FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Efektivní sběr a zpracování zpětné vazby ve službách DSW
Student:	Bc. František Štěpánek
Vedoucí:	Ing. Marek Suchánek, Ph.D. et Ph.D.
Studijní program:	Informatika
Obor / specializace:	Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2024/2025

Pokyny pro vypracování

Data Stewardship Wizard (DSW) je nástroj pro efektivní plánování správy dat ve vědeckých projektech. Uživatelé vyplňují dotazníky vytvořené na základě tzv. znalostních modelů, které definují strukturu dotazníků a doplňují další informace pro usnadnění odpovídání. Již nyní mohou uživatelé dávat jednoduše zpětnou vazbu na otázky v dotaznících, ale tato zpětná vazba je velmi jednoduchá (pouze text) a vytváří issue v nakonfigurovaném GitHub repozitáři, což není z různých pohledů vhodné a udržitelné. Cílem této práce je navrhnout novou službu pro sběr a zpracování zpětné vazby z DSW a souvisejících služeb.

- Analyzujte a popište DSW a související služby, zaměřte se na komponenty a funkcionality týkající se zpětné vazby a detailně analyzujte současný mechanismus pro zpětnou vazbu, zachyťte aktuální problémy.
- Proveďte rešerši tématu sběru a zpracování zpětné vazby v obdobných systémech i na teoretické úrovni (po technické i uživatelské stránce). Zaměřte se mimo jiné na to, jak motivovat uživatele poskytovat zpětnou vazbu, omezit duplicitní zpětnou vazbu a jaké metody zpracování jsou nejefektivnější.
- Sestavte požadavky na novou službu v rámci DSW ekosystému, která bude umět sbírat zpětnou vazbu z jiných služeb a poskytovat ji jiným službám pomocí API. Dále služba musí umožňovat zpracování zpětné vazby přímo v rámci služby (např. odezva či označení jako vyřešené). Služba musí být technologicky i vizuálně sladěná s existujícími DSW službami.
- Navrhněte architekturu, API a uživatelské rozhraní nové služby. V návrhu zohledněte

---

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 15. června 2023 v Praze.

budoucí rozšiřitelnost a integrovatelnost v jiných (i budoucích) službách.
- Implementujte službu dle návrhu a řiďte se osvědčenými postupy a zkušenostmi z DSW a dalšími pro zvolené technologie (dokumentace, CI/CD, příprava pro nasazení pro Docker apod.).
- Výslednou implementaci řádně otestujte pomocí automatizovaných testů i uživatelských či integračních testů.
- Zhodnoťte přínosy služby v kontextu předchozího řešení s GitHub Issues a popište další možný rozvoj a postup integrace služby v rámci DSW.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 15. června 2023 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Efektivní sběr a zpracování zpětné vazby ve službách DSW

Bc. František Štěpánek

Katedra softwarového inženýrství
Vedoucí práce: Ing. Marek Suchánek, Ph.D. et Ph.D.

7. května 2024

Poděkování

Chtěl bych poděkovat především vedoucímu mé práce Ing. Markovi Suchánkovi, Ph.D. et Ph.D., který mi pomohl s volbou tématu a v průběhu práce mi vždy dobře poradil a nasměroval mě správným směrem. Dále bych rád poděkoval své přítelkyni Mgr. Adéle Výborné za psychickou podporu při tvorbě práce a za její korekturu.

Prohlášení

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

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

V Praze dne 7. května 2024

Odkaz na tuto práci

Štěpánek, František. *Efektivní sběr a zpracování zpětné vazby ve službách DSW*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2024.

Abstrakt

Hlavním cílem diplomové práce je navrhnout a implementovat službu pro sběr a správu zpětné vazby, kterou bude možné integrovat do systému Data Stewardship Wizard. Služba musí umožňovat zaznamenání libovolné zpětné vazby, kterou uživatel požaduje. Součástí práce je rešerše tématu zpětné vazby, která zkoumá teoretickou i uživatelskou stránku problematiky. Výsledkem práce je nová služba pro sběr a správu zpětné vazby, která umožňuje uživatelům lépe vyjadřovat jejich potřeby a poskytovat kvalitnější zpětnou vazbu nejen na vědecké projekty v rámci DSW.

Klíčová slova Data Stewardship Wizard, Zpětná vazba, Serverová aplikace, JSON schéma, Haskell

Abstract

The main goal of the thesis is to design and implement a feedback collection and management service that can be integrated into the Data Stewardship Wizard system. The service must allow capturing any feedback requested by the user. The thesis includes a review of feedback topics, examining both theoretical and user aspects of the issue. The result of the thesis is a new feedback collection and management service that enables users to better express their needs and provide higher-quality feedback not only on scientific projects within DSW.

Keywords Data Stewardship Wizard, Feedback, Server application, JSON schema, Haskell

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Úvod

Data Stewardship Wizard (DSW) je nástroj pro efektivní plánování správy dat ve vědeckých projektech. Uživatelé vyplňují dotazníky vytvořené na základě tzv. znalostních modelů, které definují strukturu dotazníků a doplňují další informace pro usnadnění odpovídání. Již nyní mohou uživatelé poskytovat zpětnou vazbu na otázky v dotaznících, ale tato zpětná vazba je velmi jednoduchá (pouze text) a vytváří issue v nakonfigurovaném GitHub repozitáři, což není z různých pohledů vhodné a udržitelné.

Hlavním cílem práce je navrhnout službu pro sběr a správu zpětné vazby, kterou bude možné integrovat do systému DSW. Práce si klade za cíl, aby navržená entita zpětné vazby byla dostatečně flexibilní, aby bylo možné službu začlenit do komplexních aplikací, kde se zpětná vazba může vázat k různým entitám v různých kontextech.

Struktura práce vychází z úkolů, které bylo nutné postupně vykonat, aby bylo splněno zadání práce. Začíná kapitolou *Data Stewardship Wizard*, kde je čtenář seznámen s nástrojem DSW včetně stávajícího řešení sběru zpětné vazby. Poté následuje kapitola *Zpětná vazba*, která se zabývá rešerší tématu zpětné vazby po technické i uživatelské stránce. Na rešerši navazuje kapitola *Existující řešení*, která představuje stávající řešení v oblasti sběru a správy zpětné vazby. Jsou zahrnuta jak teoretická, tak i komerčně využívaná řešení. Práce pokračuje kapitolou *Požadavky*, která uvádí souhrn shromážděných požadavků včetně jejich kategorizace podle priorit. Další kapitolou v pořadí je *Návrh*, kde se nachází detailní návrh nové služby. Z návrhu vychází navazující kapitola *Implementace* zaměřující se na technický vhled do zdrojového kódu s vysvětlením, proč byla učiněna určitá rozhodnutí. Práci uzavírá kapitola *Rozšíření*, ve které se nachází popis dalšího možného rozvoje implementované služby.

Kapitola 1

Data Stewardship Wizard

Data Stewardship Wizard (DSW) je nástroj, který výzkumníkům a správcům dat umožňuje jednoduchou a efektivní tvorbu plánů správy dat (DMP) za dodržení FAIR principů. [1]

Datoví správci mohou snadno zachytit znalosti, včetně konkrétních dat požadovaných pro projekt, v tzv. znalostních modelech. Modely jsou následně přeměněny v dotazníky pro jednotlivé projekty, které výzkumníci vyplňují. Dotazníky vedou výzkumníky procesem s využitím doporučení, FAIR principů a zobrazují pouze relevantní otázky na základě předchozích odpovědí. [1]

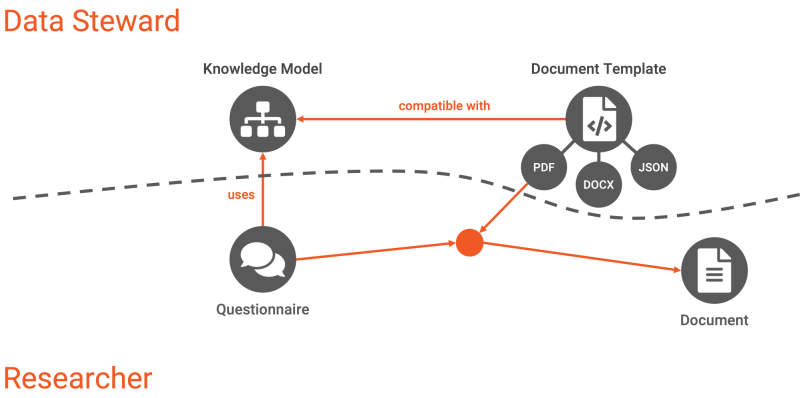
1.1 Komponenty DSW

Různé komponenty jsou v DSW propojené pro vytvoření plánu správy dat (DMP). Komponenty jsou vytvářeny a používány odlišnými uživatelskými rolemi. Správci dat (Data Stewards) pracují na přípravě obsahu, jako je znalostní model nebo šablona dokumentu. Ty jsou využity výzkumníky (Researchers) při vyplňování dotazníků a exportování dokumentů (například ve formátech PDF, DOCX nebo JSON). Každá komponenta v DSW má své vlastní UUID, které je vygenerováno systémem. Pomocí UUID je možné každou komponentu jasně a jednoznačně identifikovat. [1]

Propojení komponent je zachyceno na obrázku 1.1.

1.1.1 Znalostní model

Znalostní model (Knowledge Model) slouží jako šablona pro dotazník. Není to však lineární šablona, ale spíše šablona stromové struktury, kdy výběr jednotlivých větví závisí na předchozích odpovědích. Ve výsledném dotazníku se proto objeví pouze specifické otázky a ne všechny, které jsou definované ve znalostním modelu. Správa znalostních modelů je svěřena správcům dat. Pokud je ve znalostním modelu provedena změna, je vytvořena jeho nová verze, přičemž předešlá verze je zachována. Znalostní model tak obsahuje všechny své předešlé verze, což slouží ke zpětné kompatibilitě a k zotavení v případě zanesení chyby. [1]



Obrázek 1.1: Komponenty DSW a uživatelské role [1]

1.1.2 Dotazník

Dotazník je součástí projektu, kde výzkumníci odpovídají na otázky týkající se jejich výzkumu. Dotazník používá konkrétní znalostní model pro definování své vlastní struktury. Pokud ve zdrojovém znalostním modelu byly definovány fáze, obsahuje dotazník informaci o tom, v jaké fázi se aktuálně nachází. S ohledem na fázi může být například prioritnější získat odpovědi na konkrétní otázky, aby bylo možné přejít do další fáze. Dalším elementem dotazníku jsou kapitoly, které zjednodušují orientaci a zlepšují přehlednost v otázkách. Kapitola obsahuje informaci o počtu ještě nezodpovězených položek a umožňuje jednoduchý přístup ke konkrétním otázkám. Hlavním elementem jsou samotné otázky. Vedle samotné odpovědi je možné *přidat do TODO*, *přidat komentář* nebo poskytnout *zpětnou vazbu* k položené otázce. [1]

1.1.3 Šablona dokumentu

Znalostní modely určují to, jaká bude podoba dotazníku, ale nespecifikují, jak bude vypadat výsledný dokument. K tomu slouží šablony dokumentu. Šablony dokumentu převádějí odpovědi z dotazníků do dokumentů v libovolných formátech jako například PDF, MS Word nebo RDF. Díky tomu stačí vyplnit dotazník pouze jednou a výsledek vyexportovat v různých formátech za pomocí různých šablon. [1]

1.1.4 Dokument

Dokumenty vznikají z dotazníků a šablon dokumentu. Šablony dokumentu rozumí struktuře znalostního modelu a ví, jak transformovat odpovědi z dotazníku do specifického dokumentu v požadovaném formátu. Dokumenty jsou uloženy uvnitř projektu, ve kterém byly vytvořeny. [1]

1.1.5 Projekt

Projekty jsou centrální částí DSW. Každý projekt je založen na konkrétním znalostním modelu a využívá jednu nebo více dokumentových šablon pro ge-

nerování dokumentů. Jeho hlavní částí je dotazník 1.1.2, kde výzkumníci odpovídají na otázky. Jakákoliv interakce s dotazníkem je zaznamenána a uložena do historie projektu, takže je možná sledovat, kdo a kdy odpověděl na konkrétní otázku nebo kdy došlo ke změně předešlé odpovědi. Těchto změn může být mnoho, proto projekt nabízí možnost pojmenovat aktuální verzi a následně vyhledávat jen v pojmenovaných verzích. [1]

1.2 Architektura DSW

Data Stewardship Wizard se skládá ze 5 hlavních částí – serverová aplikace, úložiště dat, klientská aplikace, document worker a mailer. Pro účely této práce nejsou poslední dvě zmíněné části (document worker a mailer) relevantní, a proto se jimi práce dále nezabývá. Všechny části vyvíjené pro DSW aplikaci jsou open-source projekty dostupné ve veřejném GitHub repozitáři `github.com/ds-wizard`.

1.2.1 Serverová aplikace

Serverová část DSW je implementována v programovacím jazyce Haskell. Jedná se o pokročilý, čistě funkcionální programovací jazyk, který je výsledkem více než dvacetiletého výzkumu a vývoje. Jelikož se jedná o open-source produkt s aktivní komunitou uživatelů, usnadňuje Haskell vývoj flexibilního, snadno integrovatelného a udržitelného softwaru. [6]

Pro implementaci serverové aplikace DSW v jazyce Haskell je použit webový framework Servant. Jedná se o sadu balíčků pro deklaraci webových API na úrovni typů a následné použití těchto deklarací pro implementaci serverů. [7]

Pro komunikaci mezi serverem a klientem je využito REST API (Representational State Transfer Application Programming Interface). Jedná se o architektonický styl pro návrh webových služeb, který se vyznačuje bezstavovým přístupem. API je založeno na práci se zdroji, kdy každý zdroj je jednoznačně identifikovaný pomocí URI (Uniform Resource Identifier). Komunikace probíhá pomocí standardních HTTP metod jako GET, POST, PUT a DELETE. [8] Data jsou reprezentována ve formátu JSON.

Zdroje nejsou dostupné pro veřejnost a pro přístup k nim je nutná autentizace. Základní autentizační metodou v serverové části aplikace je uvedení přihlašovacích údajů – přihlašovací jméno a heslo. Po úspěšné autentizaci je uživateli poskytnut uživatelský token, pomocí kterého je možné komunikovat se serverem bez opakovaného zadávání přihlašovacích údajů. Uživatelský token má serverem definovanou dobu, po kterou je platný, a po jejím vypršení je nutné provést opětovné přihlášení. Další autentizační metodou v serverové aplikaci DSW je použití API klíče. Vytvoření API klíče je dostupné pro autentizované uživatele. Při jeho vytváření je nutné zvolit, jak dlouho bude klíč platný. Jak název napovídá, API klíče slouží pro komunikaci klienta se serverem přes API.

