a její namapovanou supertřídu. Servisní třídy těchto entit pak implementují rozhraní `Service`.

4.5.2 Vytváření struktury s daty o výčtových typech

Aby bylo jednodušší s velkým množstvím servisních tříd pracovat jednotně, je v kontroleru implementována metoda, která je automaticky zavolána po inicializaci parametrů třídy a je označena anotací `@PostConstruct`. V ní je vytvořena datová struktura slovník, jejíž klíčem je řetězec reprezentující název výčtového typu a hodnotou je třída `Pair` sloužící pro uložení dvojice objektů. Pár je tvořen instancemi třídy implementující rozhraní `Service` a druhou položkou jsou instance třídy implementující rozhraní `EnumService`. Slovník tedy na základě názvu výčtového typu (např. Priority) poskytne odpovídající pár service (pro uvedený příklad získá instance tříd PriorityClassificationService a PriorityService). Tato struktura umožní snazší zacházení se všemi dotčenými instancemi.

```
1  private List<Pair<Classification, List<EnumType>>> createClassPairs(Service
       service, Set<EnumType> enums) {
2
3      // List with enum types classes and their enums
4      List<Pair<Classification, List<EnumType>>> pairs = new ArrayList<>();
5
6      for(Classification enumClass : service.getAllClasses()) {
7          // For each enum classification create new list for enums
8          Set<EnumType> enumSet = new HashSet<>();
9
10         //If enum class is equal to current class, add enum into list
11         for(EnumType e : enums) {
12             if(Objects.equals(e.getClassId().getId(), enumClass.getId())) {
13                 enumSet.add(e);
14             }
15         }
16
17         //If enum set has some enums in it, add pair to the final list
18         if(!enumSet.isEmpty()) {
19             List<EnumType> enumTypes = new ArrayList<>(enumSet);
20             Collections.sort(enumTypes, Comparator.comparing(EnumType::getName,
                   String.CASE_INSENSITIVE_ORDER));
21             Pair<Classification, List<EnumType>> newPair = Pair.of(enumClass,
                   enumTypes);
22             pairs.add(newPair);
23         }
24     }
25     return pairs;
26 }
```

Listing 4.3: Metoda pro vytvoření klasifikačních párů

Aby bylo možné pracovat s výčtovými typy, je potřeba aby uživatel vybral nějaký projekt k editaci. Zde se provádí operace totožné s těmi, které jsou uvedeny v sekci 4.4.1. Poté se pro každý záznam ve slovníku (druh výčtového typu) iteruje přes instance zvoleného projektu a do připravené množiny se vždy přiřadí všechny instance výčtových typů, které projekt obsahuje. V momentě kdy je množina naplněna, je zavolána metoda

`createClassPairs()`, která pro všechny existující klasifikační třídy aktuálního výčtového typu vytvoří seznam s typy kategorizovanými do této třídy. Pokud tento seznam není prázdný, je vytvořen nový pár obsahující mapovací třídu a vzniklý seznam. Tímto způsobem je tedy pro každý druh výčtového typu vytvořen seznam, který obsahuje páry s klasifikační třídou a neprázdným seznamem výčtových typů mající tuto třídu. Algoritmus této metody je uveden výše.

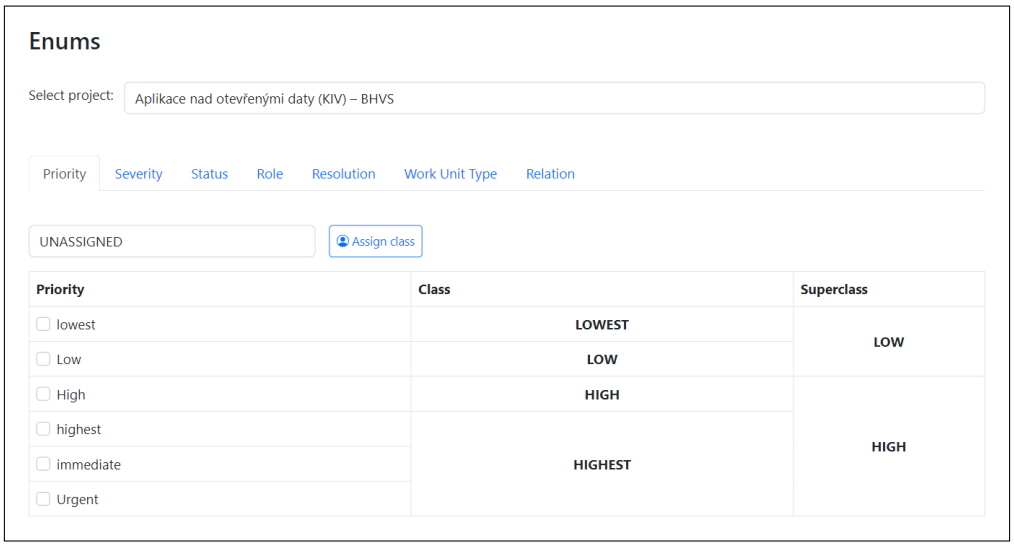
Dále je nutné vypočítat velikost atributu `rowspan`, který bude použit při generování sloupců se supertřídami, aby bylo možné je správně namapovat při vizualizaci na klasifikační třídy. Pro tento výpočet slouží metoda `calculateSpan()`, která iteruje přes všechny vzniklé páry s klasifikační třídou a zjisťuje, zda se její supertřída shoduje s tou v minulé iteraci. Pokud ano, je do rowspanu přičten počet instancí, které aktuální třída vlastní. Pokud ne, počítadlo se nastaví pouze na aktuální velikost seznamu s výčtovými typy. Tímto procesem vznikne pro každý druh výčtového typu pole o velikosti počtu klasifikačních tříd a je zde uveden vždy počet řádků, který se má v šabloně spojit, aby byla vizualizace validní.

Na konec se do modelu přidají jednotlivé seznamy s klasifikačními třídami a jejich výčtovými typy, všechny existující klasifikační třídy pro každý druh výčtového typu (potřebný pro možnost přeřazení výčtového typu pod jakoukoliv jinou třídu) a pole s rowspany supertříd.

4.5.3 Uživatelský pohled na výčtové typy

Jak bylo již uvedeno v návrhu 3.3, výčtových typů je několik druhů a uživatel bude chtít upravovat více těchto typů zároveň. Je tedy logickým krokem, že je uživatelské rozhraní implementováno pomocí záložek, mezi kterými může uživatel přepínat a v každé z nich editovat jiný druh výčtových typů.

Pro uživatelské rozhraní byla implementována šablona `enums.html`, která obsahuje základní kostru s ovládacími prvky a záložkami. Do této šablony se poté pro každý výčtový typ vloží fragment `enum-segment.html` se specifickými daty z modelu. Ten na základě páru, který získá od kontroleru, vygeneruje pro každou klasifikační třídu záznamy o instancích jejích výčtových typů. Velikost atributu `rowspan` u supertřídy je závislá na hodnotě v seznamu se jménem `superSpan`, který byl společně s ostatními daty vložen do modelu. Příklad toho, jak může výsledné uživatelské rozhraní vypadat je uvedeno na obrázku 4.6



Obrázek 4.6: Příklad uživatelského rozhraní s výčotvými typy

4.5.4 Manipulace s výčtovými typy

Pokud chce uživatel změnit mapování nějakých výčtových typů na jejich klasifikační třídu, musí u těchto záznamů označit zaškrtávací pole, vybrat z rozbalovacího seznamu jejich cílové mapování a potvrdit svou volbu stisknutím tlačítka `Assign class`.

Odeslaný formulář získá kontroler a na základě jména editovaného druhu výčtových typů získá servisní třídy odpovídající tomuto typu. Díky nim je pak možné z databáze získat instance vybraných výčtových typů a upravit u nich atribut `classId` odkazující na klasifikační třídu. Poté je instance uložena zpět do databáze.

V momentě kdy se toto provede u všech vybraných záznamů, je uživateli opět zobrazena původní stránka s již provedenými změnami.

4.6 Segmenty projektů

Případů užití aplikace, ve kterých se upravují segmenty projektů je více. Tato sekce zahrnuje popis implementace pro všechny tři samostatné případy užití. První se zabývá transformací kategorií pracovních úkolů na iterace, fáze nebo aktivity, druhá možností záměny iterací za fáze a poslední umožňuje přiřazovat podmnožině pracovních úkolů konkrétní aktivitu.

4.6.1 Kategorie

První implementovanou funkcí je transformace kategorií projektu na iteraci, fázi nebo aktivitu. Všechny tyto tabulky mají v databázi vazbu s instancemi třídy `WorkUnit`. Kategorie jsou spojeny vazbou many-to-many, ostatní zmíněné pak one-to-many. Tabulka 4.2 popisuje atributy třídy `WorkUnit`, které jsou použity ke zmíněným vazbám.

Vazba s tabulkou	Atribut	Popis vazby
WorkUnitCategory	id	Úkol je označen 0..N kategoriemi, pomocná tabulka M:N relace
Activity	activityId	Úkol má přiřazen maximálně 1 aktivitu
Iteration	iterationId	Úkol je součástí maximálně 1 iterace
Phase	phaseId	Úkol je součástí maximálně 1 fáze

Tabulka 4.2: Relevantní vazby se třídou WorkUnit

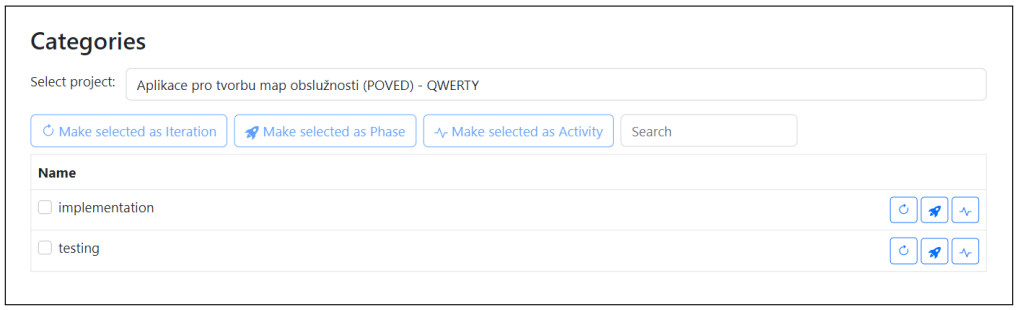
V případě transformace je potřeba vytvořit novou instanci požadované třídy, pokud existuje nějaký úkol, který transformovanou kategorii používá. Také je nutné validně změnit příslušné atributy úkolu, aby odkazovali na nově vytvořenou instanci. Zde spočívá problém v tom, co udělat v momentě, kdy má úkol již nějaké přiřazené fáze, iterace nebo aktivity a uživatel by transformací toto přiřazení chtěl přepsat. Implementace tohoto problému je uvedena v 4.6.1.

Vizualizace kategorií

Ve třídě `SegmentCategoryController.java`, která je definována v balíku `controller/management`, která představuje kontroler pro práci s kategoriemi, je implementována metoda `category(Model model, ...)`. Ta je namapována na HTTP request metodou GET. Po vybrání projektu, které je popsáno v 4.4.1 jsou z repozitáře kategorií pomocí servisní třídy vybrány ty záznamy, které jsou použity v instancích vybraného projektu. Tyto kategorie jsou seřazeny podle jejich jména a vloženy do modelu. Šablona `segment-category.html`, kde je definováno uživatelské rozhraní nejdříve vykreslí formulář s ovládacími prvky a poté iteruje přes seznam kategorií v modelu. Na základě těchto dat vygeneruje tabulku s popisem všech kategorií a tlačítky, které jsou uvedeny v návrhu 3.4 a mají sloužit k transformaci daného záznamu. Příklad uživatelského rozhraní pro tento případ užití lze vidět na obrázku 4.7.