1.2.2 Úložiště dat

Pro trvalé uchování dat je v DSW využit databázový systém PostgreSQL. Jedná se o open-source objektově relační databázový systém s vysokou odolností proti chybám, který podporuje relační datový model, transakce, dotazování, indexování, dědičnost tabulek a mnoho dalších funkcí. PostgreSQL lze

rozšířit o vlastní datové typy a operace, což poskytuje vývojářům flexibilitu. Pro dotazování a manipulaci s daty používá jazyk SQL (Structured Query Language). [9]

V DSW je nutné ukládat soubory, ať už se jedná například o Znalostní modely nebo Šablony dokumentů. Soubory jsou uloženy do S3 úložiště (Amazon Simple Storage Service). Jedná se o službu pro ukládání objektů, která nabízí škálovatelnost, dostupnost dat, bezpečnost a výkon. Zákazníci mohou využívat Amazon S3 k ukládání a ochraně libovolného množství dat pro různé účely, jako jsou webové stránky, mobilní aplikace, zálohování, obnovení a archivace. [10]

1.2.3 Klientská aplikace

Klientská aplikace poskytuje uživatelské rozhraní, které umožňuje komunikaci se serverovou aplikací pomocí REST API. Klientská aplikace je implementována ve funkcionálním jazyce Elm. Jedná se o deklarativní programovací jazyk určený pro vytváření grafických uživatelských rozhraní založených na webovém prohlížeči. Elm je čistě funkcionální jazyk a je vyvíjen s důrazem na použitelnost, výkon a robustnost. Hlavní výhodou jazyka je neexistence téměř žádných výjimek při běhu aplikace, což je zaručeno kontrolou statických typů při kompilaci kompilátorem Elm. [11]

Uživatelské rozhraní se liší v závislosti na roli, kterou uživatel nabývá. Přehled je zobrazený na obrázku 1.2.

- **Výzkumník (Researcher)** – Má přístup k Projektům, které mu byly sdíleny. Dále si může zobrazit znalostní modely, které však nemůže editovat.
- **Správce dat (Data Steward)** – Vedle pravomocí Výzkumníka má přístup k editaci a vytváření Znalostních modelů a Šablon dokumentů.
- **Administrátor (Admin)** – Má přístup ke všem částem aplikace. Na rozdíl od rolí Výzkumník a Správce dat má přístup ke všem Projektům, tedy nejen k těm, které mu byly sdíleny. Dále má přístup do částí, které se zabývají obecným nastavením aplikace, nastavením lokalizace a správou uživatelů.

	Researcher	Data Steward	Admin
Projects	✓	✓	✓ (all)
Knowledge Models	✓ (read-only)	✓	✓
Knowledge Models Editors		✓	✓
Document Templates		✓	✓
Settings			✓
Locales			✓
Users			✓

Obrázek 1.2: Přehled pravomocí v závislosti na uživatelské roli v DSW [1]

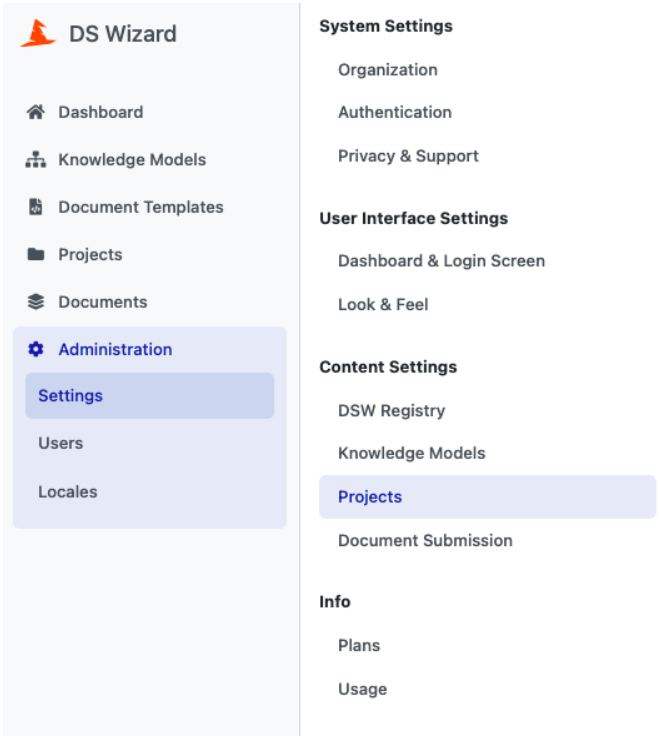
1.3 Zpětná vazba v DSW

U každého projektu, jak již bylo zmíněno, je možné poskytnout zpětnou vazbu k položené otázce (Feedback). Zpětná vazba v DSW je entita, která obsahuje nadpis (Title) a obsah (Content). Jejím hlavním účelem je, aby mohli uživatelé zaznamenat své připomínky k položené otázce. Ty slouží vývojářům nebo správcům dat k opravě nebo vylepšení otázek. Může se jednat o překlep, formulaci otázky nebo relevanci otázky v kontextu daného dotazníku.

Pro sběr a správu zpětné vazby se v DSW využívá webová platforma GitHub [12]. GitHub slouží pro správu verzí a spolupráci při vývoji softwaru za pomoci verzovacího nástroje Git [13]. Uživatelům, mimo mnoho jiných vlastností a funkcí, poskytuje možnost založení GitHub repozitáře. Repozitář slouží k ukládání a správě kódu týkajícího se jednoho konkrétního projektu. V repozitáři je možné vytvářet GitHub Issues, které slouží k sledování a řešení problémů, návrhů a úkolů v rámci repozitáře.

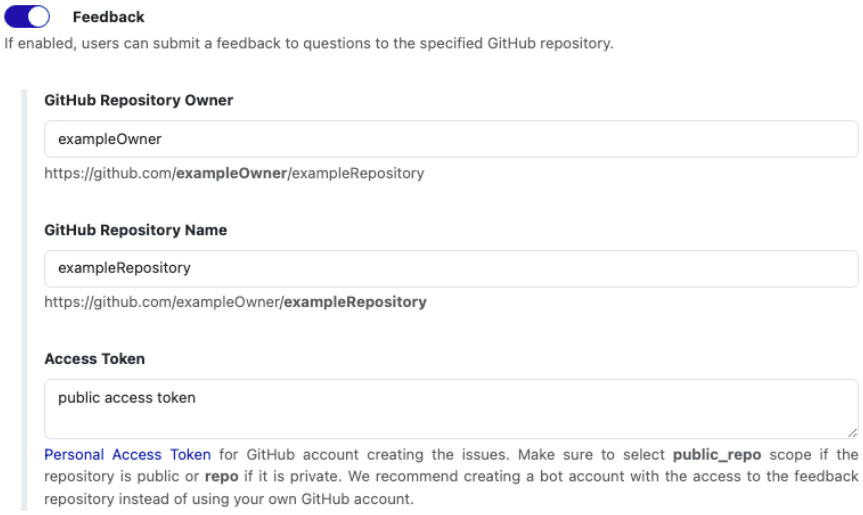
1.3.1 Zpětná vazba v klientské aplikaci

Zpětná vazba v klientské aplikaci je navázána na jednotlivé otázky z dotazníku. Aby bylo možné zadávat zpětnou vazbu, je nutné v administraci povolit její zadávání a nakonfigurovat potřebné informace pro spojení s GitHub platformou. Pro povolení je zapotřebí, aby administrátor v klientské aplikaci přešel do sekce Administration → Settings → Content Settings → Projects (stejně jako je na obrázku 1.3).



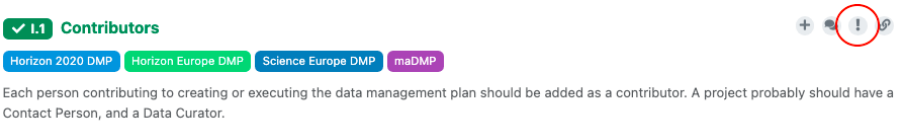
Obrázek 1.3: Sekce pro nastavení projektu v klientské aplikaci DSW

Zde je nutné povolit pole *Feedback* a vyplnit vlastníka GitHub repozitáře (GitHub Repository Owner) a jméno samotného repozitáře (GitHub Repository Name), který bude využit pro sběr a správu zpětné vazby. Poté je potřeba vyplnit přístupový token, pomocí kterého bude DSW přistupovat do repozitáře. Nastavení v klientské aplikaci poté může vypadat například tak jako na obrázku 1.4.

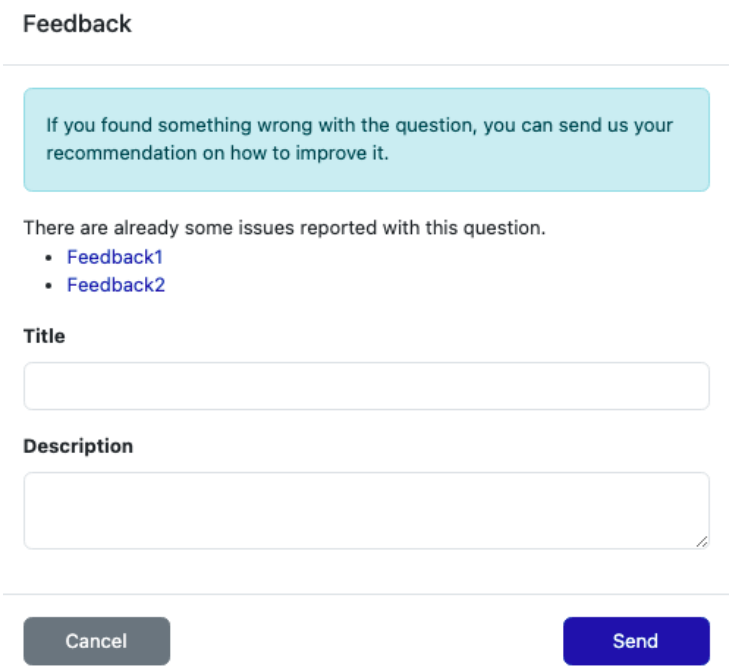


Obrázek 1.4: Povolení a nastavení zpětné vazby v klientské aplikaci DSW

Když je zpětná vazba nakonfigurována v administrátorské sekci, je možné ji vyplnit u libovolné otázky stisknutím tlačítka s ikonkou vykřičníku (zvýrazněno červeným kolečkem na obrázku 1.5). Po jeho stisknutí se zobrazí dialogové okno pro zadání názvu a popisu zpětné vazby. Pokud u otázky již byly nějaké zpětné vazby zadány, jsou jejich názvy při zadávání nové zpětné vazby zobrazeny ve formě seznamu, kdy jednotlivé položky slouží jako odkazy na konkrétní GitHub Issue v nakonfigurovaném GitHub repozitáři. Příklad dialogového okna pro zadání zpětné vazby je zobrazen na obrázku 1.6.



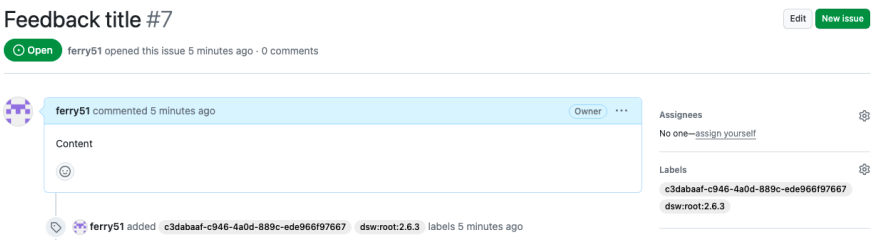
Obrázek 1.5: Otázka se zvýrazněným tlačítkem pro zadání zpětné vazby



Obrázek 1.6: Dialogové okno pro zadání zpětné vazby

1.3.2 DSW zpětná vazba v GitHub repozitáři

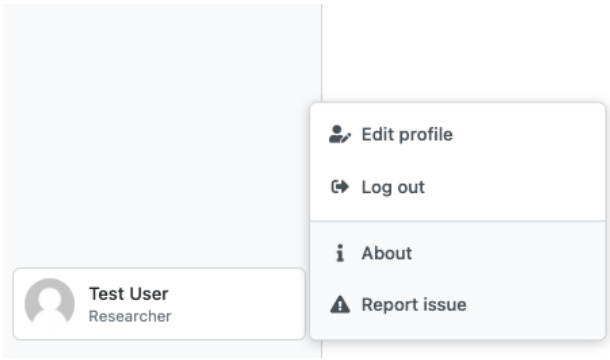
Jak již bylo zmíněno, pro sběr a správu zpětné vazby v systému DSW slouží GitHub repozitář. Po zadání zpětné vazby v klientské aplikaci DSW se v nakonfigurovaném GitHub repozitáři vytvoří Github Issue. Jeho název odpovídá zadanému názvu zpětné vazby a obsah je zaznamenán jako jeho komentář. Zároveň jsou k Issue přidány označení (Label) – první odpovídá identifikátoru, který je složený z identifikátoru organizace, identifikátoru Znalostního modelu, který byl použit pro sestavení dotazníku, a jeho verze, druhé označení odpovídá univerzálnímu identifikátoru (UUID) otázky. Příklad výsledného GitHub Issue je možné vidět na obrázku 1.7.



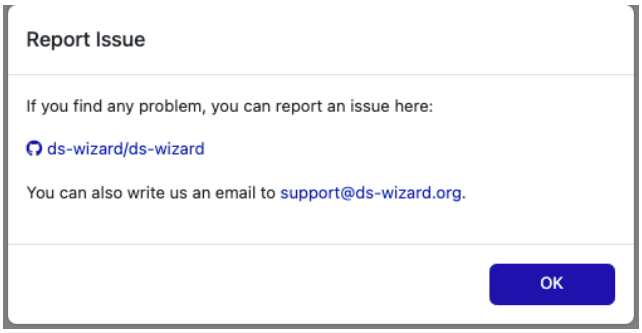
Obrázek 1.7: Zpětná vazba ve formě GitHub Issue

1.3.3 Hlášení chyb v klientské aplikaci

Hlášení chyb pomocí klientské aplikace je dostupné pro všechny přihlášené uživatele. Jedná se o způsob, kterým uživatel může zaznamenat připomínky k jakékoliv části DSW aplikace. Forma hlášení chyb je buď pomocí napsání e-mailu na poskytnutou e-mailovou adresu nebo nahlášení chyby na stránce podpory, na kterou je uživatel přesměrován po kliknutí na zobrazený odkaz. Pro zobrazení kontaktních informací (e-mail a odkaz na stránku podpory) je zapotřebí, aby uživatel přešel na ikonku se svým účtem, která se nachází v levé dolní části obrazovky, a poté vybral možnost *Report issue* (stejně jako je na obrázku 1.8). Poté se zobrazí dialogové okno s kontaktními informacemi, které je zobrazeno na obrázku 1.9.

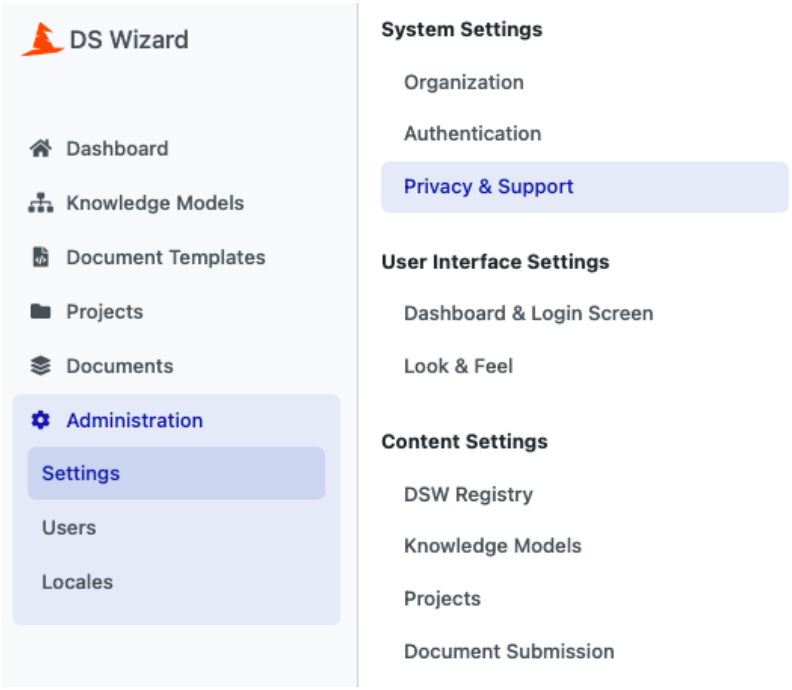


Obrázek 1.8: Sekce pro hlášení chyb



Obrázek 1.9: Dialogové okno pro hlášení chyb

Kontaktní informace jsou konfigurovatelné. Pro jejich správu je potřeba, aby administrátor přešel do sekce *Privacy & Support* (podle obrázku 1.10). Zde je možné nastavit kontaktní e-mail a odkaz na stránku podpory s nastavením ikony a textu odkazu. Ve výchozím nastavení je jako stránka podpory nakonfigurován GitHub repozitář, do kterého je možné přidat Issue s připomínkami.