Transformace kategorie

Pokud chce uživatel transformovat kategorii, může k tomu použít dva způsoby. Buď vybere potřebné kategorie a v horním segmentu s ovládacími



Obrázek 4.7: Příklad uživatelského rozhraní s kategoriemi

prvky stiskne tlačítko s entitou, na kterou chce výběr transformovat. Alternativou ke skupinové transformaci je změna pouze jedné kategorie, při které stačí stisknout jedno z tlačítek, které je dostupné u každého záznamu v tabulce. V obou případech se odešle do metody kontroleru `categoryChange(Model model, ...)` formulář s příslušnými vstupními daty. Tato data obsahují buď seznam všech vybraných kategorií nebo jeden identifikátor vybrané kategorie. K nim je také přidán typ cílové transformace, který má být na vybraných kategoriích proveden.

V metodě se nejdříve musí identifikovat jakým způsobem byl formulář odeslán. To lze provést na základě přítomnosti atributů získaných z formuláře. V případě, že se jedná o změnu pouze jedné kategorie, je vytvořen nový seznam, do kterého se kategorie přidá, aby bylo možné pracovat s oběma způsoby shodně. Po validaci typu cílové entity se přejde k samotné transformaci, kterou provádí metoda `processSelectedCategories()`.

Ve zmíněné metodě se iteruje přes všechny kategorie v seznamu a zjišťuje se, zda existuje nějaký pracovní úkol v databázi, který aktuální kategorii používá. Třída `Category` má definovanou vazbu s `WorkUnit`, a proto je možné pouze pomocí metody `getWorkUnits()` získat všechny příslušné úkoly a zjistit, zda je velikost této kolekce nenulová. Pro všechny úkoly, které vlastní danou kategorii se pak zkontroluje, zda mají prázdný atribut související z cílovou třídou, např. pokud uživatel transformuje kategorii na aktivitu, je zkontrolováno, že instance `WorkUnit`, které vlastní tuto kategorii mají i prazdný atribut `activityId`. Pokud existuje alespoň jeden takový úkol, může být nová transformovaná instance vytvořena a následně přiřazena jako atribut všem odpovídajícím úkolům s prázdným atributem.

Infomování o průběhu transformace

Jak již bylo nastíněno, každá transformace, která je vyžádána uživatelem nemusí být uspěšně dokonána. Hlavní překážkou je fakt, že pro vybranou

kategorii nemusí existovat instance úkolu, která by ji vlastnila anebo má tato instance vazební atributy již obsazené.

Z tohoto důvodu je v šabloně implementován alert, který po každé operaci vyhodnotí celý průběh transformace. Uživatel v něm může najít jak proběhli transformace jednotlivých kategorií a jaké úkoly tím byly ovlivněny. Implementace spočívá ve vytvoření instance třídy `StringBuilder`, do které se postupně přidávají jednotlivé texty o provedených operacích. Pro každou kategorii se nejdříve vytvoří záznam s jejím jménem a pokud projde všemi testy popsanými výše, připojí se informace o úspěšném dokončení operace. V opačném případě je uživateli vysvětleno proč nelze operaci dokončit.

Tato funkce umožňuje uživateli dokončit operace, které jsou validní, i přes to, že některé nemusejí skončit podle představ uživatele.

4.6.2 Fáze a iterace

Další implementovaná funkce umožňuje změnu fází projektu na iterace a obráceně. V principu řeší stejné problémy jako předchozí funkcionalita s rozdílem uživatelského rozhraní, které obsahuje vizualizaci obou zmíněných databázových tabulek.

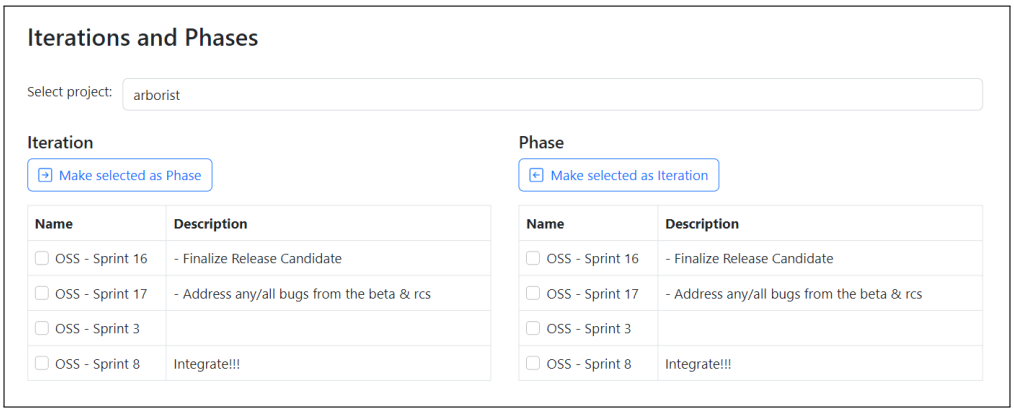
Výše popsaná funkcionalita je implementována především v kontroleru `SegmentIterationPhaseController.java` nacházející se v balíku `controller/management`, který kromě jiných, poskytuje hlavní metodu zobrazující základní vizualizaci. Po zvolení projektu jsou do modelu přidány seznamy s příslušnými instancemi tříd `Phase` a `Iteration`, které byly získány pomocí metod zvoleného projektu.

Vizualizace iterací a fází

Šablona `segment-iteration-phase.html` byla implementována podle návrhu 3.5. Po získání seznamů s instancemi z modelu se nejdříve vygeneruje tabulka s iteracemi a tlačítkem pro transformaci těchto entit na fáze. Poté je to samé provedeno s fázemi. Příklad uživatelského rozhraní lze vidět na obrázku 4.8. V testovacích datech, které jsou uloženy v databázi jsou momentálně záznamy fází a iterací shodné, není proto vidět rozdíl v jejich instancích.

Manipulace s fázemi a iteracemi

Po vybrání množiny fází či iterací a stisknutí odpovídajícího tlačítka pro transformaci se odešle do kontorleru formulář, ve kterém se seznam získaných



Obrázek 4.8: Příklad uživatelského rozhraní s fázemi a iteracemi

instancí transformuje.

K tomu slouží metody `changeToPhase()` a `changeToIteration()`, které nejdříve zkontrolují, zda jsou předané atributy validní a následně zjišťují jestli existuje nějaká instance úkolu, která iteraci/fázi používá. Pokud ano, vytvoří se nová instance reprezentující transformovanou entitu a přiřadí se do atributu `iterationId`/`phaseId` u všech takových úkolů, které mají tento atribut prázdný. Následně se prvnotní instance smaže z databáze.

Algoritmus je totožný s tím, který byl použit u kategorií a je popsán v sekci 4.6.1. Stejně tak probíhá i informování uživatele při nemožnosti dokončit některou z transformací, které je popsané v 4.6.1. Po dokončení je uživateli zobrazeno původní rozhraní s provedenými změnami.

4.6.3 Aktivity

Poslední funkcí, která se týká segmentů projektu je nastavení konkrétní aktivity pro určitou podmnožinu pracovních úkolů. K tomu slouží vytvořená třída `SegmentActivityController.java` uložena v balíku se jménem `controller/management`, která obsahuje metody zpracovávající vstupní data uživatele. V následujících sekcích je popsáno, jakými prostředky je dosaženo požadovaných funkčností.

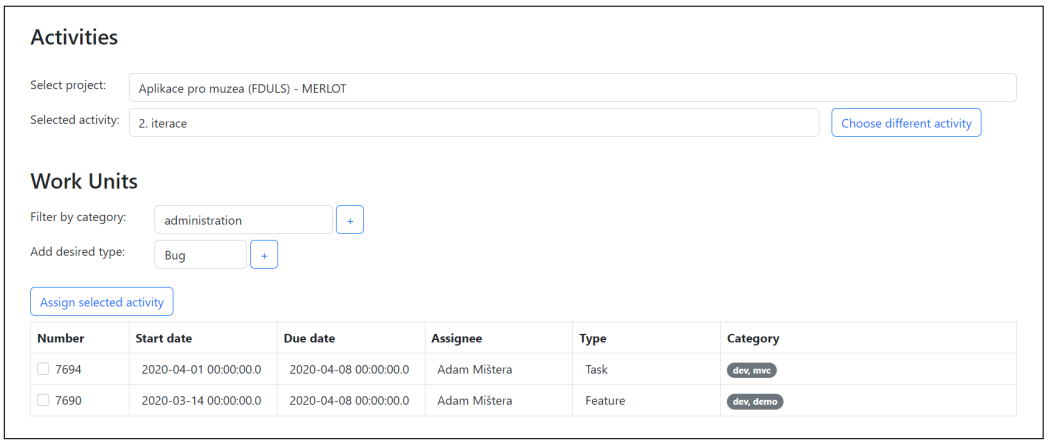
Vizualizace aktivit a úkolů

Po výběru projektu uživatelem jsou ze všech instancí tohoto projektu získány všechny instance tříd `Category` a `WuType`, které budou sloužit k filtraci úkolů. Jejich funkce je blíže popsána v sekci 4.6.3. Dále jsou do modelu přidány všechny aktivity projektu, které budou přiřazovány množině úkolů a instance `WorkUnit` na základě použitého filtru (při prvním načtením

stránky se zobrazí všechny instance). Následně je uživateli zobrazeno rozhraní definováno v souboru `segment-activity.html`, ve kterém zvolí jednu aktivitu, kterou bude přidělovat úkolům.

Na základě výběru aktivity přijme kontroler vyplněný formulář, jehož obsah zpracuje v metodě `selectActivity()`. Jde pouze o nalezení aktivity v databázi na základě získaného identifikátoru. Nalezená aktivita je pak přidána jako atribut do probíhající relace a uživateli se zobrazí rozhraní s již vybranou aktivitou a dodatečným segment. Tento segment obsahuje všechny úkoly projektu a panel s možností filtrace dle typu a kategorie úkolu. Vybraná aktivita lze kdykoliv změnit.

Tím je zobrazeno celé uživatelské rozhraní a uživatel s ním dále může pracovat. Příklad výsledné vizualizace je ukázán na obrázku 4.9.



Obrázek 4.9: Příklad uživatelského rozhraní s aktivitami

Přiřazení aktivity k úkolům

V momentě, kdy chce uživatel přiřadit aktivitu v vybraným úkolům, stiskne tlačítko *Assign selected activity* a tím odešle formulář do kontroleru. Zde se zvaliduje, že sezam s úkoly není prázdný a poté každému z nich změní pomocí metody `setActivity()` atribut `activityId`. Změněné instance se poté uloží zpět do databáze za použití servisní metody `saveWorkUnit`. Ta využívá definovaného repozitáře a zabudované metody pro uložení aktuální instance.

Funkce filtru

V kontroleru jsou definovány metody, které přidávají jednotlivé kategorie do filtru nebo je z něj odstraňují. Jde o metody `addCategoryFilter()` a `removeCategoryFilter()`. Stejné metody jsou pak použity i v případě

filtrace podle typy úkolu. Pokud uživatel v rozhraní přidá nebo odebere do/z filtru nějakou instanci, je zavolána příslušná metoda konstruktoru. V ní se do atributu relace s filtry přidá/odebere tato instance a stránka je znovu načtena.