Obrázek 1.10: Sekce pro nastavení soukromí a podpory v klientské aplikaci DSW

1.4 Nedostatky stávajícího řešení

Stávající řešení sběru a správy zpětné vazby s využitím GitHub Issues má jisté nedostatky. V první řadě se jedná o velmi jednoduchou entitu obsahující pouze název a textový popis. Uživatel není nijak motivován blíže specifikovat důvod, proč zpětnou vazbu zadává.

Dalším z nedostatků je absence informace o stavu zadané zpětné vazby. V klientské aplikaci DSW neexistuje prostředek, jak zjistit, jestli byl požadavek zpracován nebo byl například označen za irelevantní. U podobných požadavků (jako je zpětná vazba, chyby, atd.) je obvyklé vést informaci o statusu, který může nabývat hodnot například Nové, Rozpracováno nebo Hotové. Aktuálně, pokud je Issue v GitHub repozitáři vyřešeno, je v klientské aplikaci DSW při nejbližší synchronizaci odebráno ze seznamu zpětných vazeb u konkrétní otázky. Tento přístup za prvé není transparentní z pohledu uživatele, protože v DSW není žádná informace o tom, jak byla zpětná vazba zapracována a odkaz na propojené Issue se již nevyskytuje v seznamu záznamů. Za druhé se nemůže budoucí uživatel poučit z předešlých záznamů zpětné vazby od ostatních uživatelů, protože vyřešená Issue se již v seznamu nevyskytují. Za třetí to z technického hlediska vyžaduje periodickou synchronizaci a kontrolu stavu jednotlivých Issue v GitHub repozitáři, protože neexistuje integrace ze strany GitHub, která by při označení Issue jako vyřešené propagovala tuto informaci do systému DSW.

Obecně není ideální vysoká závislost na GitHub repozitáři. DSW nemá kontrolu nad dostupností webové platformy GitHub, kde se nachází nakonfigurovaný repozitář pro sběr a správu zpětné vazby. Dále je již zaznamenaná

zpětná vazba úzce svázána s aktuálním repozitářem. To znamená, že v případě, kdy je z jakéhokoliv důvodu potřeba změnit aktuální repozitář za jiný, dochází ke ztrátě všech GitHub Issues, které byly vytvořeny při zadávání zpětné vazby. V klientské aplikaci sice zůstane přehled záznamů zpětné vazby u každé otázky, ale odkazy vedoucí na GitHub Issues budou neplatné a uživatele přesměrují pouze na generickou GitHub stránku s návratovým kódem 404, který značí, že požadovaná stránka nebyla nalezena.

Na závěr kapitoly popisující DSW se nachází shrnutí klíčových nedostatků aktuálního řešení správy a zpracování zpětné vazby:

- **Vysoká závislost na platformu GitHub** – Pro poskytování zpětné vazby u otázek je nutnost nakonfigurovat GitHub repozitář, nad jehož dostupností nemá DSW žádnou kontrolu.
- **Decentralizovaný zdroj zpětné vazby** – Zpětná vazba je aktuálně navázána na konkrétní otázky a jediným dalším nástrojem pro poskytnutí zpětné vazby je pomocí nahlášení chyb, které nemusí využívat pro její správu stejné nástroje.
- **Nedostatečná komplexnost zpětné vazby** – Zpětná vazba obsahuje pouze název a popis, což není vždy dostačující pro zaznamenání kompletní informace, se kterou je možné následně pracovat.
- **Netransparentní propagace stavu zpětné vazby** – Není vedena informace o aktuálním stavu zpětné vazby. Při vyřešení je záznam pouze odstraněn ze seznamu bez jakýchkoliv navazujících akcí.
- **Nezobrazující se zkušenost ostatních uživatelů** – Vyřešené Issue se nezobrazují v seznamu záznamů zpětné vazby, z nichž by se mohl budoucí uživatel poučit, čímž by mohlo dojít ke snížení počtu duplicitních dotazů.
- **Hlášení chyb bez kontextu** – Všechny entity v DSW, kromě otázek v dotazníku, nemají vlastní způsob, jak k nim přímo přidat zpětnou vazbu. Jediným způsobem je využít obecné nahlášení chyb, kde však není automaticky zachycen kontext entity, se kterou je zpětná vazba svázána.

Kapitola 2

Zpětná vazba

Sběr a zpracování zpětné vazby od uživatelů je klíčová aktivita pro neustálé zlepšování a růst systémů. Poskytuje jedinečný pohled na silné a slabé stránky, umožňuje vývojářům činit rozhodnutí na základě informací získaných ze zpětné vazby a pomocí toho efektivně provádět iterativní vývoj. Dále pomáhá identifikovat oblasti ke zlepšení, odkrývat nové myšlenky a potvrzovat předpoklady. Využitím zpětné vazby od zákazníků mohou vývojáři zdokonalit svou práci, zlepšit uživatelský prožitek a v konečném důsledku dodat výjimečná softwarová řešení. [14]

2.1 Motivace pro získávání zpětné vazby

Zpětná vazba hraje klíčovou roli při vývoji softwaru. Nejedná se o průzkum spokojenosti zákazníků, jak je v některých zdrojích uváděno, ale zpětnou vazbou je jakákoliv informace, ať už pozitivní či negativní, kterou získáme od uživatelů o jejich zkušenostech s používáním softwaru. [15]

Jak již bylo zmíněno, zpětná vazba poskytuje jedinečný pohled na silné a slabé stránky. Jedná se především o takové stránky, které jsou pro uživatele nejzásadnější a mohou mu ztěžovat příjemné užívání softwaru. Pokud při vývoji nebudou zohledněny názory uživatelů, může dojít k tomu, že software nebude chtít nikdo používat, protože nebude jednoduše splňovat potřeby uživatele. Podle studie společnosti PWC [16] více než 73 % uživatelů uvádí jako důležitý faktor při rozhodování o využití softwaru názor ostatních uživatelů. [17] Z toho vyplývá, že pokud při vývoji není zohledněn názor stávajících uživatelů, může dojít ke ztrátě nejen aktuálních, ale i potenciálně budoucích uživatelů. Proto je žádoucí zpětnou vazbu sbírat, analyzovat a na základě toho provádět informovaná rozhodnutí při plánování budoucího vývoje.

Cíle a motivace pro sběr zpětně vazby mohou být různorodé a je potřeba si před samotným sběrem definovat, co od zpětné vazby očekáváme, které oblasti nebo oblastí se má zpětná vazba týkat a jak správně motivovat uživatele, aby zpětnou vazbu poskytovali. Cíle pro sběr zpětné vazby mohou být podle [18] například:

- **Vylepšení produktu a služeb** – Analyzováním požadavků cílových zákazníků je možné se přiblížit ideálnímu řešení. V průběhu času se však požadavky mění, a tudíž je důležité sbírat zpětnou vazbu, na základě

které je možné vylepšovat stávající řešení, aby lépe odpovídalo aktuálním požadavkům cílových zákazníků.

- **Měření spokojenosti** – Pro úspěšný produkt je zapotřebí, aby s ním byli jeho uživatelé spokojeni. Pokud spokojeni nebudou, pravděpodobně se pokusí najít alternativu, která bude lépe splňovat jejich požadavky. Přirozeně nejlepší způsob, jak zjistit spokojenost uživatelů, je se jich zeptat na jejich názor. Při použití otázek obsahujících hodnocení na číselné škále, je možné jednoduše vyhodnotit spokojenost zákazníků a následně předpovědět i možnou finanční kondici v budoucnosti.
- **Zpětná vazba je doceněna** – Samotný sběr zpětné vazby implikuje, že názory zákazníků jsou pro poskytovatele služeb a produktů důležité. Zároveň posiluje vztahy se zákazníky, kteří, za předpokladu že jsou spokojeni, mohou sami rozšiřovat povědomí o produktu pomocí kladných referencí a doporučení. Jedná se o nejefektivnější a zároveň nejlevnější způsob, jak získat nové zákazníky.
- **Věrohodný zdroj informací pro ostatní uživatele** – Nově příchozí uživatelé se velmi často rozhodují, zda využít či nevyužít danou službu na základě hodnocení od ostatních uživatelů. Pokud budou tato hodnocení pozitivní, zvyšuje to pravděpodobnost, že se nový uživatel rozhodne službu využít. V případě negativních hodnocení pravděpodobnost velmi klesá.
- **Podpora při rozhodování** – Rozhodovat se na základě neověřených odhadů není nejlepší přístup, jak efektivně vylepšovat a rozvíjet stávající řešení. Díky zpětné vazbě je možné provádět analýzy shromážděných dat a dělat informovaná rozhodnutí opřená o výsledky získané z analýz.

2.2 Motivace pro poskytování zpětné vazby

Z pohledu samotného uživatele nesmí být poskytování zpětné vazby náročné či nepříjemné. Pokud dojde k tomu, že uživatel nebude správně motivován, nebo bude dokonce demotivován k poskytování zpětné vazby, systém nebude mít dostatek dat, ze kterých by mohla být provedena dostatečně kvalitní rozhodnutí pro vylepšení stávajícího řešení. S tím úzce souvisí postupy získávání zpětné vazby a forma zpětné vazby, čemuž se věnují následující části.

Způsoby, jak lze uživatele motivovat, aby pravidelně poskytovali zpětnou vazbu, mohou být podle [19] následující:

- **Benefit nejen pro příjemce** – Vysvětlit uživateli, že z podávání zpětné vazby netěží jen strana příjemce, ale také strana poskytovatele, protože se mohou aktivně účastnit vývoje a jejich návrhy, potřeby a preference mohou být zahrnuty v nově implementovaných změnách.
- **Jednoduché a pohodlné** – Nabídnout uživateli více kanálů pro poskytování zpětné vazby, jako jsou například online průzkumy, e-maily, sociální média nebo chatboty, z nichž si uživatel může vybrat takový kanál, který je mu nejpříjemnější. Dále je doporučeno navrhnout dotazníky tak, aby byly jednoduché, dotazovaly se na relevantní věci, nezabíraly příliš času a nedotazovaly se na příliš informací najednou.

- **Motivace ve formě odměn** – Pro zvýšení motivace uživatelů je možné nabídnout za poskytnutí zpětné vazby slevy nebo omezený bezplatný přístup při používání služeb, kterých se zpětná vazba týká.

- **Pěstování kultury** – Zahrnuje vést pokračující dialogy s uživateli pro budování vztahů a vytvářet prostředí, kde je zpětná vazba nedílnou součástí zákaznických služeb. Získávat zpětnou vazbu nejen jednou za čas, ale vždy po vybraných uživatelských akcích, jako je například platba, interakce s podporou nebo využití služby. Dále je možné pozvat uživatele, aby se připojili k online komunitám, fórům nebo skupinám, kde se mohou podělit o své názory, zkušenosti a návrhy s ostatními uživateli.

- **Odpověď na poskytnutou zpětnou vazbu** – Při poskytnutí zpětné vazby je velmi žádoucí, aby poskytovatel obdržel odpověď a ujištění, že jeho zpětná vazba byla zaznamenána a že bude brána v potaz při budoucích rozhodováních. Při pozitivní zpětné vazbě může být odpověď ve formě poděkování a naopak při negativní zpětné vazbě může obsahovat omluvu, návrh řešení nebo kompenzaci za vzniklé problémy.

- **Analýza a následné využití** – Transparentně ukázat uživatelům, že na základě jejich zpětné vazby byly provedeny úpravy stávajícího řešení. V návaznosti na to je možné s uživateli komunikovat, že na základě jejich zpětné vazby byla učiněna rozhodnutí vedoucí k vylepšení produktu nebo služby.

2.3 Motivace pro zpracování zpětné vazby

Sesbíraná data ze zpětné vazby mohou být neúplná, nekonzistentní a někdy dokonce protichůdná. Proto je důležité data dobře analyzovat, aby bylo možné učinit informovaná rozhodnutí. Pro správnou a efektivní analýzu je potřeba si uvědomit, že existují různé skupiny zákazníků. Jejich seskupení podle frekvence používání může být velmi užitečné, protože každý zákazník má od služby svá vlastní očekávání. Po pečlivém přečtení dat je možné zakódovat zpětnou vazbu a odhalit tak společná témata. Důležitým aspektem je také naléhavost a opakování zpětné vazby. [20]

2.4 Běžné formy zpětné vazby

Před samotným získáváním zpětné vazby od uživatelů je potřeba definovat, co je konkrétním cílem a motivací pro sběr a zpracování zpětné vazby. Tím může být například design produktu, použitelnost, průzkum poptávky trhu, preference uživatelů, spokojenost uživatelů, návrhy na vylepšení a mnoho dalších. Dále je potřeba zvolit vhodné nástroje a postupy pro spojení se zákazníky a získání jejich názorů. Poté je potřeba zvolit jasné a nejlépe měřitelné metriky, podle kterých bude možné vyhodnotit dosažení nebo nedosažení vytyčených cílů. [15]

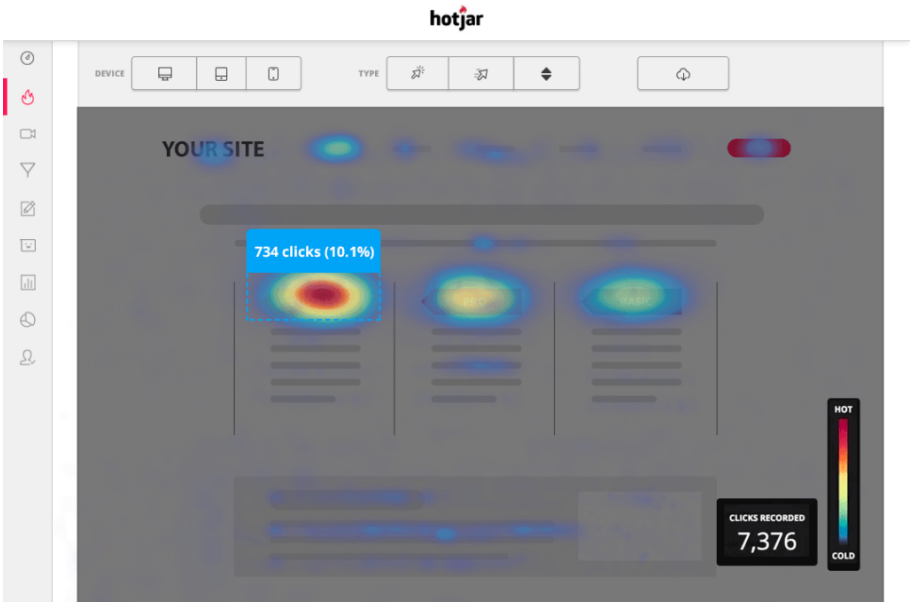
Podle cílů, nástrojů a metrik lze zvolit nejvhodnější formu zpětné vazby a postup pro její získání. Tato část se věnuje nejčastějším formám a jejich rozdělení na základě různých kritérií.