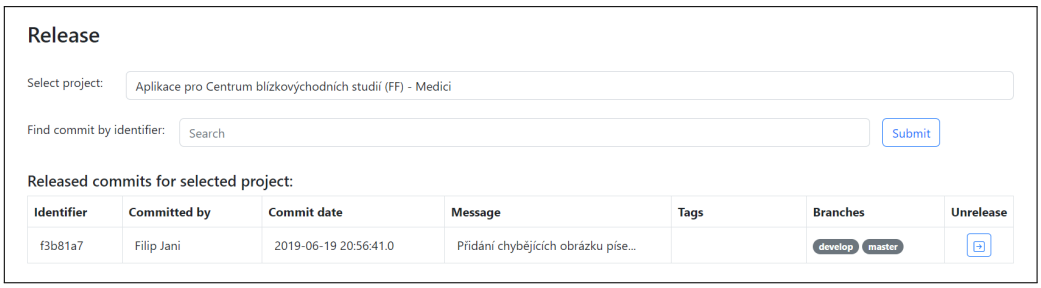
Při znovu načtení stránky se z kolekce všech kateogrií či typů odstraní pomocí rozdílu množin ty instance, které jsou zahrnuty ve filtru. Tím je zajištěno, že uživatel nemůže přidat stejný filtr vícekrát. Následně se podle aktuálního filtru vyberou z databáze pouze ty úkoly, které mu vyhovují.

4.7 Release

Posledním implementovaným případem užití je vizualizace konfigurací a úprava tagu release. Vizualizace této funkce je rozdělena do dvou částí, kde každá plní jinou úlohu. První poskytuje uživateli informaci o tom jaké konfigurace již byly označeny jako released a umožňuje toto označení odstranit. Součástí této části je také vyhledání konfigurace projektu na základě jejího identifikátoru. Po úspěšném vyhledání se zobrazí druhá, podrobnější část vizualizace, která poskytuje veškeré podstatné informace o vyhledané konfiguraci a také umožňuje manipulovat s atributem released.

První část implementace

Jakmile uživatel vybere projekt, ve kterém chce konfigurace editovat, zobrazí se základní část vizualizace, která je definována v souboru `release.html`. Ta obsahuje vyhledávací pole pro konfigurace a následně i tabulku s již commitnutými konfiguracemi. Příklad zobrazeného rozhraní lze vidět na obrázku 4.10



Obrázek 4.10: Příklad uživatelského rozhraní s konfiguracemi

V tomto uživatelském rozhraní lze konfiguraci, která je označena jako released, odstranit toto označení. Při tomto úkonu je zavolána metoda kontroleru s názvem `unRelease()`, která vybrané instanci nastaví atribut

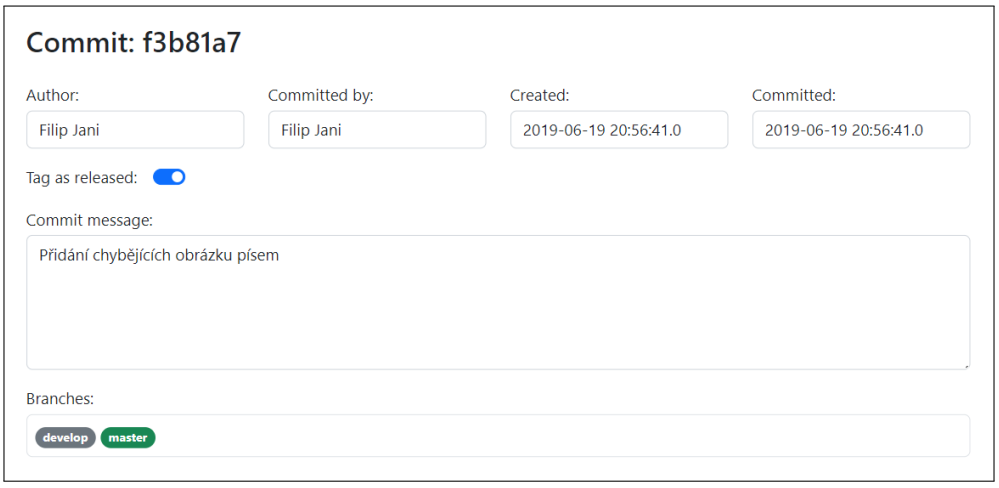
`isRelease` na hodnotu `false` a uloží ji zpět do databáze za pomoci servisní třídy.

Po vyhledání konfigurace na základě identifikátoru je zavolána metoda `openCommit()`, která se pokusí o získání instance z databáze. Pokud je instance nalezena, je přidána do modelu a uživateli se poté zobrazí šablona `commit.html`, která obsahuje detailní informace o konfiguraci.

Druhá část implementace

Commitnutou konfiguraci reprezentuje třída `Commit`, která však v databázi neobsahuje všechna potřebná data, a proto je za použití relačních vazeb rozšířena o atributy ze třídy `CommittedConfiguration`, `Configuration` a `WorkItem`. Dále má třída relaci many-to-many s třídou `Branch`, která říká na jakých větvích byla konfigurace commitnuta. Všechna data získaná z těchto tříd se promítnou do celkové vizualizace.

Nejdříve jsou vygenerována needitovatelná pole se základními údaji o autorovi, osobě která konfiguraci commitnula a datech ve kterých vše proběhlo. Následně se vygeneruje přepínač, který při své změně zavolá metodu `changeRelease()`. Tato metoda na základě aktuální polohy přepínače pouze změní atribut `isRelease`. Dále je ve vizualizaci zobrazena zpráva, která byla uvedena u commitu a branche, na kterých byl commit proveden. Generování textů s názvy branchí jsou generovány v cyklu a pokud je některá označena jako hlavní (lze identifikovat atributem `IsMain`) je zvýrazněna zelenou barvou. Příklad výsledné vizualizace je vidět na obrázku 4.11



Obrázek 4.11: Příklad uživatelského rozhraní s commitem

4.8 Shrnutí implementace

Celkem bylo vytvořeno či rozšířeno 124 tříd a rozhraní, pričemž bylo implementováno cca 8000 řádků kódu. Velký počet vytvořených tříd se odvíjí od faktu, že každá modelová třída potřebuje pro své fungování další obslužný kód v podobě servisních tříd a repozitářů. Dalším důvodem je použití velkého množství rozhraní.

5 Ověřování funkčnosti

Aby bylo možné zaručit předpokládanou funkčnost aplikace, proběhlo u ní testování, které je popsáno v následujích sekcích.

5.1 Způsob testování

Způsob ověřování výsledků vychází z předpokladu, že implementovaná webová aplikace pracuje převážně s databází a neobsahuje velké množství kódu, které by bylo za potřebí jednotkově testovat. Na základě dohody s vedoucím práce není její součástí rozsáhlé testování i z toho důvodu, že je aplikace paralelně vyvíjena dalšími studentskými týmy v rámci univerzitních předmětů, které se hlubším testováním aplikace zabývají (především mockováním databáze). Z těchto důvodů bylo na aplikaci provedeno uživatelské testování podle scénářů, které dokáže nejvíce vyhovujícím způsobem ověřit funkčnost implementovaných funkcionalit.

5.2 Testové sady

Celkem bylo vytvořeno 43 testovacích případů, které pokrývají ověření všech implementovaných funkcí. Pro každý use-case je vytvořena testovací sada, ve které jsou jednotlivé případy sdruženy. Každý případ má definované prerekvizity, které je nutné splnit před zahájením testování a následně je tester instruován dle definovaného scénáře.

Kroky scénáře jsou definovány popisem, ve kterém je vysvětleno, co má tester provést za akci a jaký je její očekávaný výsledek. Do záznamového archu se poté vyplní, jestli skutečnost odpovídá očekávání. Poté je zvolen výsledek testového kroku – `PASS` nebo `FAIL` a tester může přidat do záznamového archu jakoukoliv poznámku. Pokud jsou všechny kroky vyhodnoceny pozitivně (`PASS`), je celému testovému případu přiřazena tato hodnota. V testovém archu je na základě celkových výsledků každé testové sady vyznačena míra úspěšnosti testů.

Příklad testového případu lze vidět na obrázku 5.1. V archivu odevzdaném společně s touto prací jsou přiloženy i původní archy, které byly testerům předány.

ENUMS_02			
Popis testu:	Úspěšné přeřazení výčtových typů do jiné třídy		
Prerekvizity:	Spuštěna webová aplikace, přepnuto na stránku Enums, zaškrtávací pole prázdné, karta Priority		
Vyhodnocení testu:	FAIL		
Poznámka:			
Testovací kroky			
Popis kroku	*Očekávaný výsledek*	*Popis výsledku*	*Výsledek*
Přepnutí na záložku "Work Unit Type"	Zobrazení správné karty, supertřídy nejsou vyplněny		FAIL
Vybrání 5 výčtových typů, které nejsou ve třídě "Unassigned" v tabulce pomocí zaškrtávacího políčka	U všech vybraných výčtových typů se zaškrtávací políčko zabarví do modra		FAIL
Z rozbalovacího seznamu tříd vyber "Unassigned" a klikni na tlačítko "Assign"	Všechny vybrané výčtové typy byly přeřazeny do třídy "Unassigned". V horní části stránky se zobrazí zelený alert, který potvrzuje úspěšné dokončení akce		FAIL

Obrázek 5.1: Příklad jednoduchého testového případu

5.3 Výsledky testování

Vyrobené testové sady byly zaslány třem testerům společně s manuálem na spuštění aplikace a podrobným vysvětlením okolností projektu. Skupina uživatelů vytvořeného nástroje byla definována jako ‘technicky zdatní uživatelé srozuměni s projektovými daty v databázi’. Dle této definice byli vybráni i testeři, kterým byla částečně znalost databáze doplněna. Vyplněné testové sady lze najít v přiloženém ZIP souboru, viz příloha A. Záznamové archy jsou anonymizovány.

V rámci testování byla odhalena 4 selhání a jeden vizuální problém. Zároveň také byla objevena chybná implementace jedné funkce, která musela být opravena. Celková úspěšnost testů tedy činila **90,7%**. Všechna selhání byla opravena a znovu otestována připravenými sadami.

6 Závěr

Cílem této práce bylo vytvořit aplikaci pro správu vybraných aspektů dat v databázi projektu SPADe pomocí uživatelského rozhraní, která by umožňovala opravy chyb způsobené nevhodnou klasifikací vybraných položek a pomáhala by tak udržovat konzistenci dat. Za tímto účelem byly nastudovány odlišnosti v konceptech jednotlivých ALM nástrojů, ze kterých se projektová data v nástroji SPADe dolují, společně s datovým modelem tohoto nástroje, jeho tabulkami a relacemi.

Na základě získaných poznatků byly vybrány aspekty databáze, které se podílejí na nekonzistenci dat a po seznámení se s aplikací Web Interface bylo navrženo její rozšíření o správu těchto projektových dat.

Následně byla provedena implementace navržených funkcionalit při použití stanovených technologií a dodržení zavedené architektury aplikace. Celkem bylo implementováno cca 8000 řádků kódu a práce obsahovala práci s databází o velikosti 85 tabulek (započítano i s pohledy). Funkčnost a použitelnost vytvořené aplikace byla ověřena sadou testů s celkovou úspěšností 90,7%, která byla po opravě nalezených chyb zvýšena na 100%. Tím byly splněny všechny body v zadání práce.

Přínos práce je především zvýšení efektivity a přesnosti detekování antivzorů v tomto nástroji a budoucího výzkumu. V rámci práce byla nalezena další rozšířené, které je možné implementovat v navazujících pracích. Aplikace bude i nadále spravována dalšími univerzitními týmy v různých oblastech.

Seznam zkratek

- **ALM** – Application Lifetime Management – nástroje využívané k efektivnímu řízení vývoje softwarových projektů
- **CI/CD** – Continuous Integration/Continuous Delivery – nástroje využívané k nasazení a integrací změn aplikace
- **CVS** – Concurrent Version System – systém sloužící pro správu verzí projektu
- **HTTP** – Hypertext Transfer Protocol – internetový protokol sloužící pro přenost hypertextových dokumentů
- **JDBC** – Java Database Connectivity – rozhraní pro přístup k databázi pomocí jazyka Java
- **SVN** – Apache Subversion – systém pro správu verzí projektu, která je dnešní náhradou za CVS
- **SQL** – Structured Query Language – strukturovaný dotazovací jazyk používaný pro práci s relačními databázemi
- **VCS** – Version Control System – nástroje pro sledování vývoje změn v projektech

Literatura

[1] Bird, C. et al. The promises and perils of mining git. In *2009 6th IEEE International Working Conference on Mining Software Repositories*, s. 1–10, 2009. doi: 10.1109/MSR.2009.5069475.

[2] Chacon, S. – Straub, B. *Pro Git.* Apress, 2014. ISBN 978-1-4842-0077-3.

[3] Holy, L. et al. Software Engineering Projects Analysis using Interactive Multimodal Graph Explorer – IMiGEr. In *Proceedings of the 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 3: IVAPP*, s. 330–337, Setubal, Portugal, 2019. SciTePress.

[4] Kroll, P. – Kruchten, P. *The Rational Unified Process made easy: A practitioner's guide to the RUP.* Addison-Wesley, 2003. ISBN 0-321-16609-4.

[5] Lakeworks. *The Scrum project management method* [online]. Open Clip Art Library, 2009. [cit. 2023/03/05]. Part of the image is based on public domain graphics from Open Clip Art Library (openclipart.org). Dostupné z: `https://cs.wikipedia.org/wiki/Scrum`.

[6] Mania, M. *Vodopádový model (Waterfall model)* [online]. Management Mania, 2015. [cit. 2022/06/11]. Dostupné z: `https://managementmania.com/cs/vodopadovy-model-waterfall-model`.

[7] Pícha, P. *Detecting software development process patterns in project data.* Západočeská univerzita v Plzni, 2019. Technická zpráva.

[8] Pícha, P. *Software Process Anti-patterns Detector* [online]. 2022. [cit. 2023/04/30]. Dostupné z: `https://github.com/ReliSA/SPADe`.

[9] Pícha, P. – Brada, P. ALM tool data usage in software process metamodeling. In *Software Engineering and Advanced Applications (SEAA), 2016 42th Euromicro Conference on*, s. 1–8, Piscataway, NJ, USA, 2016. IEEE.

[10] Red Hat, I. *What is CI/CD?* [online]. Red Hat, Inc., 2022. [cit. 2023/05/02]. Dostupné z: `https://www.redhat.com/en/topics/devops/what-is-ci-cd`.

[11] Schwalbe, K. *Information Technology Project Management, 4th edition.* Course Technology, a Thomas Learning Company, 2006. ISBN 978-80-251-1526-8.

[12] Scrum.org. *What is Scrum?* [online]. Scrum.org, 2020. [cit. 2023/03/05]. Dostupné z: `https://www.scrum.org/learning-series/what-is-scrum`.

[13] Tittel, E. – Brush, K. *Application lifecycle management (ALM)* [online]. TechTarget, 2022. [cit. 2023/04/24]. Dostupné z: `https://www.techtarget.com/searchsoftwarequality/definition/application-lifecycle-management-ALM`.

[14] Váně, O. *Analýza přítomnosti anti-vzorů v datech nástrojů pro řízení projektů.* Západočeská univerzita v Plzni, 2022. Diplomová práce.

[15] William J. Brown, S. W. T. H. W. M. *AntiPatterns in Project Management.* John Wiley & Sons, Inc., New York, NY, USA, 2000. ISBN 978-0471363668.

[16] Štěpánek, P. *Detekce špatných praktik projektového řízení v open-source projektech.* Západočeská univerzita v Plzni, 2022. Bakalářská práce.

Seznam obrázků

Seznam tabulek

Přílohy

A Struktura ZIP souboru

Práce je uložena v souboru ZIP, který má následující adresářovou strukturu:

- `Text_prace` je složka obsahující soubor s textem bakalářské práce a jeho zdrojové kódy v LaTeXu.
  - `Zdrojove_kody` je podadresář se zdrojovými kódy bakalářské práce.
  - `Text` je podadresář obsahující PDF soubor s textem práce.
- `Aplikace_a_knihovny` je složka se zdrojovými kódy aplikace, konfiguračními soubory a dokumentací Javadoc.
  - `AntiPatternDetectionApp` je podadresář obsahující celou aplikaci.
    - Adresář `src` obsahuje zdrojové kódy aplikace.
    - Adresář `java-doc-new` obsahuje dokumentaci Javadoc.
    - Adresář `db` obsahuje databázové skripty
    - Soubor `docker-compose.yml` obsahuje konfiguraci aplikace pro spuštění v Dockeru
- `Vstupni_data` je složka, která obsahuje SQL skripty pro definici a nastavení databáze SPADe a také celý fyzický datový model databáze.
  - `Databazove_skripty` je podadresář s SQL skripty, které vytvářejí databázi potřebnou k běhu aplikace.
  - `Fyzicky_model` je podadresář obsahující obrázky s fyzickým modelem databáze.
- `Vysledky` je složka obsahující záznamy z testování.
  - `Prazdna_sada` je podadresář s nevyplněnými testovacími archy.
  - `Vyplnene_sady` je podadresář obsahující vyplněné záznamové archy od testerů.
- `Readme.txt` je dokument popisující adresářovou strukturu ZIP souboru.

B Instalační manuál

Z důvodu, že je tato práce rozšířením již existující aplikace, byl dodržen způsob jakým je možné aplikaci spustit. Projekt byl již dříve nakonfigurován tak, aby ho bylo možné spustit pomocí nástroje Docker. Tento manuál tedy reflektuje původní nastavení a vysvětluje jakým způsobem vytvořenou aplikaci v nástroji spustit.

B.1 Nástroje potřebné ke spuštění

Aby bylo možné aplikaci spustit, je zapotřebí mít nainstalované tyto nástroje:

- Docker Desktop[1]
- Docker Compose[2]
- WSL2[3] (pouze pro Windows)
- Nástroj pro připojení do správy databáze (např. MySQL Workbench[4])

B.2 Postup spuštění aplikace

Pro úspěšné spuštění aplikace je potřebné dodržet následující kroky:

1. Přesunutí se do složky s názvem `AntiPatternDetectionApp`, která je kořenovým adresářem projektu. Zde by se měl nacházet konfigurační soubor `docker-compose.yml`.
2. Spuštění nástroje Docker Desktop a otevření WSL konzole v adresáři, který je definován v kroce 1.
3. Zde se spustí příkaz:

```
docker-compose build
```

[1]Návod pro instalaci Docker Desktop dostupný na adrese: `https://docs.docker.com/get-docker`

[2]Návod pro instalaci Docker Compose dostupný na adrese: `https://docs.docker.com/compose/install`

[3]Návod pro instalaci WSL2 dostupný na adrese: `https://ubuntu.com/tutorials/install-ubuntu-on-wsl2-on-windows-10`

[4]Návod pro instalaci MySQL Workbench dostupný na adrese: `https://dev.mysql.com/downloads/workbench`

Tento příkaz v Dockeru vytvoří tzv. container, který obsahuje databázi aplikace a její samotnou instanci.

4. Po vytvoření komponent je nutné aplikaci spustit pomocí příkazu:

   `docker-compose up -d`

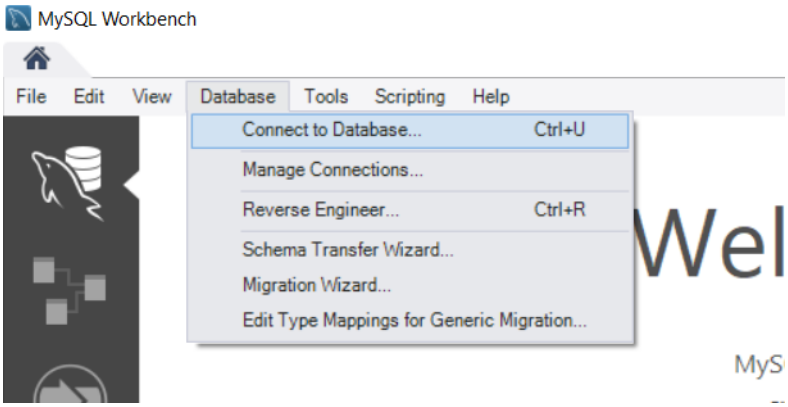
   Tento příkaz spustí MySQL databázi na portu `3306` a samotnou webovou aplikaci na portu `8080`.

5. Aby byla možná správná funkce aplikace, musí být vytvořena databáze a provedena její konfigurace s naplněním projektových dat. Toto je popsáno v sekci B.3

B.3 Nastavení databáze

Aplikace potřebuje pro svůj běh databázi s definovanou strukturou. Z tohoto důvodu je potřeba se připojit na vytvořenou lokální databázi pomocí některého nástroje pro její správu. Návod je psán pro MySQL Workbench.

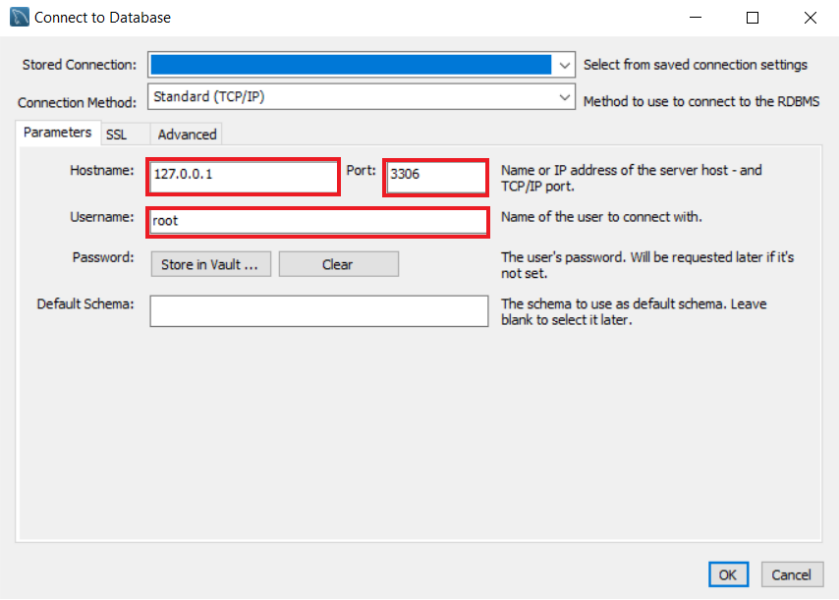
1. Otevřít aplikaci MySQL Workbench a pomocí horním menu aplikace vybrat možnost **`Database -> Connect do database...`**



Obrázek 1: Připojení k databáze pomocí MySQL Workbench

2. Zobrazí se dialogové okno s údaji o připojení k databázi. Parametry připojení vyplňte dle následujících údajů a poté klikněte na tlačítko **OK** ve spodní částí okna:

   Hostname: **`127.0.0.1`** , Port: **`3306`** a Username: **`root`**

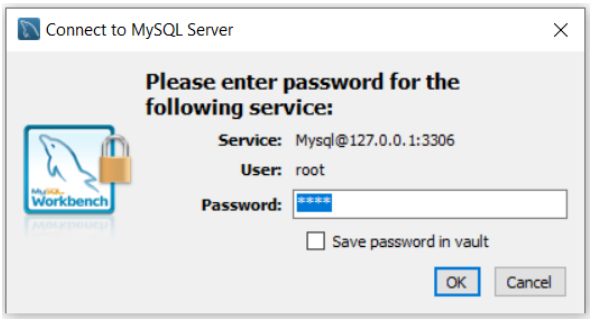


Obrázek 2: Dialogové okno s vyplněnými údaji o připojení

3. Zobrazí se dialogové okno s prosbou o zadání hesla:

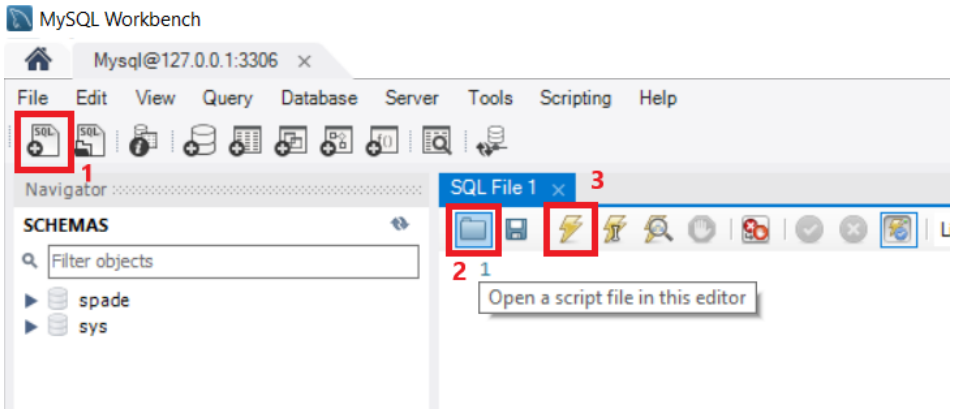
   Password: **`root`**

   Po stisknutí na tlačítko **OK** by jste měli být připojeni do databáze.