2.4.1 Nepřímá zpětná vazba

Z definovaných cílů může vyplynout, že není potřeba explicitně žádat uživatele o zpětnou vazbu, ale stačí sledovat a analyzovat jejich interakci při používání produktu. Taková zpětná vazba se nazývá nepřímá. Nepřímá zpětná vazba je vstup zákazníka, o který nebylo žádáno, ale existuje k němu přístup pro analýzu. [21] Například se jedná o již zmíněnou interakci uživatele se softwarovým produktem nebo komentáře na sociálních sítích.

Interakci se softwarovým produktem, například webovou stránkou, je možné zachytit pomocí analytických nástrojů, které umožňují analyzovat celou cestu uživatele, jeho akce, zjištěné problémy a často používané funkce. Shromážděné informace umožňují proaktivně vylepšovat softwarový produkt dříve, než dojde k nahlášení nedostatků stávajícího řešení samotným uživatelem. Nástroje pro záznam uživatelských relací, jako jsou například tepelné mapy, navíc nabízejí vhled do chování uživatelů a zvýrazňují takové části softwaru, které jsou pro uživatele nejpoutavější. [2]

Příklad tepelné mapy zachycující nejpoužívanější části webové stránky je zobrazen na obrázku 2.1.



Obrázek 2.1: Tepelná mapa používání komponent webové stránky [2]

2.4.2 Přímá zpětná vazba

Častějším druhem zpětné vazby je přímá zpětná vazba. Jedná se o takovou zpětnou vazbu, o kterou byl uživatel explicitně požádán. [22]

Umožnění poskytování přímé zpětné vazby je prvním krokem k budování pozitivních vztahů s uživateli. Pokud by neměli možnost, jak své připomínky zaznamenat, může dojít až k tomu, že přestanou produkt využívat. Přidáním

jednoduchého kontaktního formuláře nebo e-mailové adresy je uživatel vtažen do procesu a je s ním zahájen dialog – vše za účelem vylepšování produktu. Po poskytnutí přímé zpětné vazby by měl uživatel obdržet odpověď, která by ideálně neměla být automatická a generická, ale reagující na obsah poskytnuté zpětné vazby. Taková odpověď nemusí být dlouhá, ale měla by být doručena v relativně krátké době. [23]

2.4.3 Kvantitativní zpětná vazba

Kvantitativní zpětná vazba je měřitelná a číselně vyjádřitelná. Místo aby se zaměřovala na objevování příběhu nebo studium emocí a potřeb zákazníků, usiluje o získání čistých dat vyjádřených v číslech. Příkladem sběru kvantitativní zpětní vazby je NPS průzkum [24] se skóre od 0 do 10. [25]

Data ze sběru kvantitativní zpětné vazby mohou odpovídat na otázky [26]:

- Co?
- Kde?
- Jak?
- Kdy?
- Kdo?

Data nabývají číselných hodnot a jsou výsledkem uzavřených otázek. Z toho důvodu lze prohlásit data za objektivní. Pro objektivitu je zapotřebí velké množství dat, které vyloučí odchylku měření. Kvantitativní data slouží k potvrzení či vyvrácení hypotéz, které byly stanoveny před samotným sběrem dat. [26]

2.4.4 Kvalitativní zpětná vazba

Kvalitativní zpětná vazba představuje formu popisné zpětné vazby, která je získána od zákazníků a týká se určitého aspektu produktu či služby. Zachycuje subjektivní názory a pocity zákazníků týkající se různých aspektů nabízených služeb například prostřednictvím rozhovorů, kde zákazníci detailně popisují své potřeby a důvody, proč by měl být upraven proces zákaznické podpory. Kvalitativní zpětná vazba má dvě klíčové charakteristiky. Poskytuje náhled na slabé stránky produktu a pomocí kvalitativních metod lze nahlédnout na postoje zákazníků. Druhou klíčovou charakteristikou je, že na rozdíl od kvantitativní zpětné vazby je obtížně měřitelná. Výsledkem není číslo, které by určovalo spokojenost zákazníků nebo naléhavost nahlášeného problému. [25]

Sesbíraná kvalitativní data nejčastěji odpovídají na otázku *Proč?*, která je velmi důležitá. Jak již bylo zmíněno, není založena na číselných hodnotách, nýbrž na názorech a prožitcích. Z toho vychází, že jsou data z pohledu poskytovatele zpětné vazby subjektivní. Ke stanovení závěrů není potřeba takového množství dat jako u kvantitativní zpětné vazby. Výsledky ze zpětné vazby vedou k navržení hypotéz a nových nápadů, které mohou vést ke zlepšení stávajícího produktu. [26]

2.5 Běžné postupy získávání zpětné vazby

Před samotným získáváním zpětné vazby od uživatelů je potřeba si definovat, co je konkrétním cílem. Dále je potřeba si zvolit vhodné nástroje pro spojení se zákazníky a získání jejich názorů. Poté je podstatné zvolit jasné a měřitelné metriky, podle kterých bude možné vyhodnotit dosažení nebo nedosažení vytyčeného cíle. Podle cílů, nástrojů a metrik lze vybrat nejvhodnější formu zpětné vazby. Nakonec je možné zvolit nejlepší postup pro její získání. Následující části se věnují představení vybraných postupů.

2.5.1 Získávání přímé zpětné vazby

Přímá zpětná vazba je od uživatelů explicitně vyžádána. Způsob, kterým je získávána, se odvíjí od jejího typu. Mezi nejčastější způsoby patří například ankety, dotazníky, recenze, uživatelská testování nebo rozhovory. Některé ze zmíněných jsou vhodné především pro sběr kvalitativní zpětné vazby, ostatní pro sběr zpětné vazby kvantitativní.

- **Kvalitativní zpětná vazba**
  - **Uživatelské testování** – Přímé pozorování uživatelů při používání produktu a následná debata s nimi o jejich zkušenostech a názorech pro sběr kvalitativních dat. [27]
  - **Hlubinné rozhovory** – Strukturované rozhovory s uživateli, které umožňují podrobně prozkoumat jejich názory, potřeby a zkušenosti se zaměřením na produkt. [28]
  - **Ankety a dotazníky s otevřenými otázkami** – Vytvoření online anket nebo dotazníků, které obsahují otevřené otázky. Ankety lze distribuovat prostřednictvím e-mailu, webových stránek, sociálních médií nebo pomocí specializovaných platforem zaměřujících se na ankety a dotazníky. [29]
  - **Prototypování a testování uživatelských rozhraní** – Vytváření prototypů a jejich testování s uživateli pro získání reálné zpětné vazby před uvedením produktu na trh. [30]
  - **Zpětná vazba od zákazníků** – Sledování e-mailových zpráv nebo podaných reklamací od zákazníků.
- **Kvantitativní zpětná vazba**
  - **Skóre spokojenosti zákazníka (CSAT)** – Zákazník je požádán, aby ohodnotil spokojenost s produktem na škále od 1 například do 5. V takovém případě odpovídají číselné hodnoty následujícím odpovědím: 1 – velmi nespokojen, 2 – nespokojen, 3 – neutrální, 4 – spokojen, 5 – velmi spokojen. Výsledné skóre spokojenosti je vypočteno pomocí podílu součtu odpovědí s hodnotou 4 a 5 (ti, kteří byli spokojeni) a celkového počtu odpovědí. Pro vizualizaci hodnot může být využita škála zobrazená na obrázku 2.2. [3]
  - **Skóre pravděpodobnosti doporučení (NPS)** – Obdobně jako CSAT, NPS slouží ke měření spokojenosti zákazníka s produktem. Primárně se však zaměřuje na míru loajality zákazníka k produktu.

Zákazník je dotázán, jak by na škále od 0 do 10 doporučil produkt svým přátelům nebo kolegům. Na základě jejich odpovědi je možné zákazníky zařadit do jedné ze tří skupin: Promotéři (9–10), Pasivní (7–8) a Kritici (0–6). Výsledné skóre je vypočteno pomocí rozdílu procent Promotérů a procent Kritiků. Hodnotící škála pro NPS je zobrazena na obrázku 2.3. [3]

– **A/B testování**: Průběh metody A/B testování spočívá ve vytvoření několika testovacích variant, z nichž vždy původní varianta A (označována jako kontrolní) je zobrazena jedné polovině uživatelů a nová varianta B (označována jako vyzyvatel) je zobrazena polovině druhé. Poté, co testovací proces dosáhne požadované statistické významnosti, dochází k vyhodnocení obou variant podle zvolených kritérií, z nichž ta úspěšnější se stane variantou A, neboli kontrolní. V tomto okamžiku lze provést další iteraci testu. Vyhodnocení probíhá na základě kvantitativních dat, například podle počtu kliknutí nebo podle stráveného času u jednotlivých částí. [31]

Obrázek 2.2: Škála od 1 do 5 průzkumu spokojenosti CSAT [3]

Obrázek 2.3: Hodnotící škála NPS [4]

2.5.2 Získávání nepřímé zpětné vazby

Nepřímá zpětná vazba není od uživatele přímo vyžádána. Jedním ze zdrojů nepřímé zpětné vazby jsou data, která jsou výsledkem užívání produktu. Může se jednat například o to, kolik času stráví na dané stránce, jak často bylo určité tlačítko stisknuto nebo kolik uživatelů navštívilo konkrétní webovou stránku. Pro zachycení, analýzu a zobrazení takových dat existují specializované nástroje, pomocí kterých je jednoduché analyzovat nepřímou zpětnou vazbu od zákazníků. Jedním z nástrojů pro zobrazení shromážděných dat jsou tepelné mapy. Příklad tepelné mapy je zobrazen na obrázku 2.1.

Dalším zdrojem nepřímé zpětné vazby mohou být sociální sítě, recenze na specializovaných stránkách nebo hodnocení aplikace. O takovou zpětnou vazbu nebyl uživatel explicitně požádán, ale může se jednat o velmi přínosný zdroj, který pomůže odhalit slabé stránky nebo navrhnout nová řešení.

2.6 Cyklus zpětné vazby

Sběr, správa a analýza zpětné vazby není jednorázová akce. Jedná se o neustálou výměnu názorů mezi byznysem a zákazníky, která pomáhá vést produkt nebo službu k tomu, aby se stávala stále kvalitnější. Tento cyklus zpětné vazby má jediný cíl – podporovat kulturu, ve které je každý motivován sdílet své myšlenky upřímně a otevřeně. [32]

Jedním z cyklů zpětné vazby je takzvaný **A.C.A.F** cyklus. Zkratka pochází z anglických slov **A**sk (zeptat se), **C**ategorize (kategorizovat), **A**ct (jednat), **F**ollow-up (navázat) a každému odpovídá jedna ze čtyř fází cyklu. Obrázek A.C.A.F cyklu zpětné vazby je zobrazen na obrázku 2.4. [5]

První fází každé jednotlivé iterace cyklu je fáze **Ask**, tedy zeptat se. V této fázi je shromažďována zpětná vazba od zákazníků na základě cílů, které buď byly stanoveny byznysem, nebo vyplynuly z předešlé iterace. Uživatel musí být správně motivován, aby byl ochoten zpětnou vazbu poskytnout. K tomu může sloužit například motivace ve formě odměn (více v sekci 2.2). Dále je potřeba zvolit vhodnou formu zpětné vazby podle cílové skupiny a cílů (sekce 2.4), proč je zpětná vazba shromažďována. Detailní popis postupů, jak zpětnou vazbu shromažďovat, je popsán v sekci 2.5.

Ve druhé fázi **Categorize** probíhá kategorizace zpětné vazby, aby bylo možné zpětnou vazbu efektivně analyzovat. Kategorizace může být provedena na základě různých aspektů, například podle způsobu získání, zda je pozitivní či negativní nebo podle obsahu. Obecná motivace, proč zpětnou vazbu zpracovávat, je blíže popsána v sekci 2.3.

Třetí fáze s názvem **Act** je nejdůležitější fází celého cyklu. Zde dochází k akcím se shromážděnou zpětnou vazbou. Akce, které je podle [33] doporučeno podniknout, jsou následující:

- Poděkovat zákazníkům za pozitivní zpětnou vazbu.
- Poděkovat zákazníkům za průměrné hodnocení a doptat se, co by jejich zkušenost s produktem vylepšilo.
- Omluvit se, pokud měl zákazník negativní zkušenost s produktem a zeptat se, co konkrétně bylo špatně.

- Při nalezení problému informovat zákazníka, že se na problému pracuje, a poskytnout odhadovaný čas, kdy bude problém vyřešen.
- Pracovat na problémech nalezených uživateli a informovat je o stavu procesu jejich odstranění.

Poslední fáze **Follow-up** může být často opomíjena, což je velká chyba. Právě v této fázi dochází k uzavření celého cyklu zpětné vazby. Dochází zde k informování uživatelů o tom, že jejich zpětná vazba byla zohledněna při vývoji produktu. Bez toho by uživatel mohl nabýt dojmu, že jím poskytnutá zpětná vazba není vyslyšena, a tak by ji příště nemusel vůbec poskytnout. [33]

Obrázek 2.4: A.C.A.F. cyklus zpětné vazby [5]

Kapitola 3

Existující řešení

V této kapitole se nachází shrnutí současných řešení v oblasti shromažďování a využívání zpětné vazby uživatelů v softwarových systémech. Zaměřuje se na různé přístupy i známá praktická řešení, které slouží jako zdroj inspirace a základ pro výsledné navržené řešení.

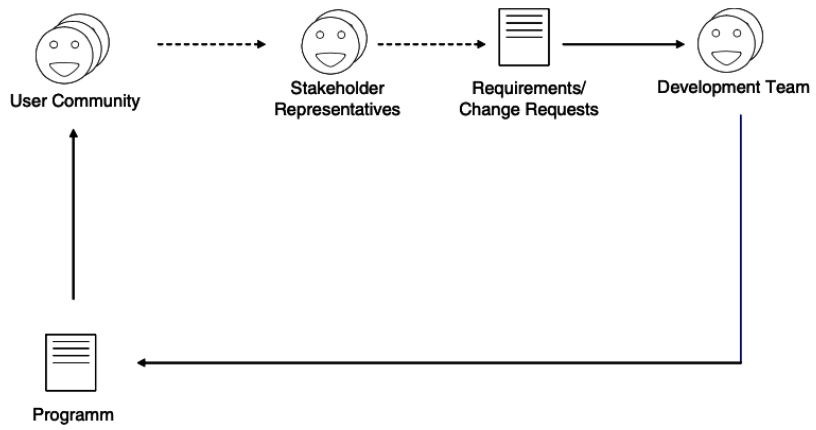
3.1 Leveraging Feedback (využití zpětné vazby)

Vývoj velkých softwarových systémů kritických pro podnikání často vyžaduje několik zlepšovacích cyklů. Je do toho zapojeno mnoho zainteresovaných stran a také celá řada rozsáhlých a složitých podnikových procesů. Přístup iterativního zlepšování může být nákladný a časově náročný z důvodu neefektivního sběru zpětné vazby. Zdrojem zpětné vazby jsou uživatelé, kteří ji předávají zástupcům zainteresovaných stran primárně verbální formou. Ti si musí klíčové požadavky, plynoucí ze zpětné vazby, zapamatovat, zaznamenat a následně předat vývojářům. Na konci tohoto toku, obsahujícího informace o zpětně vazbě, si musí vývojáři přečíst zapsané požadavky a implementovat vylepšení softwarového systému. V tomto toku dochází ke třem informačním transformacím: [34]

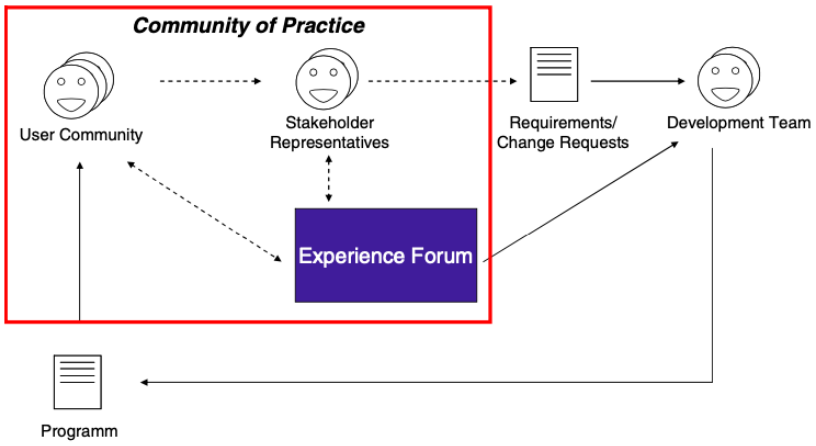
- **Zapamatování** – provádí zástupci zainteresovaných stran,
- **Zapsání** – provádí zástupci zainteresovaných stran,
- **Čtení** – provádí vývojáři.