Obrázek 3: Zadání hesla pro připojení k databázi

4. V horní části panelu klikněte na ikonku papíru s textem SQL+, která otevře prostředí pro vytváření skriptů. Následně klikněte na ikonku adresáře, čímž se otevře dialogové okno pro výběr souboru. Zde vyberte skript ze složky `Vstupni_data` této práce s názvem **`spade.sql`**. Poté stiskněte ikonu blesku, čímž se skript vykoná a vytvoří základní strukturu databáze.



Obrázek 4: Načtení SQL skriptů a jejich spuštění

5. Krok 4 opakujte pro všechny soubory ve složce `Vstupni_data` v pořadí **`spade-views.sql`**, **`spade-config.sql`** a jako poslední **`spade-data.sql`**. Celkově vytvoří databázi naplněnou daty.

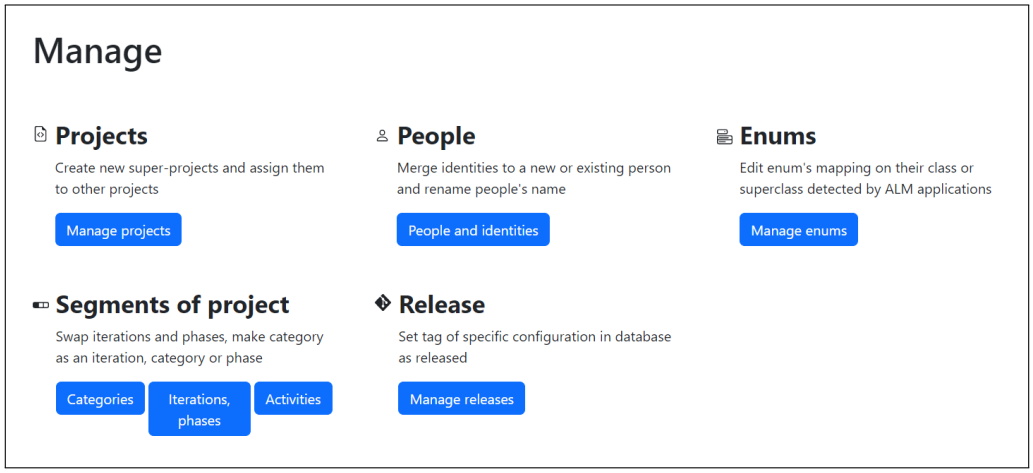
6. Po vykonání všech databázových skriptů se můžete odpojit z databáze pomocí symbolu křížku u záložky s aktuálním připojením. Aplikace by nyní měla být dostupná na **`localhost:8080`**

C Uživatelská příručka

Aplikace je určena pro manipulaci s vybranými aspekty projektových dat. V příručce je uveden návod na použití implementovaných funkcionalit.

C.1 Hlavní stránka aplikace

Pro zobrazení základního rozhraní implementované aplikace je potřeba otevřít prohlížeč a z hlavní stránky přejít pomocí navigačního menu v záhlaví na záložku *Manage*. Poté se zobrazí stránka sloužící jako rozcestník pro jednotlivé funkcionality, která je zobrazena na obrázku číslo 5.



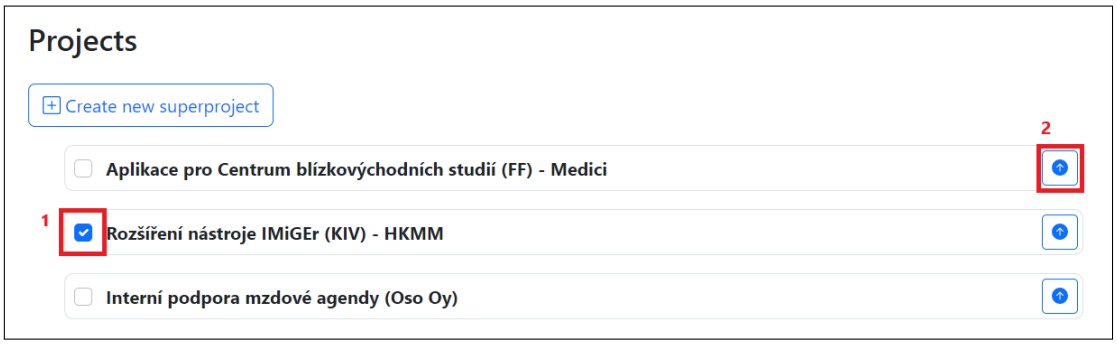
Obrázek 5: Hlavní stránka sloužící jako rozcestník aplikace

C.2 Změna hierarchie projektů

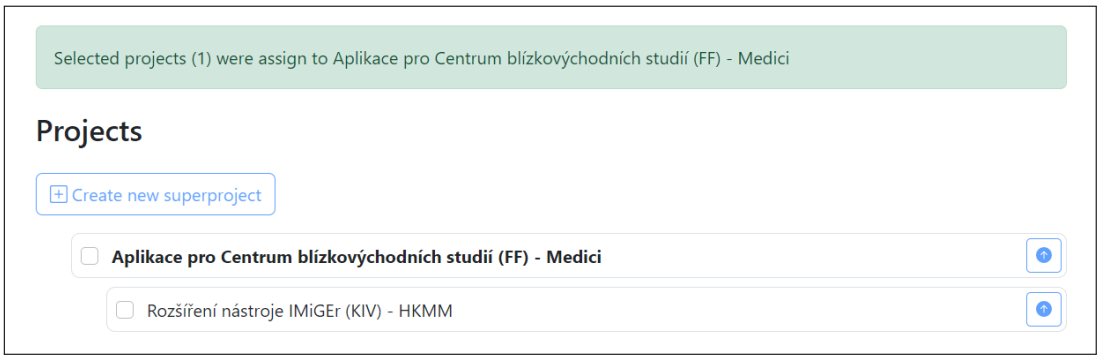
Návod na změnu hierarchie projektů je uveden v následujících bodech:

1. Stisknutí tlačítka *Manage projects* na stránce `Manage`. Poté se zobrazí stránka s projekty a vizualizace jejich hierarchie. Každý projekt má na levé straně zaškrtávací pole sloužící k výběru množiny projektů a na pravé straně tlačítko, které slouží pro přiřazení výběru pod projekt u kterého bylo stisknuto. Popsané prvky lze vidět na obrázku číslo 6.

2. Učinění výběru pomocí zaškrtávacích políček a poté stisknutí tlačítka u projektu, který má být rodičem pro vybrané projekty. Změna se projeví ve vizualizaci přesunutím vybraných záznamů, což lze vidět na obrázku číslo Zároveň se zobrazí hláška o úspěšně provedené akci 7.



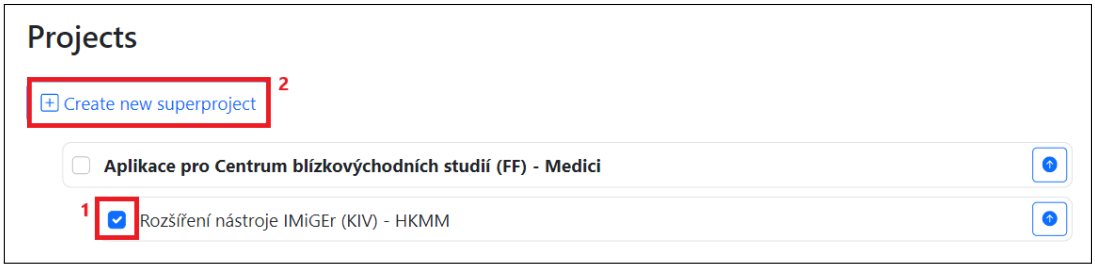
Obrázek 6: Rozvržení ovládacích prvků projektů



Obrázek 7: Alert potvrzující úspěšné dokončení akce

Pokud je potřeba vytvořit nový projekt, lze to provést dle následujících bodů:

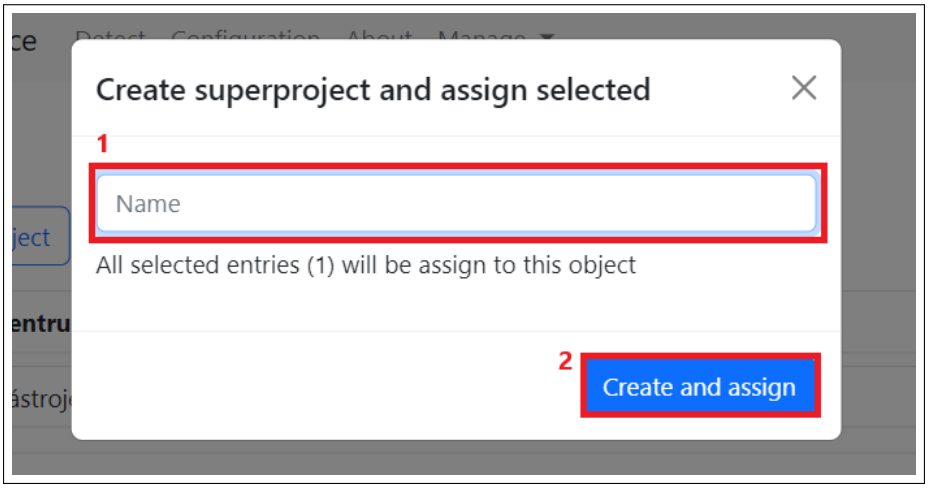
1. Zobrazení stránky *Projects* dle kroku `1` v předchozím návodu.
2. Výběr projektů, které chceme podřadit nově vytvářenému projektu a stisknutí tlačítka `Create new superproject` v horní části stránky tak je je zobrazeno na obrázku 8.



Obrázek 8: Vytvoření nového projektu

3. Zobrazí se dialogové okno s textovým polem pro zadání názvu nového projektu. Zde se doplní jméno projektu a stiskne se tlačítko `Create`

`and assign`. Proces je zobrazen na obrázku 9. Změna se projeví ve vizualizaci přesunutím vybraných záznamů. Pokud nebylo zadáno jméno projektu, aplikace zobrazí chybovou hlášku 10.



Obrázek 9: Dialogové okno pro vyplnění jména projektu

Obrázek 10: Chybová hláška při nezadání jména

Výběr rodičovského projektu a jeho potomků

Při výběru projektů nelze vybrat rodičovský projekt a zároveň jeho potomky, protože při změně hierarchie se vždy přesouvá celá stromová struktura. Takovému výběru je zamezeno díky vypnutí ovládacích prvků u potomků. Pokud chcete změnit hierarchii i u potomků vybrnaého projektu, je nutné toto provést ve více krocích dle popsaného návodu.

Znehodnocení stromové struktury

Jelikož hierarchie projektů odpovídá stromové struktuře, není možné vytvářet v takovém grafu smyčky. Z tohoto důvodu je při takovém pokusu o změnu hierarchie uživateli zobrazena chybová hláška 11.

Obrázek 11: Chybová hláška při nedovolené úpravě stromu

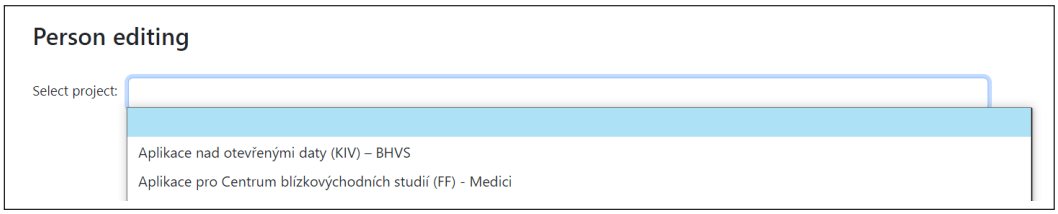
Nastavení rodičovského projektu

Někdy je potřebné nastavit podřízený projekt jako rodičovský a vymanit ho tak z dosavadní struktury. Pro tento use-case je nutné vybrat projekt pomocí zaškrtávacího pole a kliknout na tlačítko přímo u jeho záznamu. Toto není možné provést v případě, že je vybráno více projektů.

C.3 Slučování osob a jejich identit

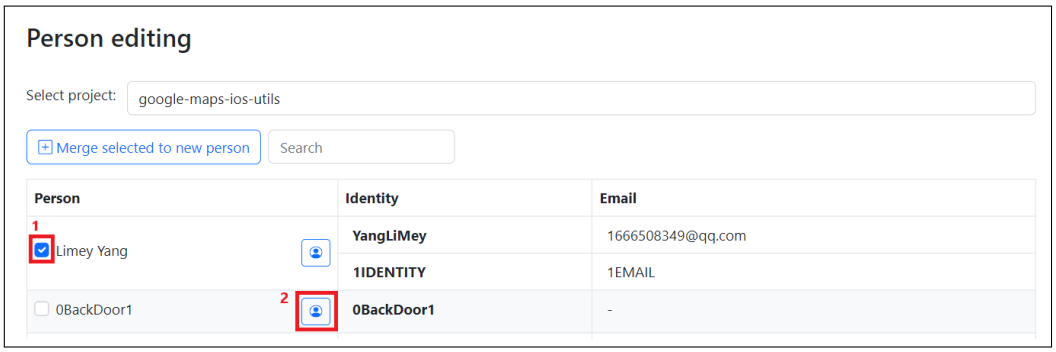
Sloučení osob je možné provést dle následujícího návodu, přičemž dojde k odstranění původních osob a převedení veškerých navazujících atributů na osobu novou.

1. Stisknutí tlačítka *People and identities* na stránce `Manage`. Pokud dosud nebyl vybrán žádný projekt pro editaci, zobrazí se rozbalovací seznam s výběrem projektu. Zde je potřeba učinit výběr 12. Poté se zobrazí tabulka s osobami u kterých je ovládací prvek pro selekci a tlačítko pro sloučení osob. V tabulce jsou pak uvedeny všechny identity dané osoby.



Obrázek 12: Výběr projektu

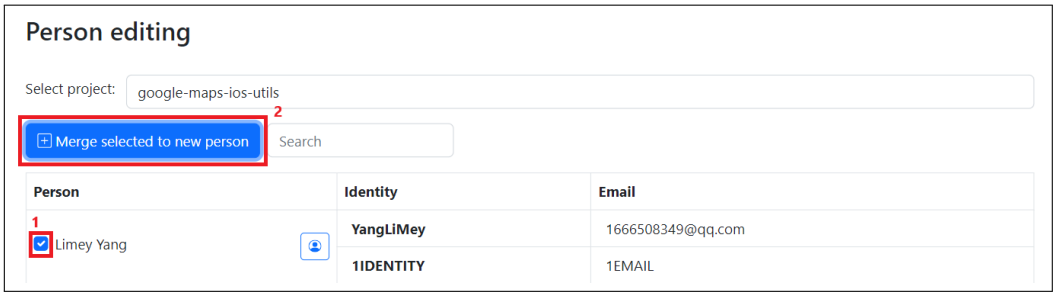
2. Výběr osob, které chceme sloučit a stisknutí tlačítka u cílové osoby. Ukázku výběru lze vidět na obrázku 13. Poté je provedeno samotné sloučení osob, identity jsou přiřazeny cílové osobě a další navazující atributy také. Původní osoby jsou smazány.



Obrázek 13: Výběr osob a jejich sloučení

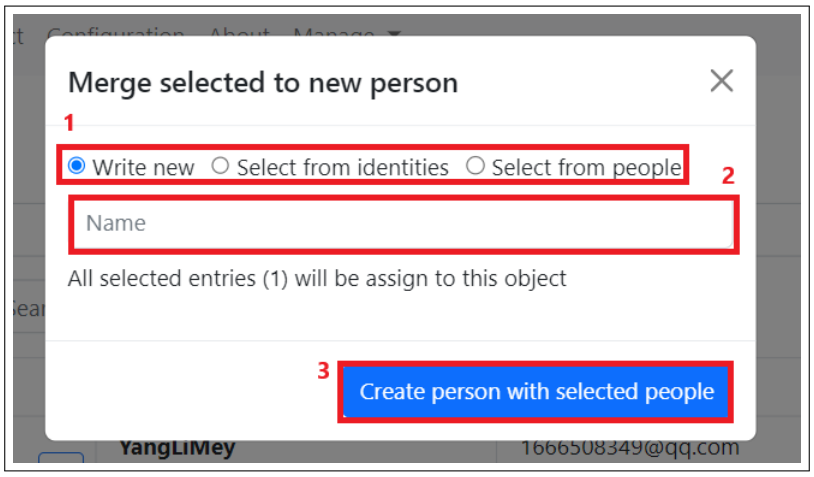
Pokud je potřeba vytvoření nové osoby, lze to provést dle následujících bodů:

1. Výběr osob, které budou sloučeny do nově vytvářené osoby.
2. Stisknutí tlačítka *Merge selected to new person*, které zobrazí dialogové okno pro zadání jména nové osoby. Tlačítko je v horní části obrázku 14.



Obrázek 14: Sloučení osob do nové osoby

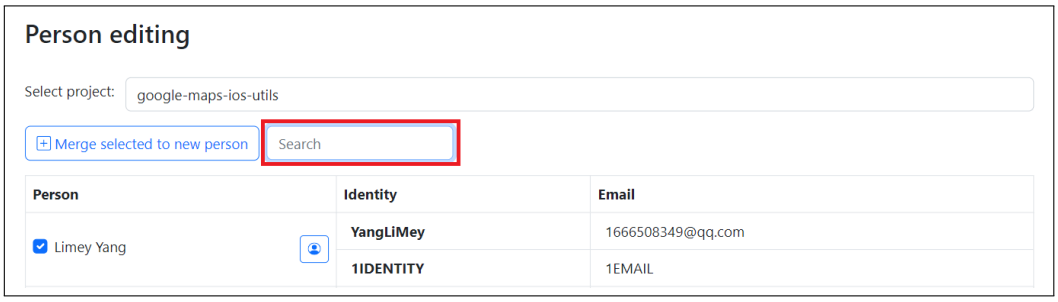
3. Výběr jakým způsobem bude jméno zadáno. První možností je zadat jméno textově, druhá vybrání jména osoba z jmen identit vyselektovaných osob a třetí možností je výběr z jmen vybrnaých osob. Mezi možnostmi lze přepínat pomocí přepínačů v horní části dialogového okna. Po výběru je nutné stisknout tlačítko *Create person with selected people*. Okno lze vidět na obrázku 15.
4. Osoby budou poté sloučeny do nově vytvořené osoby, která přebere veškeré atributy od těch původních.



Obrázek 15: Zadávání jména nového osoby

Vyhledávání osob

Osoby lze vyfiltrovat zadáním řetězce do vyhledávacího pole vedle tlačítka *Merge selected to new person*. Vyhledávají se shody řetězce se jménem osoby, identity i jejího emailu. Při filtraci se vždy zobrazí celá osoba nehledě na to, kolik identit v ní splňuje podmínky filrace. Pozice ovládacího prvku lze vidět na obrázku 16.



Obrázek 16: Vyhledávácí pole pro filtrování osob

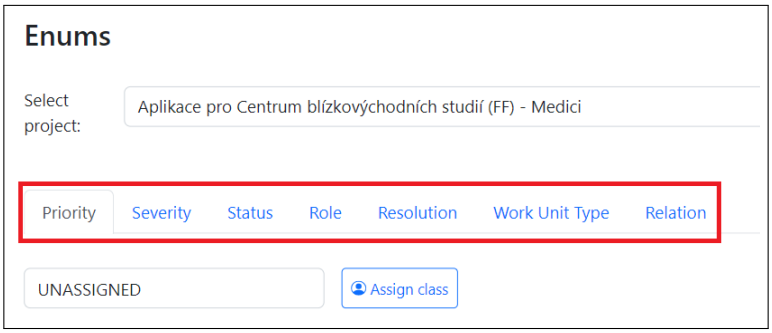
C.4 Změna mapování výčtových typů

Výčtové typy mají přiřazenou kategorizační třídu, která lze změnit díky implementované funkcionalitě. Provedení také změny je v krocích popsáno v následujícím seznamu:

1. Stisknutí tlačítka *Manage enums* na stránce `Manage`. Pokud dosud nebyl vybrán žádný projekt pro editaci, zobrazí se rozbalovací seznam s výběrem projektu, stejně jako na obrázku 12. Zde je potřeba učinit

výběr. Zobrazí se záložky pro jednotlivé druhy výčtových typů, kterým lze měnit definované třídy. V každé je pak tabulka s konkrétními výčtovými typy a jejich mapování na třídy a supertřídy.

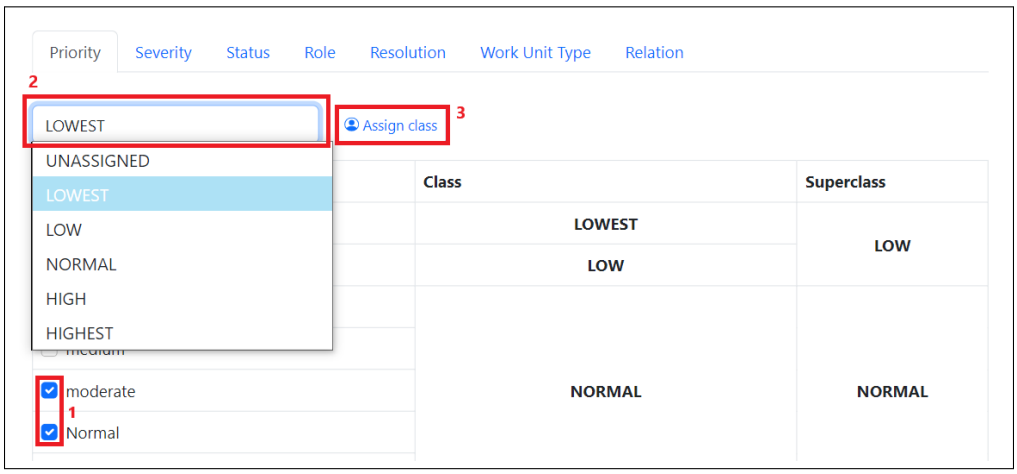
2. Výběr druhu výčtového typu, který budeme editovat. Výběr se provádí stisknutím záložky s název potřebného výčtového typu. Umístění prvků lze vidět na obrázku 17.