Pokud jakákoliv z informačních transformací selže, pravděpodobnost uspokojení původního požadavku nebude příliš vysoká. [34] Tok zpětné vazby je zobrazen na obrázku 3.1.

Pro zkrácení toku zpětné vazby je navrhovaným řešením implementace Fóra zkušeností (Experience forum), které se podobá Základně zkušeností (Experience Base) [35]. Hlavním rozdílem mezi Fórem zkušeností a Základnou zkušeností je cílová skupina. Zatímco Základna zkušeností slouží ke sdílení poznatků v rámci vývojového týmu, Fórum zkušeností má za úkol je získat z uživatelské komunity a sdílet je nejen s vývojářským týmem, ale také se všemi ostatními uživateli. Integrace Fóra zkušeností do struktury toku zpětné vazby je zobrazena na obrázku 3.2. [34]



Obrázek 3.1: Tok zpětné vazby v SOA projektech



Obrázek 3.2: Integrace Fóra zkušeností do SOA projektů

3.2 Zlepšení hlášení chyb, detekce duplicit a lokalizace

Nestrukturovaný obsah ve formě přirozeného jazyka produkovaný uživateli jako součást zpětné vazby zahrnuje popis pozorovaného chování (Observed Behaviour – OB), kroky k reprodukci (Steps to reproduce – S2R) a normálního očekávaného chování softwaru (Expected behavior – EB). Takový obsah je softwarovými vývojáři považován za jednu z nejužitečnějších informací při pokusu o třídění a opravování chyb. [36]

Tyto informace bohužel často chybí, jsou neúplné, povrchní, dvojznačné nebo jsou příliš komplexní pro další využití. Téměř všechny (93,5 %) pozorované zprávy o chybách obsahují OB, ale jen jedna polovina z nich (51,4 %)

obsahuje EB a pouze jedna třetina (35,2 %) chybových hlášení má informace o S2R. [37]

Plánem tohoto výzkumu je poskytnout uživatelům softwaru akceschopnou zpětnou vazbu, aby byla zajištěna přítomnost OB, EB a S2R. Autoři výzkumu tvrdí, že je možné s vysokou přesností předpovědět přítomnost či nepřítomnost OB, EB a S2R při hlášení chyb, a proto jsou schopni upozornit uživatele na jejich nepřítomnost. Cílem výzkumu je také zlepšit přesnost lokalizace a detekce chyb založené na Text Retrieval [38] definováním, implementováním a vyhodnocením přístupu, který kombinuje různé typy obsahu vykazující specifické diskurzové vzory. [37]

3.3 Predikce závažnosti reportovaných chyb v open source softwarech

Stupeň závažnosti nahlášených chyb je jedním z kritických atributů pro efektivní plánování vývoje a údržby v open source softwarech. Tento atribut udává výši dopadu chyby na úspěšnou exekuci softwarového sytému a také, jak rychle je potřeba chybu opravit. Uživatelé při zadávání chybového hlášení poskytují vedle popisu i úroveň závažnosti. Jelikož je úroveň přiřazena ručně, může být její hodnota subjektivní a tedy neodpovídající reálné závažnosti. Ve spolupráci s tím, že velké softwarové projekty často dostávají vysoký počet chybových hlášení, je ruční přiřazení závažnosti náchylné k chybám. [39]

K vyřešení zmíněného problému byl provede výzkum predikce závažnosti chyb z chybových hlášení. Byla provedena komplexní mapovací studie hodnotící nedávné výzkumy ohledně automatické predikce závažnosti hlášení chyb, která kategorizovala různé aspekty experimentů uváděných v několika dokumentech. Výsledný souhrn vyzdvihuje relevantnost zkoumaného tématu a vědeckou vyspělost v rámci dané oblasti. Dále identifikuje prostor pro zlepšení, který může sloužit jako inspirace pro další výzkumy. Jako klíčové metody pro predikci závažnosti chyb se ukázalo použití metod nestrukturovaného textu, algoritmy strojového učení a metod text-miningu [40]. [39]

3.4 Technické řešení – Gitlab Issues

GitLab je webový Git repozitář poskytující zdarma veřejné i soukromé repozitáře, sledování problémů a wiki. Jedná se o kompletní platformu pro DevOps, která umožňuje profesionálům provádět různé úkoly v projektu – od plánování projektu a správy zdrojového kódu až po monitorování a zabezpečení. [41]

Pro zpracování zpětné vazby GitLab nabízí GitLab Issues. Tento nástroj umožňuje týmům efektivně spolupracovat a zjednodušit jejich pracovní postup. S GitLab Issues mohou uživatelé vytvářet, přiřazovat a sledovat problémy, které souvisí s řešením úkolů. Nástroj také nabízí funkce, jako jsou štítky, milníky, termíny splnění a nástěnky problémů, které pomáhají organizovat a prioritizovat práci. Šablony poskytují standardizovaný přístup k vytváření problémů, zatímco analytika poskytuje přehled o pokroku projektu. GitLab Issues umožňuje týmům přizpůsobit si jejich pracovní postupy a upravit je konkrétním potřebám. Lze jej integrovat do systému pomocí REST API. Díky schopnosti programově vytvářet problémy pomocí předdefinovaných šablon

mohou vývojáři zajistit konzistentí formu a snížit pravděpodobnost opomenutí důležitých detailů v popisech problémů. [42]

Gitlab byl vybrán jako reprezentant konkrétních komerčních technických řešení. Systémy jako GitHub, BitBucket (JIRA), YouTrack a Redmine fungují v podstatě podobným způsobem a podporují efektivní spolupráci a sledování problémů v softwarových vývojových projektech. Tyto platformy usnadňují hostování kódu, správu verzí a spolupracující pracovní postupy, což umožňuje pracovat na projektech více vývojářům současně a zároveň zajišťuje integritu kódu. Navíc zahrnují možnosti sledování problémů, které umožňují týmům dokumentovat, prioritizovat a řešit softwarové chyby, vylepšení a úkoly. Tyto systémy podporují transparentnost a zodpovědnost v životním cyklu softwarového vývoje a pomáhají týmům řídit projekty, sledovat pokrok a efektivně komunikovat, ať už se zaměřují na open-source projekty nebo na vývoj vlastního softwaru.

Kapitola 4

Požadavky

V této kapitole jsou definovány funkční a nefunkční požadavky na službu pro správu zpětné vazby. Definování požadavků je klíčové pro úspěšný provoz služby a slouží jako základ pro návrh, vývoj a testování. Cílem je zajistit, aby služba splňovala zadání práce a poskytovala požadovanou funkcionalitu.

Při definování požadavků je nezbytné umění prioritizovat. To pomáhá určit, které požadavky jsou klíčové pro úspěšné dokončení projektu a které mohou být řešeny později nebo dokonce vynechány. Pro prioritizaci byla zvolena metoda MoSCoW [43].

4.1 Metoda prioritizace MoSCoW

Metoda MoSCoW je čtyřkrokový přístup k prioritizaci. MoSCoW je zkratka pro **M**ust have (musí být), **S**hould have (mělo by být), **C**ould have (mohlo by být) a **W**ill not have (nemusí být). Písmena *o* vytvářejí akronym lépe vyslovitelný. [43]

- **M: Must have** (musí být) – První kategorie zahrnuje požadavky, které jsou nezbytné pro úspěšné dokončení projektu. Jedná se o nezbytné prvky, které poskytují minimální použitelnou podmnožinu požadavků. Pokud existuje způsob, jak konkrétní požadavek obejít, požadavek by neměl být zařazen do této kategorie. [43]
- **S: Should have** (mělo by být) – Druhá kategorie požadavků je o krok níže než Must have. Může obsahovat požadavky pro budoucí rozšíření bez dopadu na stávající projekt. Prvky, které by měly být obsaženy, jsou důležité pro dokončení projektu, ale nejsou nezbytné. Jinými slovy, pokud finální produkt neobsahuje požadavky, které by měly být obsaženy, stále funguje. Pokud však obsahuje prvky, které by měly být obsaženy, značně tím zvyšuje svou hodnotu. Drobné opravy chyb, vylepšení výkonu a nová funkcionalita jsou všechno příklady požadavků, které by mohly spadat do této kategorie. [43]
- **C: Could have** (mohlo by být) – Tato kategorie zahrnuje požadavky, které mají malý dopad, pokud jsou vynechány z projektu. Proto jsou požadavky, které by mohly být obsaženy, často prvními, které týmy deprioritizují – požadavky, které musí být obsaženy a které by měly být

obsaženy, mají vždy přednost, protože mají větší dopad na produkt. Příkladem požadavku, který by mohl být obsažen, je požadavek, který je žádoucí, ale není důležitý. [43]

- **W: Will not have** (nemusí být) – Poslední kategorie zahrnuje všechny požadavky, které tým považuje za nedůležité. Zařazení prvků do této kategorie má za následek posílení zaměření na požadavky ve zbývajících třech kategoriích a zároveň stanoví realistická očekávání ohledně toho, co finální produkt neobsahuje. Tato kategorie je velmi prospěšná, protože zabraňuje rozšiřování rozsahu během vývoje nad rámec toho, co tým předpokládal. [43]

4.2 Funkční požadavky

Softwarové a systémové inženýrství definují funkční požadavek jako funkci systému nebo jeho komponenty, kde je funkce popsána jako souhrn (nebo specifikace či výrok) chování mezi vstupy a výstupy. [44]

F1 Registrace služeb

Kategorie: Must have

Aplikace musí umět zaregistrovat externí služby, ze kterých bude přijímat zpětnou vazbu. Současně musí umět externí službu odebrat.

F2 Přístup

Kategorie: Must have

Aplikace musí poskytovat dvě úrovně přístupu:

- **Běžný uživatel** – Dokáže přistupovat k veřejnému API aplikace, které poskytuje informace o záznamech zpětné vazby včetně filtrování.
- **Administrátor** – Mimo oprávnění, kterých nabývá běžný uživatel, umí přidávat (registrovat) nové služby a odebírat služby stávající.

F3 Přijímání zpětné vazby

Služba musí umět přijímat zpětnou vazbu.

F3.1 Základní údaje

Kategorie: Must have

Zpětná vazba musí obsahovat název, popis a informaci o svém typu.

F3.2 Vlastní atributy

Kategorie: Must have

Zpětná vazba musí umožňovat definici vlastních atributů, které se mohou lišit v závislosti na externí službě a kontextu.

F3.3 Jazyk

Kategorie: Could have

Zpětná vazba by mohla obsahovat informaci o jazyku, ve kterém je poskytována.

F4 API

Kategorie: Must have

Aplikace musí poskytovat API rozhraní, pomocí kterého je možné přidávat, odebírat a filtrovat zpětnou vazbu.

F5 Operace se zpětnou vazbou

Kategorie: Should have

Aplikace by měla poskytovat rozhraní pro správu záznamů zpětné vazby, zahrnující reakce, přiřazení a změnu stavu.

F6 Statistiky a reporty

Kategorie: Could have

Aplikace by mohla poskytovat reporty a pokročilé statistiky o agregovaných záznamech zpětné vazby.

F7 Detekce duplicit

Kategorie: Could have

Aplikace by mohla umět omezit výskyt duplicit v záznamech zpětné vazby.

F8 Pravidla pro přiřazování

Kategorie: Will not have

Aplikace nemusí implementovat automatická pravidla pro přiřazování a reagování na záznamy zpětné vazby.

F9 Integrace s Issue trackery

Kategorie: Will not have

Aplikace nemusí být připravena na integraci s Issue trackery, jako jsou Jira, Github Issues, Bugzilla, Redmine a další.

4.3 Nefunkční požadavky

Nefunkční požadavky jsou soubor specifikací, které popisují operační schopnosti a omezení systému. Jedná se v podstatě o požadavky, které definují, jak dobře systém funguje, včetně aspektů jako rychlost, bezpečnost, spolehlivost, integrita dat nebo využití technologií. [45]

NF1 Technologie

Pro bezproblémovou integraci s DSW systémem je požadováno využít stejné technologie, které jsou využity v sytému DSW.

NF1.1 Serverová aplikace

Kategorie: Should have

Pro implementaci serverové části služby by měl být využit programovací jazyk Haskell [6] a webový framework Servant [7].

NF1.2 Databáze

Kategorie: Should have

Pro trvalé uložení dat by měl být využit stejný databázový systém, který je využit v aplikaci DSW. Jedná se databázový systém PostgreSQL [9].

NF1.3 Klientská aplikace

Kategorie: Should have

Klientská aplikace DSW je implementována v jazyce Elm [11], proto by měl být použit stejný jazyk pro implementaci klientské části služby.

NF2 Bezpečnost

Kategorie: Must have

Aplikace musí být zabezpečená, aby zdroje byly přístupné pouze autentizovaným a autorizovaným uživatelům.

NF3 Zdokumentované API pomocí OpenAPI

Kategorie: Should have

Je požadováno, aby veškeré veřejné rozhraní aplikace bylo důkladně zdokumentováno pomocí standardu OpenAPI [46]. Tato dokumentace by měla obsahovat kompletní specifikaci API včetně všech dostupných cest, parametrů, odpovědí a popisu datových struktur. Cílem je poskytnout jasný a uživatelsky přívětivý návod k použití API pro vývojáře třetích stran, což zlepšuje interoperabilitu a zpřístupňuje funkce aplikace pro integraci a využití ve vývojářském ekosystému.

NF4 Integrační testy

Kategorie: Should have

Požadavek vyžaduje, aby aplikace byla schopna úspěšně projít integračními testy. Testy mají ověřit, že různé komponenty aplikace spolupracují správně a že propojení mezi nimi funguje podle očekávání. Integrační testy by měly být navrženy tak, aby pokryly všechny důležité interakce mezi komponentami. Cílem je zajistit, že aplikaci je možné integrovat do dalších systémů a lze s nimi komunikovat efektivně a spolehlivě. Pro jednodušší testování je požadováno vygenerovat PostMan kolekci [47].

NF5 Nasazení přes Docker

Kategorie: Should have

Aplikaci by mělo být možné nasadit pomocí nástroje Docker [48]. Docker je open-source softwarová platforma používaná k vytváření, nasazování a správě virtualizovaných kontejnerů aplikací na společném operačním systému (OS), doprovázená ekosystémem přidružených nástrojů. [49]

4.4 Shrnutí

Z požadavků vyplývá, že aplikace bude samostatná služba, do které budou moci další služby odesílat zpětnou vazbu. Dále budou moci provádět operace se záznamy zpětné vazby. Souhrn požadavků i s kategorizací MoSCoW je zobrazen v tabulce 4.1.

Požadavek	Kategorie	Shrnutí
F1	M	registrace služeb
F2	M	přístup pro běžného uživatele a administrátora
F3.1	M	popis, typ a štítek ve zpětné vazbě
F3.2	M	vlastní atributy ve zpětné vazbě
F3.3	C	jazyk zpětné vazby
F4	M	veřejné API rozhraní
F5	S	operace se zpětnou vazbou
F6	C	statistiky a reporty
F7	C	snížení duplicit
F8	W	automatická pravidla pro přiřazování
F9	W	integrace s issue trackery
NF1.1	S	server pomocí Haskell a frameworku Servant
NF1.2	S	databáze pomocí PostgreSQL
NF1.3	S	klient pomocí Elm
NF2	M	zabezpečený přístup
NF3	S	dokumentace pomocí OpenAPI
NF4	S	integrační testy
NF5	S	nasazení pomocí Docker

Tabulka 4.1: Souhrnná tabulka požadavků

Kapitola 5

Návrh

Tato kapitola se zaměřuje na návrh implementace nové služby pro správu zpětné vazby. Cílem této části je detailně popsat architekturu a technické řešení navrhované služby včetně využitých technologií a struktury aplikace, které vychází ze shromážděných požadavků na službu. Dále jsou zkoumány různé aspekty návrhu, jako jsou databázové modely, uživatelské rozhraní a struktura serverové aplikace, které jsou klíčové pro funkčnost a výkonnost systému. Tato kapitola poskytne ucelený pohled na rozhodnutí ohledně technických aspektů, které tvoří základ pro úspěšnou implementaci systému.

5.1 Architektura

Architektura výsledné služby se bude řídit vícevrstvou architekturou se třemi vrstvami. Taková architektura je označována vícevrstvou proto, že funkčnost aplikace je rozdělena mezi několik vzájemně spolupracujících vrstev, které spolu komunikují přes definované rozhraní. Nejběžnějším příkladem vícevrstvé architektury je architektura třívrstvá, kterou používá mnoho webových aplikací. V takovém případě rozlišujeme vrstvu, která se stará o uživatelské rozhraní (**prezentační vrstva**), vlastní logiku aplikace (**aplikační vrstva**) a databázi (**datová vrstva**). [50]

Mezi důvody použití vícevrstvé architektury patří podle [51] následující výhody:

- **Oddělení zodpovědností** – Umožňuje oddělení různých aspektů aplikace do samostatných vrstev. Každá vrstva má svůj specifický účel a zodpovědnost, což usnadňuje správu a údržbu kódu.
- **Modularita a znovupoužitelnost** – Díky rozdělení aplikace do více vrstev je možné jednotlivé části aplikace vyvíjet a testovat nezávisle na sobě. To umožňuje větší modularitu kódu a snadnější znovupoužitelnost jednotlivých částí aplikace.
- **Škálovatelnost** – Vícevrstvá architektura umožňuje lepší škálovatelnost aplikace. Každá vrstva může být škálována samostatně podle potřeby, což umožňuje efektivnější využití zdrojů a zajištění výkonu systému při zvyšujícím se zatížením.

- **Flexibilita** – Díky oddělení různých aspektů aplikace do samostatných vrstev je snazší provádět změny nebo aktualizace v systému. Změny v jedné vrstvě nemusí nutně ovlivnit ostatní vrstvy, což zvyšuje flexibilitu a umožňuje rychlejší reakci na nové požadavky.
- **Bezpečnost** – Vícevrstvá architektura umožňuje implementaci různých bezpečnostních opatření na různých úrovních aplikace. Například autentizace a autorizace mohou být podle potřeby implementovány na úrovni vrstvy prezentační, aplikační nebo dokonce na úrovni vrstvy datové.
- **Správa chyb** – Každá vrstva může mít své vlastní mechanismy pro zachycení a správu chyb. To umožňuje lepší monitorování výkonu, stabilitu aplikace a rychlejší řešení problémů.

Celkově lze říci, že použití vícevrstvé architektury přináší mnoho výhod v oblasti správy kódu, ladění výkonu, bezpečnosti a flexibility softwarových systémů.

5.2 Serverová aplikace

Návrh serverové části aplikace zahrnuje specifikaci použitých technologií, struktury aplikace, API rozhraní, odpovědí, chybových hlášek a bezpečnosti serverového systému. Jedná se o implementaci aplikační vrstvy v třívrstvé architektuře.

5.2.1 Technologie

Požadavek NF1.1 udává, že serverová část aplikace by měla být implementována pomocí funkcionálního jazyka Haskell [6], za využití webového frameworku Servant [7]. Pro komunikaci s klienty bude využito HTTP protokolu a HTTP metod [8]. Podle požadavku NF5 bude aplikace připravena na nasazení do Dockeru, čehož bude docíleno vytvořením a nastavením Dockerfile a docker-compose souborů ve zdrojovém kódu. Pro splnění požadavku NF3 bude aplikace automaticky generovat OpenAPI specifikaci, za využití nástroje Swagger [52].

5.2.2 Struktura zdrojového kódu

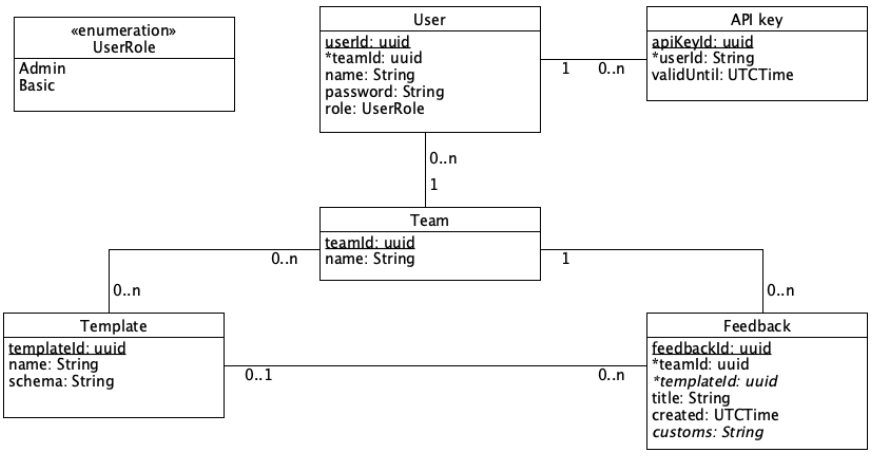
Serverová část bude implementačně rozdělena tak, aby jednotlivé části kódu byly správně oddělené podle svých zodpovědností. Serverová aplikace bude rozdělena minimálně na následující části:

- **API** – Obsahuje implementaci veřejného REST API rozhraní pomocí endpointů. Konkrétní seznam endpointů je možné zobrazit v sekcích 5.2.6 a 5.2.7.
- **Logika** – Poskytuje spojení mezi API a databází. Obsahuje veškerou logiku prováděnou s daty, jako je například validace.
- **Databáze** – Definuje databázový model a s tím spojené databázové operace s jednotlivými entitami, které jsou v ní obsaženy. Podrobnější popis jednotlivých entit se nachází v sekci 5.3.

- **Konfigurace** – Řeší přístup ke konfigurovatelným proměnným.

5.2.3 Model

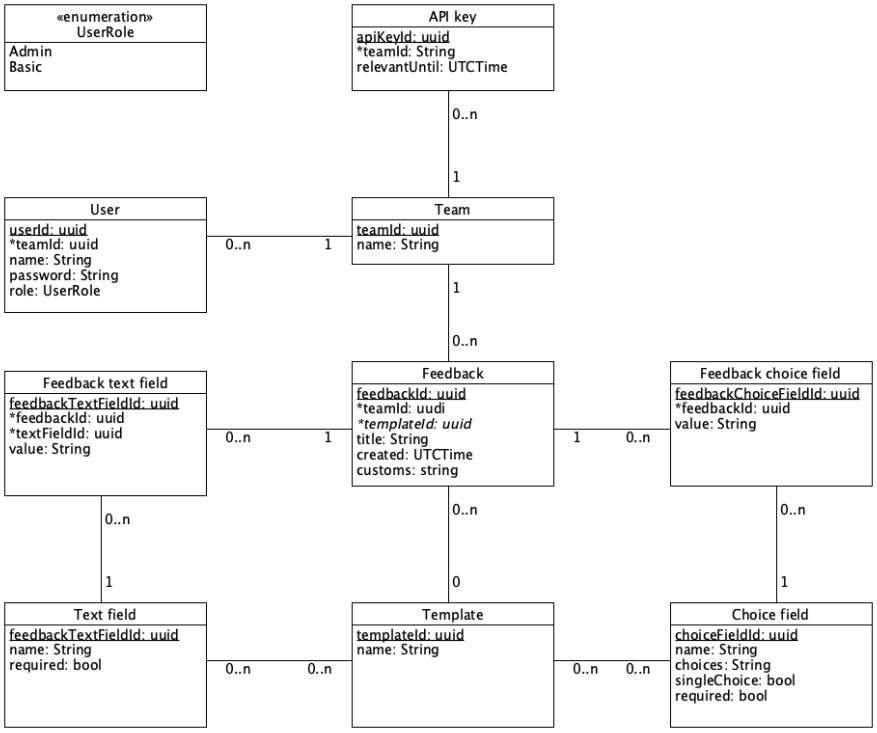
Při návrhu vznikly dvě varianty, jaké entity by se měly v řešení vyskytovat a jak by mezi sebou měly být provázány. První varianta je zobrazena na obrázku 5.1. Hlavním prvkem je samotná zpětná vazba (Feedback), která obsahuje informace, kdy byla zaznamenána, kým byla vytvořena, nadpis a následně libovolné další informace, které budou ve formátu JSON uloženy do atributu *customs*. Zpětná vazba může obsahovat identifikátor šablony (*templateId*), podle které by se měl obsah atributu *customs* řídit. Pomocí odpovídající šablony (Template) bude prováděna validace dat, konkrétně podle atributu *schema*. Každá zpětná vazba je navázána na konkrétní tým (Team), který má název a může mít libovolný počet členů. Tým reprezentuje libovolnou skupinu, například může zastřešovat jednu externí službu, a jelikož je zpětná vazba navázána na tým a ne na konkrétní uživatele, všichni uživatelé daného týmu budou mít přístup ke všem záznamům zpětné vazby spojené s týmem. U uživatele (User) bude uvedeno (mimo zmíněné příslušnosti k týmu) jméno, heslo a jeho role. Ta bude nabývat dvou hodnot – administrátor (Admin) a běžný uživatel (Basic). Pro přístup k datům může uživatel využít API klíč (API key), který má definovanou platnost.



Obrázek 5.1: Model entit (varianta č. 1)

Druhá varianta, zobrazená na obrázku 5.2, je komplexnější z pohledu entit a jejich provázání. Zpětná vazba se zde skládá z textových polí (Feedback text field) a polí výběru (Feedback choice field). Ty mají konkrétní hodnotu a následně informaci o tom, ke které zpětné vazbě patří a kterou šablonou textového pole, respektive pole výběru, se řídí. Rozdíl je také v samotné šabloně, která se skládá právě z šablon textových polí a šablon polí výběru. Šablona textového pole obsahuje jméno pole a informaci o tom, zda je povinná, nebo nikoliv. Šablona pole výběru oproti šabloně textového pole obsahuje navíc možnosti, ze kterých je možné vybrat, a zda je možné vybrat více možností. API klíč je

v této variantě navázán na konkrétní tým, jelikož jsou zpětné vazby navázány na tým a ne na uživatele.



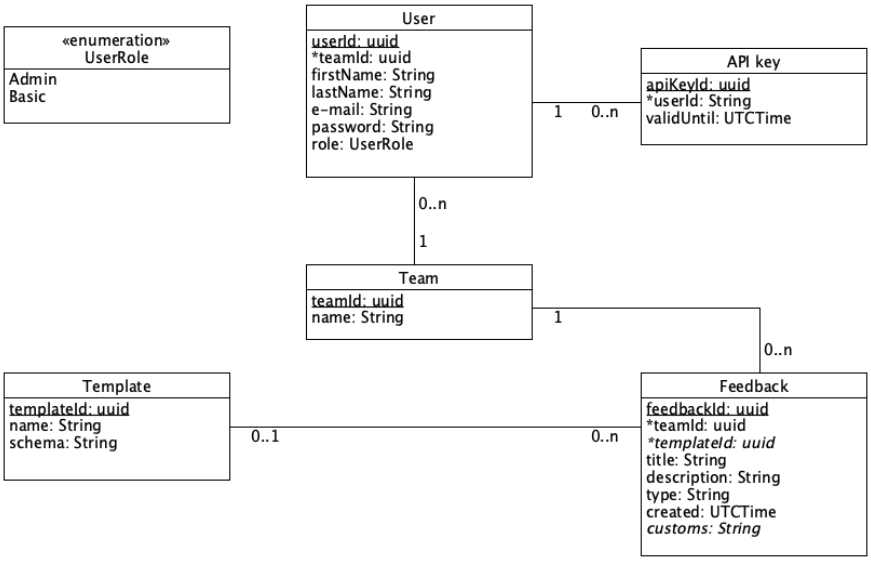
Obrázek 5.2: Model entit (varianta č. 2)

Obě dvě navržené varianty by umožňovaly základní implementaci podle požadavků, ale první varianta, i když se to na první pohled nemusí zdát, je flexibilnější, použitelnější a uživatelsky přívětivější. Druhá varianta zavádí textová pole a pole výběru, ze kterých se skládá zpětná vazba, což znemožňuje nebo velmi ztěžuje přidání nových typů polí. Uživatel pro vytvoření šablony musí nejdříve vytvořit nová pole nebo využít existující. To není optimální přístup, protože serverová aplikace tím nutí klienta, aby využil právě tuto strukturu, což v okrajových případech může být v rozporu s požadavkem F3.2, který uvádí, že ve zpětné vazbě musí být možné definovat vlastní atributy v závislosti na externí službě využívající novou aplikaci. Zodpovědnost za strukturu zpětné vazby by měla nést klientská aplikace, která využívá serverovou část. Jediným omezením ze strany serveru je, že šablona zpětné vazby musí být definována pomocí JSON schéma [53].

JSON je jedním z nejpoužívanějších formátů pro výměnu dat, který je platformě nezávislý a je dobře čitelný jak pro uživatele, tak pro systémy pracující s daty. JSON schéma je soubor pravidel definující strukturu a validaci dat v JSON formátu. Umožňuje definovat povolené hodnoty, datové typy, minimální a maximální hodnoty, vzory a další omezení, čímž popisuje očekávanou strukturu dat. JSON schéma je často používáno pro kontrolu a validaci dat,

zejména při komunikaci mezi aplikacemi nebo při zpracování uživatelského vstupu. [53]

Po zkontrolování všech požadavků byla entita zpětné vazby doplněna o atributy týkající se popisu a typu. Doplnění vychází z požadavku F3.1, který patří do kategorie M. Dále byla entita uživatele obohacena o doplňující informace jako jméno, příjmení a e-mail, který bude použit pro přihlašování místo jména. Také došlo k odstranění vazby mezi entitami tým a šablona, protože šablony jsou dostupné pro všechny týmy, takže by všechny šablony byly navázány na všechny týmy, tudíž by tato relace ve výsledku nepřidávala žádnou novou informaci. Finální podoba modelu je zobrazena na obrázku 5.3.