Obrázek 17: Záložky s druhy výčtových typů

3. Výběr výčtových typů, kterým chceme změnit mapování třídy na jednu konkrétní jinou třídu. To lze provést zaškrtnutím pole u vybraných výčtových typů.

4. Cílová třída výčtových typů lze nastavit v rozbalovacím seznamu nad tabulkou. Změny se poté potvrdí stisknutím tlačítka *Assign class*. Na obrázku 18 lze vidět rozmístění prvků. Výsledkem je změna mapování, která je v tabulce jasně viditelná.



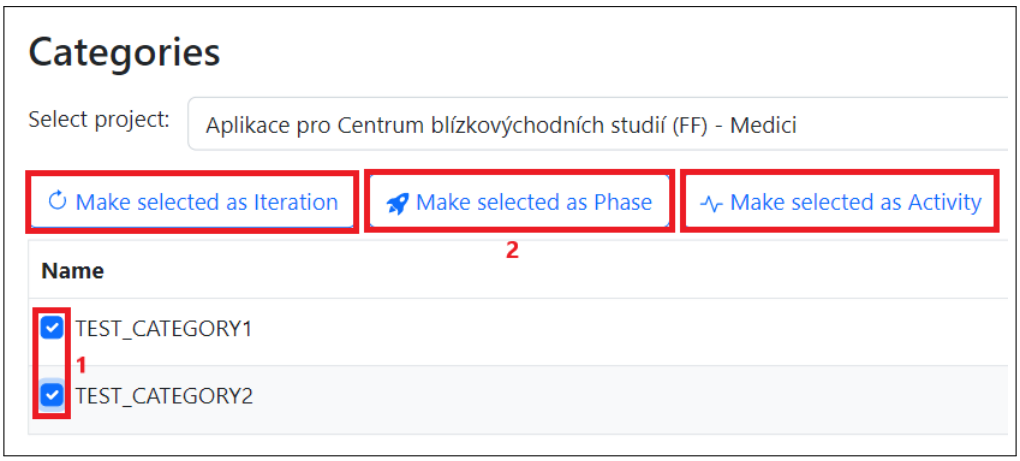
Obrázek 18: Rozmístění ovládacích prvků u výčtových typů

C.5 Transformace kategorií

Kategorii lze změnit na iteraci, fázi nebo aktivitu v případě, že existují work unity, ve kterých se změna projeví. V případě že tyto cílové work unity mají již přiřazenou iteraci, fázi nebo aktivitu, není možné transformaci u těchto úkolů provést. Návod pro transformaci je uveden v následujících krocích:

1. Stisknutí tlačítka *Categories* na stránce `Manage`. Pokud dosud nebyl vybrán žádný projekt pro editaci, zobrazí se rozbalovací seznam s výběrem projektu, které je ukázáno na obrázku 12. Zde je potřeba učinit výběr. Poté se zobrazí tabulka s kategoriemi a tlačítka, která slouží k transformaci na jednotlivé entity. Tato tlačítka jsou také u každého záznamu v tabulce a umožňují rychlou transformaci jedné kategorie.

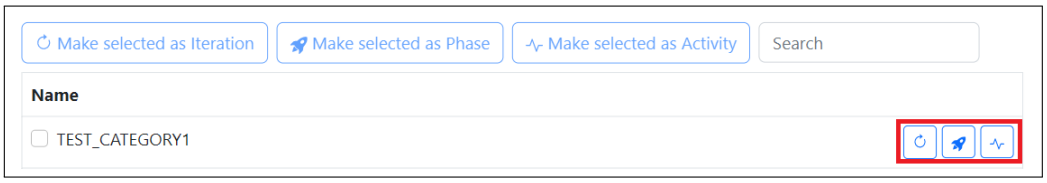
2. V případě transformace více kategorií je nutné vybrat kategorie pomocí zaškrtávacího políčka u každého záznamu. Poté se stiskne tlačítko v horní části obrazovky dle toho, jakým způsobem má být kategorie transformována. Postup naznačen na obrázku 19.



Obrázek 19: Transformace více kategorií

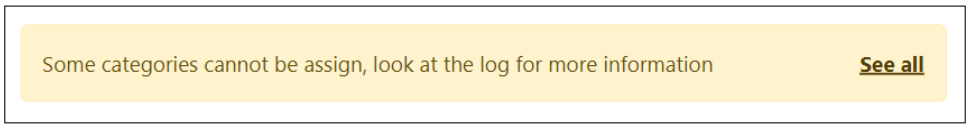
   Pokud se transformace týká pouze jedné kategorie, je možné pouze stisknout jedno ze tří tlačítek u daného záznamu. Ikony těchto tlačítek odpovídají pořadím i ikonami tlačítkům nad tabulkou a je tedy možné rozpoznat, které z nich stisknout. Pozice ovládacích prvků lze vidět na obrázku 20.

3. Po stisknutí tlačítka je výsledek operace oznámen pomocí alertu a jeho zbarvení vypovídá o úspěšnosti/neúspěšnosti operace. V případě, že některé kategorie nesplňují požadavky na úspěšnou transformaci,



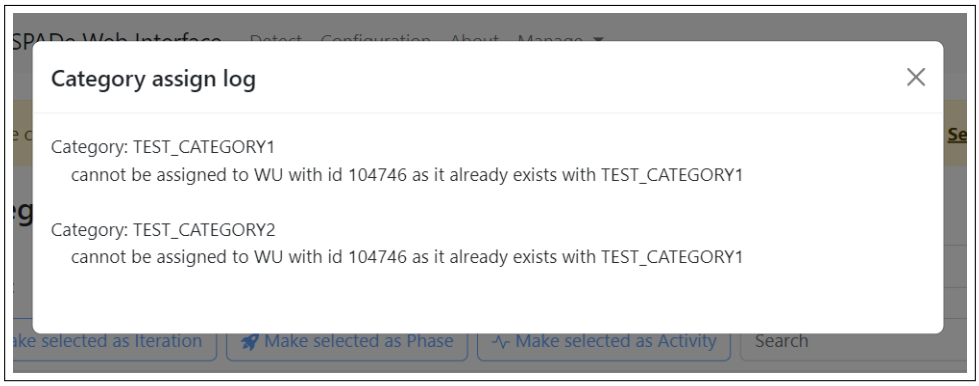
Obrázek 20: Transformace jedné kategorie

zobrazí se alert oranžové barvy, který má v pravé části tlačítko *See all*. Ten je zobrazen na obrázku 21.



Obrázek 21: Alert oznamující částečné selhání úlohy

Po stisknutí tlačítka se zobrazí dialogové okno, ve kterém je přesně popsáno jaké kategorie nemohli být transformovány a proč. Dialogové okno lze vidět na obrázku 22.

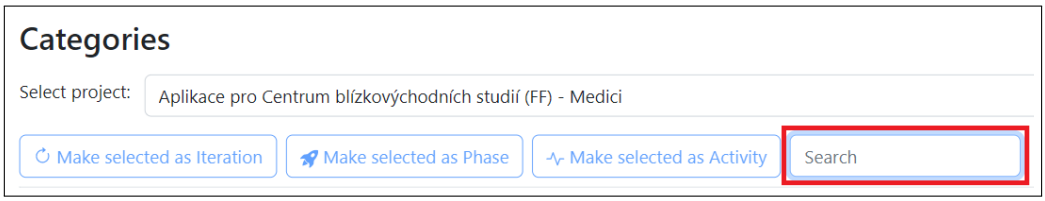


Obrázek 22: Dialogové okno s popisem transformací

4. Původní kategorie zůstane v tabulce, není tedy odstraněna.

Vyhledávání kategorií

Aby bylo možné vybrat z velkého množství kategorií ty správné, je možné je vyfiltrovat zadáním hledaného řetězce do vyhledávacího pole, které je viditelné na obrázku 23. Při zadání se vyfiltrují ty kategorie jejichž jméno odpovídá zadanému řetězci. Pokud byl učiněn výběr před vyhledáváním, zůstane tento výběr aktivní.



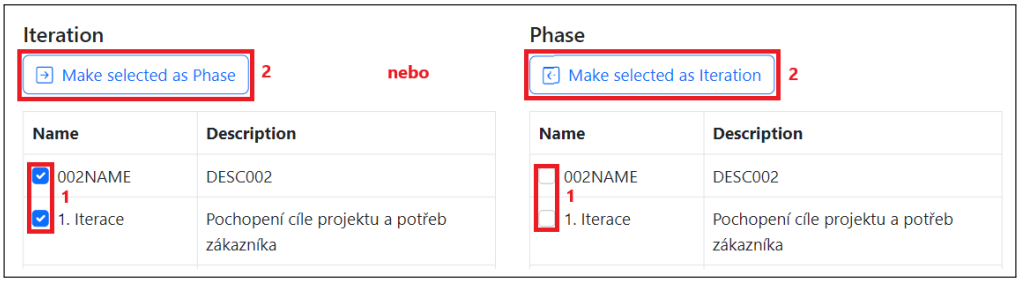
Obrázek 23: Vyhledávací pole pro filtraci kategorií

C.6 Změna iterací na fázi a opačně

Transformace iterací na fáze a opačně momentálně slouží především k editaci změněných kategorií, protože v aktuální verzi databáze jsou iterace i fáze shodné a není tedy důvod je měnit. Návod na změnu popsán v následujícím návodu:

1. Stisknutí tlačítka *Iterations, phases* na stránce `Manage`. Pokud dosud nebyl vybrán žádný projekt pro editaci, zobrazí se rozbalovací seznam s výběrem projektu, který je ukázán na obrázku 12. Zde je potřeba učinit výběr. Zobrazí se dvě tabulky, každá odpovídá jedné entitě.

2. Výběr iterací/fází z příslušné tabulky pomocí zaškrtávacího pole u každého záznamu a stisknutí tlačítka pro transformaci. Pozice ovládacích prvků jsou vidět na obrázku 24.



Obrázek 24: Ovládací prvky pro iterace a fáze

3. Úspěšnost/neúspěšnost provedené akce lze vidět v alertu, který se zobrazí v horní části stránky. Pokud nemohla být akce provedena, je možné stisknout tlačítko *See All*, které zobrazí z jakého důvodu nebyla pro některé vybrané záznamy akce dokončena. Především se tak stane v případě, že work unity, které používají daný segment projektu již mají přiřazený cílový segment v databázi nebo žádné work unity pro daný segment neexistují a je tedy zbytečné transformaci provést. Příklady výsledku jsou uvedeny v obrázkách 21 a 22.

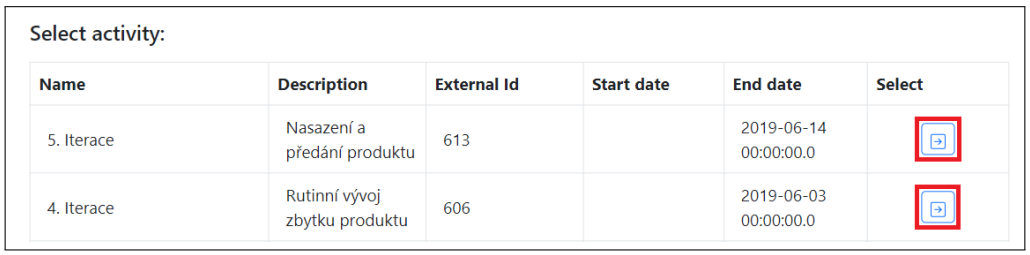
4. Pokud byla akce provedena úspěšně, lze vidět že byly vybrané segmenty transformovány a jsou nyní zařazeny v cílové tabulce. Tyto změny se také projeví v navázaných work unitech a původní fáze nebo iterace jsou smazány.