Obrázek 5.3: Model entit (konečná varianta)

5.2.4 Rozhraní API

Požadavek F4 uvádí, že aplikace musí poskytovat API rozhraní. Pro komunikaci s klientskou nebo s libovolnou jinou aplikací je vhodné implementovat API podle doporučení a standardů REST API [54]. Komunikace bude probíhat pomocí HTTP metod [8], které slouží k manipulaci se zdroji.

K manipulaci se zdroji budou použity následující HTTP metody:

- **GET** – získání existujícího zdroje,
- **POST** – vytvoření nového zdroje,
- **PUT** – aktualizace existujícího zdroje,
- **DELETE** – odstranění existujícího zdroje.

5.2.5 Bezpečnost

Data budou přístupná pouze autentizovaným a autorizovaným uživatelům. Autentizace bude možná pomocí *tokenu*, který bude aplikace vracet při úspěšném přihlášení pomocí e-mailu a hesla. Druhou možnou variantou autentizace bude pomocí *API klíče*, který se bude vztahovat k jednotlivým uživatelům. API klíč bude mít definovanou platnost, do kdy je možné ho využívat.

Po autentizaci následuje autorizace, což je souhlas umožňující přístup ke konkrétním datům. Administrátor bude mít možnost přistupovat k libovolným datům. Běžný uživatel bude mít přístup pouze k datům, která jsou navázána na tým, do kterého uživatel patří. Bude mít tak přístup i k záznamům zpětné vazby, kterou on sám nezaznamenal. Šablony zpětné vazby budou dostupné pro všechny autentizované uživatele. Správa API klíčů bude dostupná pouze administrátorům.

5.2.6 API endpointy

Serverová aplikace bude poskytovat rozhraní pro komunikaci s klienty pomocí REST API [54]. Komunikace bude probíhat pomocí standardních HTTP metod. Odpovědi budou obsahovat HTTP status kód a v těle se bude nacházet zpráva ve formátu JSON. Požadavky budou směřované na konkrétní endpointy, pomocí kterých bude docházet k manipulaci se zdroji. Ověření uživatelů bude probíhat za využití hlavičky *authorization*, pomocí které bude ověřena identita uživatele. Všechny endpointy vyžadují, aby obsahovala hodnotu uživatelského přihlašovacího *tokenu* nebo uživatelského API klíče. Aktuálním uživatelem je v následujících sekcích označován takový uživatel, který je ověřitelný pomocí zmíněné hlavičky *authorization*. Následuje seznam endpointů pro běžné uživatele, které jsou seskupené podle zdroje, se kterým pracují.

5.2.6.1 Zpětná vazba (feedback)

- **GET API/currentUser/feedbacks** – Vrací seznam záznamů zpětné vazby, ke kterým má aktuální uživatel přístup. Seznam může být vyfiltrován pomocí nepovinného parametru *querry* (filtrování podle názvu nebo typu).
- **POST API/currentUser/feedbacks** – Umožňuje vytvoření nového záznamu zpětné vazby, který je definovaný v těle požadavku. V těle odpovědi se nachází identifikátor *id* nově vytvořeného záznamu.
- **GET API/currentUser/feedbacks/{feedbackId}** – Vrací konkrétní záznam zpětné vazby podle povinného parametru *feedbackId*.
- **PUT API/currentUser/feedbacks/{feedbackId}** – Slouží k aktualizaci existujícího záznamu zpětné vazby podle zadaného povinného parametru *feedbackId*. V těle požadavku se nachází aktualizovaný záznam zpětné vazby.
- **DELETE API/currentUser/feedbacks/{feedbackId}** – Poskytuje trvalé smazání záznamu zpětné vazby podle zadaného povinného parametru *feedbackId*.

5.2.6.2 Šablona (template)

- **GET API/templates** – Vrací seznam šablon zpětné vazby, který může být vyfiltrován pomocí nepovinného parametru *querry* (filtrování podle jména šablony).
- **POST API/templates** – Umožňuje vytvoření nové šablony zpětné vazby, která je definována v těle požadavku. V odpovědi se nachází identifikátor *id* nově vytvořené šablony.
- **GET API/templates/{templateId}** – Vrací konkrétní šablonu podle povinného parametru *templateId*.

5.2.6.3 Tým (team)

- **GET API/currentUser/teams** – Vrací tým, do kterého uživatel patří.

5.2.6.4 Uživatel (user)

- **GET API/currentUser** – Vrací aktuálního uživatele definovaného pomocí hlavičky *authorization*.
- **PUT API/currentUser** – Slouží k aktualizaci aktuálního uživatele. V těle požadavku se nachází aktualizované informace o uživateli.
- **DELETE API/currentUser** – Slouží k trvalému smazání aktuálního uživatele.
- **GET API/currentUser/users** – Vrací seznam uživatelů, který může být vyfiltrován pomocí nepovinného parametru *querry* (filtrování podle jména, příjmení nebo e-mailu).
- **GET API/currentUser/users/{userId}** – Vrací uživatele podle povinného parametru *userId*.

5.2.6.5 Přihlášení

- **POST API/identity/login** – Umožňuje přihlášení uživatele pomocí jména a hesla. V odpovědi se nachází autorizační *token* pro komunikaci se serverem. Nevyžaduje hlavičku *authorization*.
- **POST API/currentUser/logout** – Poskytuje odhlášení aktuálního uživatele.

5.2.7 Správcovské API endpointy

Některé operace se zdroji nejsou přístupné běžnému uživateli. Jedná se o takové operace, které mohou mít dopad na více týmů, například modifikace šablony, která je využívána více než jedním týmem. Tyto potenciálně nebezpečné operace bude moci provádět pouze administrátor, který samozřejmě může využít i endpointy určené pro běžného uživatele. Následuje seznam endpointů, který je dostupný pouze pro uživatele s rolí Admin.

5.2.7.1 Zpětná vazba (feedback)

- **GET API/admin/feedbacks** – Vrací seznam všech záznamů zpětné vazby, který může být vyfiltrován pomocí nepovinného parametru *querry* (filtrování podle názvu nebo typu). Dále může být definovaný parametr *teamId* sloužící k získání záznamů zpětné vazby konkrétního týmu.
- **GET API/admin/feedbacks/{feedbackId}** – Slouží k získání zpětné vazby definované pomocí povinného parametru *feedbackId*.
- **DELETE API/admin/feedbacks/{feedbackId}** – Umožňuje trvalé smazání záznamu zpětné vazby, který je definován podle povinného parametru *feedbackId*.

5.2.7.2 Šablona (template)

- **PUT API/templates/{templateId}** – Slouží k aktualizaci existující šablony podle zadaného povinného parametru *templateId*.
- **DELETE API/templates/{templateId}** – Umožňuje trvalé smazání šablony podle zadaného povinného parametru *templateId*.

5.2.7.3 Tým (team)

- **GET API/admin/teams** – Vrací seznam všech týmů, který může být vyfiltrován pomocí nepovinného parametru *querry* (filtrování podle jména týmu).
- **POST API/admin/teams** – Umožňuje vytvoření nového týmu, který je definovaný v těle požadavku. V těle odpovědi se nachází identifikátor *id* nově vytvořeného týmu.
- **GET API/admin/teams/{teamId}** – Vrací konkrétní tým definovaný pomocí povinného parametru *teamId*.
- **PUT API/admin/teams/{teamId}** – Umožňuje provést aktualizaci existujícího týmu podle zadaného povinného parametru *teamId*. V těle požadavku se nachází nová hodnota týmu.
- **DELETE API/admin/teams/{teamId}** – Slouží k trvalému smazání týmu podle zadaného povinného parametru *teamId*. Smazáním týmu dojde zároveň ke smazání všech uživatelů patřících do týmu a záznamů zpětné vazby zaznamenané členy týmu.

5.2.7.4 Uživatel (user)

- **GET API/admin/users** – Vrací seznam všech uživatelů, který může být vyfiltrován pomocí nepovinného parametru *querry* (filtrování podle jména, příjmení nebo e-mailu). Dále může být definovaný parametr *teamId* sloužící k získání uživatelů konkrétního týmu.
- **POST API/admin/users** – Slouží k vytvoření nového uživatele, který je definovaný v těle požadavku. V odpovědi se nachází identifikátor *id* nově vytvořeného uživatele.

- **GET API/admin/users/{userId}** – Vrací uživatele podle povinného parametru *userId*.
- **PUT API/admin/users/{userId}** – Umožňuje aktualizaci uživatele podle povinného parametru *userId*. V těle požadavku se nachází nová hodnota uživatele.
- **DELETE API/admin/users/{userId}** – Slouží ke smazání uživatele podle povinného parametru *userId*.

5.2.7.5 API klíč (API key)

- **GET API/admin/apiKeys** – Vrací seznam všech API klíčů.
- **POST API/admin/apiKeys** – Slouží k vytvoření nového API klíče. V odpovědi se nachází nově vytvořený API klíč s hodnotou *apiToken*.
- **GET API/admin/apiKeys/{apiKeyId}** – Slouží k získání API klíče pomocí povinného parametru *apiKeyId*.
- **PUT API/admin/apiKeys/{apiKeyId}** – Slouží k aktualizaci API klíče, který je definovaný pomocí povinného parametru *apiKeyId*. V těle požadavku se nachází aktualizovaná hodnota API klíče.
- **DELETE API/admin/apiKeys/{apiKeyId}** – Slouží ke smazání existujícího API klíče, který je definovaný povinným parametrem *apiKeyId*.

5.2.8 Návratové hodnoty

Odpovědi na HTTP metody obsahují HTTP status kód. Jedná se o číselný kód, který slouží k upřesnění výsledku operace. Kódy 2xx jsou označovány jako Success (úspěšné) kódy, 4xx slouží k upřesnění chyby na straně klienta a 5xx indikují chybu na straně serveru. [8]

Následující seznam uvádí takové HTTP status kódy, které může serverová aplikace vracet.

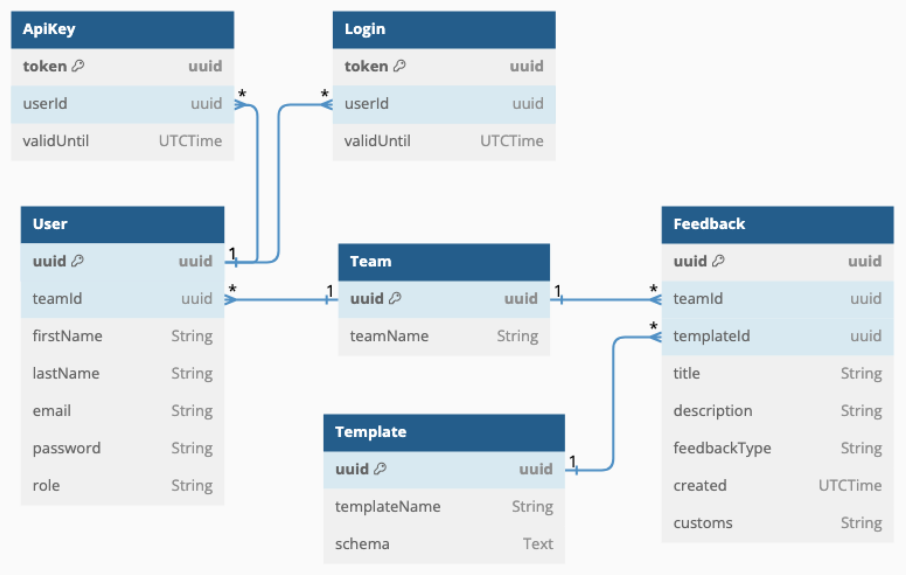
- **200 OK** – Server úspěšně zpracoval požadavek a vrátil odpovídající data. Úspěšné odpovědi pro HTTP metody GET a PUT.
- **201 Created** – Úspěšné vytvoření nového zdroje identifikovatelného dle URI. Úspěšné odpovědi pro HTTP metodu POST.
- **204 No content** – Server úspěšně zpracoval požadavek, ale nevrací žádný obsah. Úspěšné odpovědi pro HTTP metodu DELETE.
- **401 Unauthorized** – Požadavek nemůžu být zpracován, protože uživatel nebyl úspěšně ověřen.
- **403 Forbidden** – Uživatel nemá oprávnění přistupovat k požadovanému zdroji.
- **404 Not found** – Požadovaný zdroj nebyl nalezen.
- **500 Internal server error** – Obecná chybová zpráva. Při zpracovávání požadavku došlo k blíže nespecifikované chybě na straně serveru.

5.3 Databáze

Pro správu dat bude použita relační databáze PostgreSQL (požadavek NF1.2). Navržený model databáze vychází z návrhu modelu entit, který je zobrazen na obrázku 5.3. Návrh databázového modelu je zobrazen na obrázku 5.4.

Pro každou entitu, kterou je potřeba ukládat do databáze, byla vytvořena odpovídající tabulka, která má ve svých sloupcích potřebná data a *uuid* nebo *token*, které reprezentují primární klíč dané tabulky (na obrázku 5.4 zobrazen s ikonkou klíče). Pro každý sloupec byl definován datový typ. Mezi jednotlivými tabulkami existují relace, které odpovídají vztahům mezi daty. V navrženém modelu jsou všechny relace typu jeden ku mnoha, což na příkladu tabulky Uživatel (User) a Tým (Team) vyjadřuje, že tým může mít více uživatelů, ale uživatel musí být přiřazen právě do jednoho týmu. Oproti modelu entit obsahuje databázový model navíc tabulku Login, do které budou ukládány dočasné autorizační tokeny, které jsou vygenerovány po úspěšném přihlášení uživatele za pomocí e-mailu a hesla.

Vytvoření tabulek s navrženými sloupci bude provedeno přímo v implementaci serverové aplikace v jazyce Haskell. Nejsou proto potřeba skripty pro vytvoření tabulek pomocí jazyka SQL, protože Haskell umožňuje tyto operace (včetně migrací) volat přímo ze zdrojového kódu.



Obrázek 5.4: Databázový model

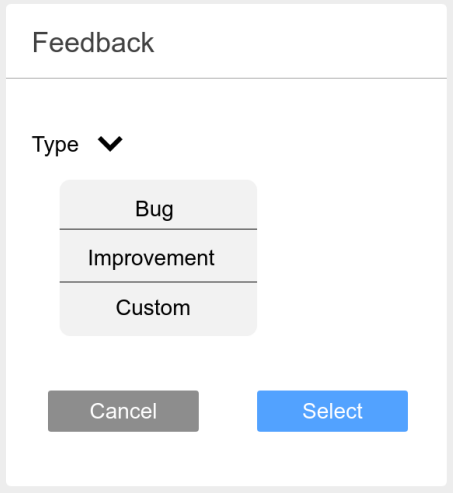
5.4 Klientská aplikace

Klientská část aplikace, která bude využívat serverovou část a bude součástí ekosystému DSW, by měla být podle požadavku NF1.3 implementována v jazyce Elm. Serverovou část aplikace ale nemusí využívat pouze DSW, ale i jiné

služby, které nemusí využívat stejné technologie. Zároveň je žádoucí, aby klientská aplikace byla vizuálně sladěna se službou, kterou je využívána. Komunikace mezi klientskou a serverovou částí bude probíhat pomocí REST API, což je technologicky nezávislý standard. Z toho důvodu bylo rozhodnuto, že realizace klientské části bude zodpovědností každé jednotlivé externí služby, která chce službu pro sběr a správu zpětné vazby využívat. Každá služba se může rozhodnout, jestli implementuje nezávislou klientskou aplikaci, nebo bude novou službu využívat přímo pomocí volání REST API. Pro inspiraci byl vytvořen návrh pro aplikaci DSW, který je integrován přímo do aplikace samotné.