C.7 Přiřazení aktivit k work unitům

Aktivity lze přiřazovat k úkolům dle následujícího postupu:

1. Stisknutí tlačítka `Activities` na stránce `Manage`. Pokud dosud nebyl vybrán žádný projekt pro editaci, zobrazí se rozbalovací seznam s výběrem projektu, jak je ukázáno na obrázku 12. Zde je potřeba učinit výběr.

2. Po výběru projektu je zobrazena tabulka s aktivitami a zde je nutné pomocí tlačítka v pravné části tabulky vybrat tu aktivitu, která bude následně přiřazena work unitům. Situace je zobrazena na obrázku 25.



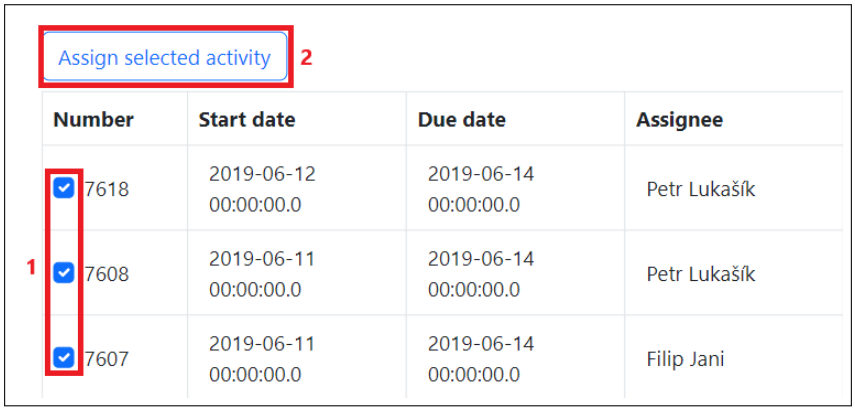
Obrázek 25: Výběr aktivity pro přiřazení k work unitům

3. V momentě kdy je vybrána aktivita, zobrazí se tabulka s work unitami a panel s filtrací. V tomto panelu je možné filtrovat záznamy na základě kategorie nebo typu úkolu. Pro zobrazení pouze těch úkolů, které mají specifickou kategorii je nutné ji z rozbalovacího seznamu vybrat a poté kliknout na tlačítko pro přidání do filtru. Následně je provedena filtrace, ve které se objeví pouze ty záznamy, které jsou označeny všemi přidanými kategoriemi. Z filtru lze odebírat stisknutím ikony křížku u každé kategorie.

4. Dále je možné filtrovat na základě typu úkolu. To lze také provést výběrem typu v rozbalovacím seznamu a kliknutím na potvrzovací tlačítko. Každá work unit má pouze jeden typ a přidáním další do filtru se tedy rozšiřuje množina záznamů, která bude v tabulce zobrazena. Odebrání typu z filtru se provádí stisknutím křížku u požadovaného typu. Panel filtrace s ovládacími prvky lze vidět na obrázku 26.

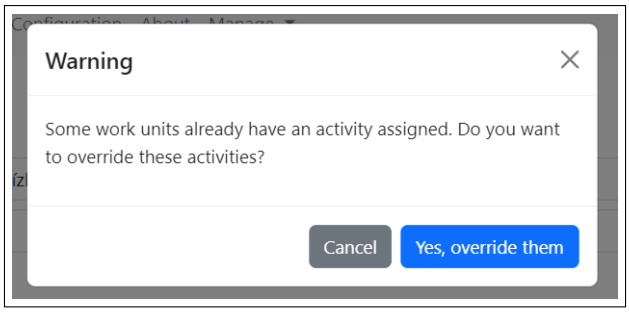
Obrázek 26: Ovládací prvky pro filtraci

5. Po filtraci je možné přikročit k výběr work unit, kterým má být aktivita přiřazena. To lze provést zaškrtnuím pole u každého záznamu v tabulce. Následně se stiskne tlačítko `Assign selected activity`. Ovládací prvky jsou zobrazeny v obrázku 27.



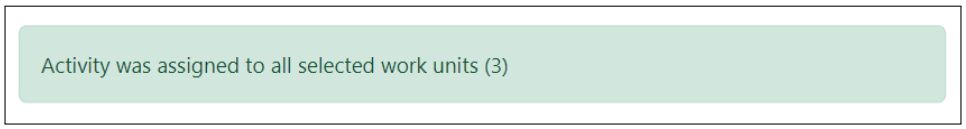
Obrázek 27: Přiřazení aktivity k work unitům

6. V případě, že má některá work unita již přiřazenou aktivitu, je o tom uživatel informován pomocí dialogového okna. Pokud je stisknuto tlačítko *Yes, override them* dojde k přepsání původních aktivit u work unitů. Tlačítko *Cancel* zruší prováděnou akci. Dialogové okno lze vidět na obrázku 28.



Obrázek 28: Dialogové okno potvrzující přepsání atributů

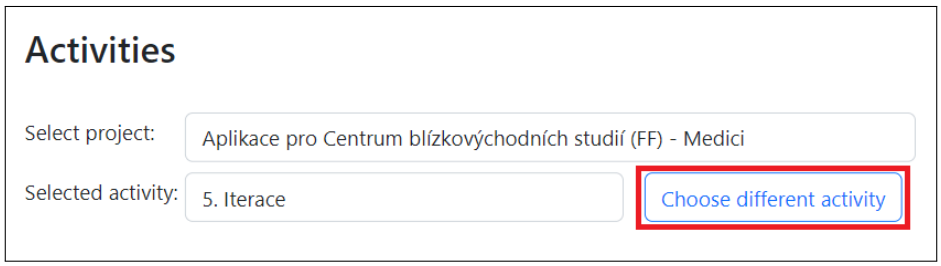
7. Pokud byl dokončen proces přiřazování aktivity, je zobrazen alert, který potvrzuje úspěšné dokončení prováděné akce. Alert lze vidět na obrázku 29.



Obrázek 29: Alert potvrzující dokončení akce

Změna aktivity

Kdykoliv je možné změnit vybranou aktivitu kliknutím na tlačítko u té momentálně vybrané. Dále je možné provést krok 2. Příklad lze vidět na obrázku 30.



Obrázek 30: Ovládací prvek pro změnu aktivity

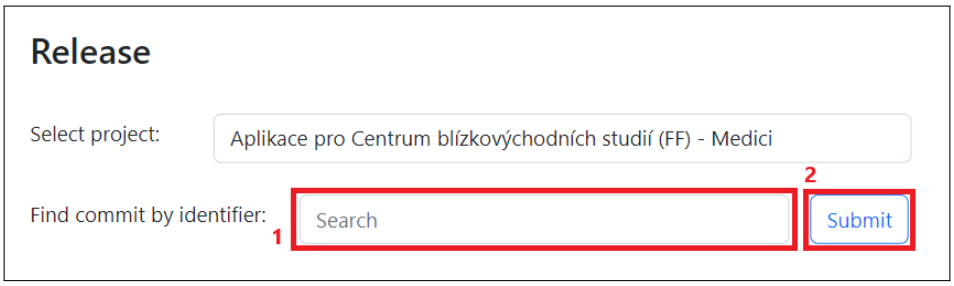
C.8 Změna tagu release u konfigurací

Manipulace s tímto tagem je popsána v následujícím návodu:

1. Stisknutí tlačítka *Manage releases* na stránce `Manage`. Pokud dosud nebyl vybrán žádný projekt pro editaci, zobrazí se rozbalovací seznam s výběrem projektu, který je ukázán na obrázku 12. Zde je potřeba učinit výběr. Poté se zobrazí vyhledávací pole pro konfigurace a seznam konfigurací vybraného projektu, který mají tag Release aktivní.

2. Zadání identifikátoru konfigurace do vyhledávacího pole a stisknutí tlačítka `Submit`. Lze vyhledat pouze konfigurace u kterých proběhl commit a které jsou přiřazeny aktuálně vybranému projektu. Ovládací prvky lze vidět na obrázku 31.

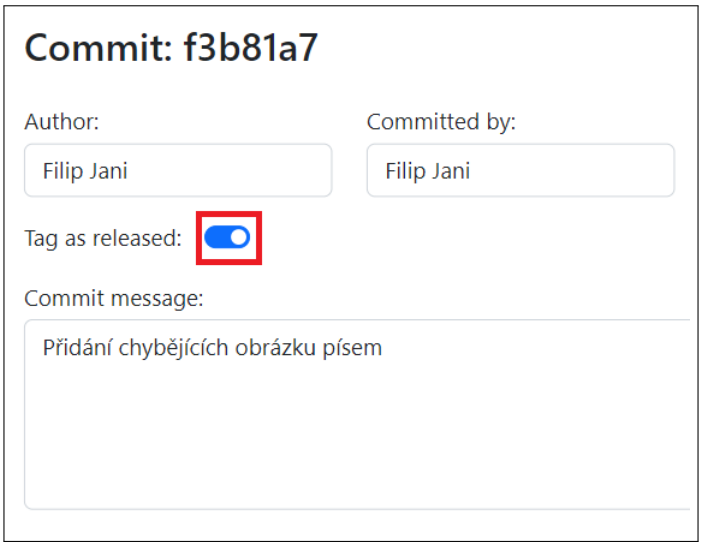
3. Po odeslání formuláře se zobrazí stránka s údaji o vybrané konfiguraci. Pokud nebyl identifikátor nalezen v databázi je o tom uživatel informován prostřednictvím alertu na obrázku 32.



Obrázek 31: Vyhledávací pole pro konfigurace

Obrázek 32: Alert informující o neexistující konfiguraci

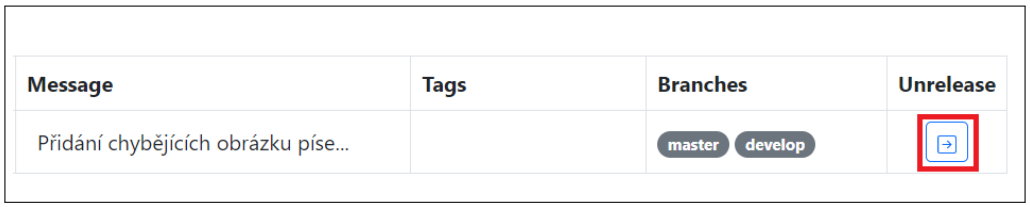
4. Přepnutí přepínače s popisem *Tag as released* do polohy, která odpovídá požadovanému úkonu. Pokud je přepínač zapnut, konfigurace má nastavení tag Released. Přepnutí je znázorněno na obrázku 33.



Obrázek 33: Přepnutí tagu Release u konfigurace

Odebrání tagu z konfigurací v hlavním menu funkce

Pro odebrání tagu u konfigurací, které ho již mají přiřazený je možné využít tlačítka v tabulce ve sloupci *Unrelease*. Po stisknutí se a v databázi pouze změní příslušný atribut, nejedná se tedy o zveřejnění nebo jinou podobnou akci. Po dokončení akce se dostraní záznam konfigurace v tabulce. Ovládací prvky jsou zobrazeny na borázku 34.



Obrázek 34: Odebrání tagu Release

D Fyzický model databáze SPADe

Tato příloha obsahuje celý fyzický model k nahlédnutí a je přiložena k výstisku bakalářské práce nebo je možné ji najít v ZIP souboru, který je popsán v příloze A.