5.4.1 DSW návrh

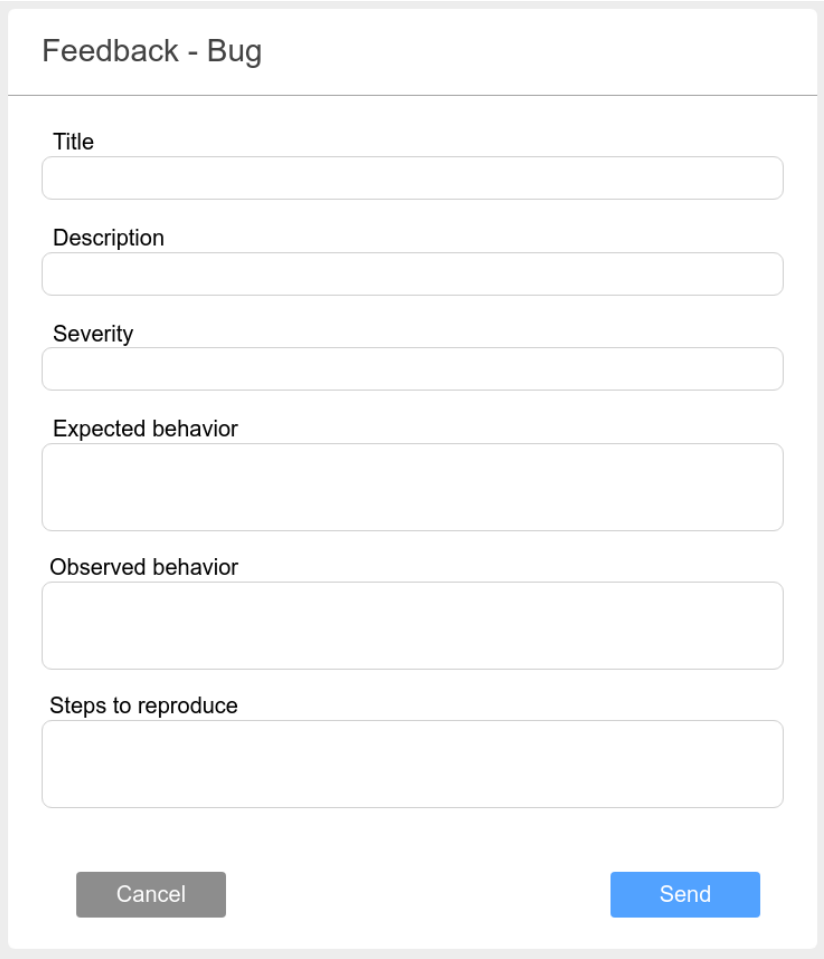
Služba pro správu zpětné vazby by mohla být integrovaná přímo do aplikace DSW. Návrh je lokalizován do anglického jazyka, protože v tomto jazyce je lokalizována aplikace DSW. Namísto aktuálního řešení hlášení obecných chyb pomocí tlačítka *Report issue* (zobrazené na obrázku 1.8) by se mohlo nacházet tlačítko *Feedback*, které implikuje nejen hlášení chyb, ale také možnost reportovat jiné druhy zpětné vazby. Po stisknutí tohoto tlačítka by bylo uživateli zobrazeno dialogové okno, kde bude možné vybrat, jaký typ zpětné vazby chce zadat. Příklad takového dialogového okna je zobrazen na obrázku 5.5.



Obrázek 5.5: Dialogové okno pro vybrání typu zpětné vazby

V něm se nachází tři typy, které je možné zvolit. Jedná se o dostupné šablony zpětné vazby, ke kterým je přidána možnost *Custom*, jež zastupuje zadání zpětné vazby bez využití šablony. Po zvolení jedné z možností a následném stisknutí tlačítka *Select*, které se nachází v pravé dolní části dialogového okna, bude uživateli zobrazeno nové dialogové okno. V něm se budou nacházet konkrétní položky podle zvolené šablony. Příklad dialogového okna po vybrání možnosti *Bug*, která slouží k nahlášení chyby, je zobrazen na obrázku 5.6.

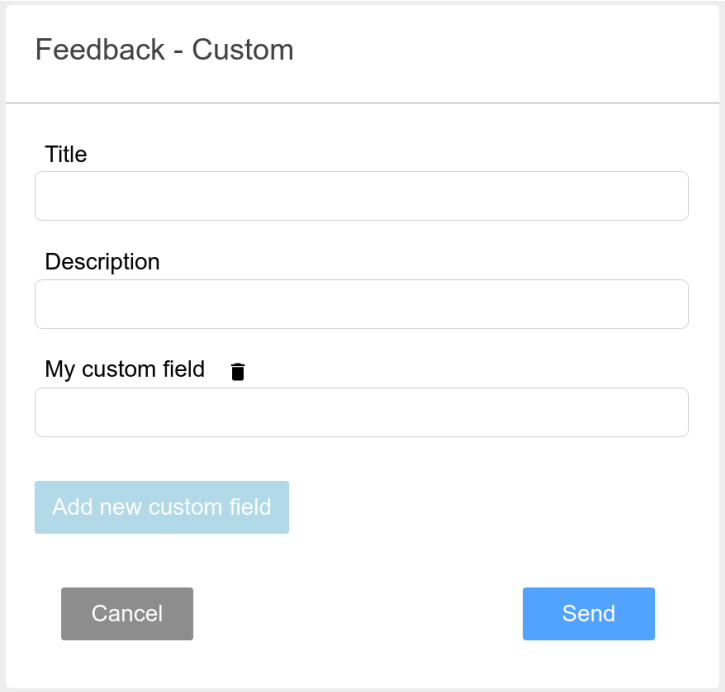
V navrženém formuláři pro nahlášení chyby se nachází obecné informace, které sdílí všechny záznamy zpětné vazby – název (*Title*) a popis (*Description*). Poté se v něm vyskytují pole, která jsou specifická pro hlášení chyb. Jedná



Obrázek 5.6: Dialogové okno pro nahlášení chyby pomocí šablony

se o závažnost (*Severity*), očekávané chování (*Expected behaviour*), pozorované chování (*Observed behaviour*) a kroky k reprodukci (*Steps to reproduce*). Obsah vychází ze studie zmíněné v sekci 3.2, která se zaměřuje právě na hlášení chyb. Po vyplnění a stisknutí tlačítka *Send* bude zpětná vazba odeslána do služby, kde bude zaznamenána. Obdobně by mohly vypadat i formuláře pro jednotlivé části DSW, jako jsou například otázky nebo znalostní modely. Každá komponenta může mít definovanou vlastní šablonu, která by udávala obsah zpětné vazby. Po kliknutí na tlačítko u komponenty, by se automaticky otevřelo dialogové okno pro konkrétní typ. Zároveň by mohlo být automaticky přidáno pole *ExternalId*, které by vyjadřovalo identifikátor dané komponenty, aby bylo možné následně vyfiltrovat zpětnou vazbu týkající se jedné konkrétní komponenty.

Pro přidání zpětné vazby, která se neřídí žádnou šablonou, slouží zmíněná možnost *Custom*. Po jejím zvolení bude uživateli zobrazen formulář zobrazený na obrázku 5.7. I v něm se nachází obecné informace název (*Title*) a popis (*Description*). Pomocí tlačítka *Add new custom field* může uživatel přidat libovolné vlastní pole, u kterého uvede název a následně jeho hodnotu.



Obrázek 5.7: Dialogové okno pro zadání vlastní zpětné vazby

5.5 Testování

Implementované řešení je potřeba důkladně otestovat, aby se odstranilo co nejvíce problémů před tím, než se bude aplikace používat. Podle návrhu bude implementována serverová část aplikace, a proto nejdůležitější částí, kterou je potřeba otestovat, jsou API endpointy. Otestování API endpointů odpovídá integračním testům. Součástí testů budou úspěšné scénáře i záměrně neúspěšné scénáře, aby bylo možné zkontrolovat, zda aplikace vrací odpovídající HTTP status kód, který obsahuje informaci o typu chyby. Bude otestován každý navržený endpoint. Testy by mělo být možné spustit automaticky.

5.6 Shrnutí

Návrh aplikace se musí řídit požadavky, které jsou na nové řešení kladeny. Pokud navržené řešení neumožňuje splnění základních požadavků, nejedná se o správný návrh. Tato sekce obsahuje kontrolu, že navržené řešení pokrývá všechny nezbytné požadavky.

- **F1 Registrace služeb** (splněno) – Registrace externích služeb bude realizována pomocí entity tým. Pro každou externí službu bude vytvořen tým, do kterého budou moci být přidáni jednotliví uživatelé, nebo budou moci zažádat o API klíč a přistupovat k aplikaci pomocí něj. Vytváření týmů a přidělování API klíčů bude mít na starosti Administrátor.

- **F2 Přístup** (splněno) – Navrhované řešení obsahuje běžného uživatele a administrátora. V závislosti na roli je řízen přístup ke zdrojům, kdy běžnému uživateli je dostupné veřejné API rozhraní definované v sekci 5.2.6. Administrátor má povolení přistupovat jak na veřejné API rozhraní, tak na rozhraní správcovské popsané v sekci 5.2.7.
- **F3.1 Základní údaje** (splněno) – Podle navrženého modelu 5.3 bude zpětná vazba obsahovat mimo dalších atributů i atributy reprezentující popis a typ.
- **F3.2 Vlastní atributy** (splněno) – Entita zpětné vazby v navrženém modelu 5.3 obsahuje atribut *customs*, který bude sloužit k zaznamenání vlastních atributů ve formátu JSON.
- **F3.3 Jazyk** (částečně splněno) – Zpětná vazba implicitně neobsahuje informaci o jazyku, ale návrh umožňuje, aby tato informace byla přidána do vlastních atributů.
- **F4 API** (splněno) – Navržené veřejné API rozhraní 5.2.6 umožňuje všem ověřeným uživatelům přidávat, odebírat a filtrovat zpětnou vazbu.
- **F5 Operace se zpětnou vazbou** (částečně splněno) – Aplikace poskytuje rozhraní pro přiřazování a změnu stavu pomocí vytvoření vlastních atributů. Neposkytuje však přívětivé rozhraní pro přidávání reakcí ke zpětné vazbě.
- **F6 Statistiky a reporty** (nesplněno) – Návrh nezahrnuje možnost generování reportů a pokročilých statistik ohledně agregovaných záznamů zpětné vazby.
- **F7 Detekce duplicit** (nesplněno) – Návrh nepočítá s omezením výskytu duplicit.
- **F8 Pravidla pro přiřazování** (nesplněno) – V návrhu nejsou navržena žádná automatická pravidla pro přiřazování a reagování na záznamy zpětné vazby.
- **F9 Integrace s Issue trackery** (nesplněno) – Integrace s Issue trackery není součástí návrhu.
- **NF1.1 Serverová aplikace** (splněno) – Návrh počítá s implementací serverové části pomocí jazyka Haskell za využití webového frameworku Servant.
- **NF1.2 Databáze** (splněno) – Pro trvalé uložení dat bude využit databázový systém PostgreSQL. Navržený databázový model je zobrazen na obrázku 5.4.
- **NF1.3 Klientská aplikace** (částečně splněno) – Klientská aplikace není součástí návrhu, protože bylo rozhodnuto, že by si každá externí služba, která se chce s aplikací integrovat a využívat její služby, měla implementovat klientskou část sama. Hlavním důvodem je, aby byla technologicky i vizuálně sladěna s požadavky externí služby. Pro ekosystém DSW byl vytvořen návrh grafického rozhraní, jak by mohla klientská část vypadat.

- **NF2 Bezpečnost** (splněno) – V návrhu je vyřešen přístup k datům pomocí autorizačního tokenu nebo pomocí API klíče, takže data jsou zabezpečena před neoprávněným přístupem.
- **NF3 Zdokumentované API pomocí OpenAPI** (splněno) – OpenAPI dokumentace bude podle návrhu automaticky vygenerována pomocí nástroje Swagger.
- **NF4 Integrační testy** (splněno) – Serverová část aplikace bude otestována pomocí automatických integračních testů, které budou provedeny pomocí volání API endpointů s konkrétními hodnotami a následné kontroly očekávaných návratových HTTP status kódů.
- **NF5 Nasazení přes Docker** (splněno) – Návrh zohledňuje nasazení do Dockeru pomocí vytvoření a nastavení Dockerfile a docker-compose souborů ve zdrojovém kódu.

Kategorie	Počet	Splněné	Částečně splněné	Nesplněné
Must have	6	6	0	0
Should have	7	5	2	0
Could have	3	0	1	2
Will not have	2	0	0	2

Tabulka 5.1: Souhrnná tabulka splněných požadavků (návrh)

Z tabulky 5.1 je patrné, že návrh počítá se všemi požadavky z kategorie *Must have*. Také dokumentuje, že pozornost není věnována méně prioritnějším požadavkům na úkor požadavkům s vyšší prioritou.

Kapitola 6

Implementace

Jedním z hlavních cílů práce je implementace služby pro sběr a správu zpětné vazby. Implementace vychází z návrhu, který byl vypracován tak, aby zohlednil všechny sesbírané požadavky vycházející ze zadání práce. V této kapitole jsou představeny hlavní technické komponenty, použité programovací jazyky a frameworky, společně s implementačními detaily jednotlivých částí služby. Kapitola slouží k pochopení rozhodnutí, která byla učiněna během vývoje služby. Implementace vychází ze semestrální práce, která byla vytvořena pro účely magisterského předmětu NI–AFP (Aplikované funkcionální programování). Autorem semestrální práce je Bc. František Štěpánek (autor diplomové práce).

6.1 Technické parametry

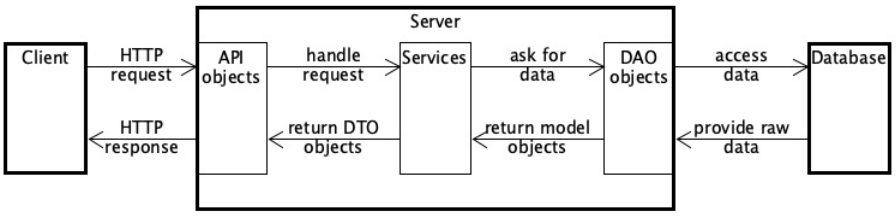
Serverová část aplikace je implementována v jazyce Haskell [6] verze 2010. Zdrojový kód je kompilován za pomoci GHC verze 9.8.2 [55]. Pro vývoj byl použit nástroj Stack [56], který slouží k instalaci potřebných závislostí a kompilátoru GHC v izolovaném prostředí (různé projekty mohou používat různé verze GHC). Dále slouží k sestavení, spuštění a testování projektu. Pro implementaci bylo nezbytné využít i jazyk Python, konkrétně verzi 3.10.4.

6.2 Struktura zdrojového kódu

Projekt, ve kterém se nachází zdrojový kód, je rozdělen do několika hlavních částí. První částí je složka **app**, ve které se nachází soubor `Main.hs`. V tomto souboru se nachází funkce `main`, která v jazyce Haskell slouží jako vstupní bod aplikace. To znamená, že při spuštění bude tato funkce spuštěna jako první. Funkce `main` slouží k načtení konfiguračních proměnných, které jsou následně použity jako vstupní parametry při spuštění serverové aplikace.

Druhá část, která se nachází ve složce **src**, obsahuje veškerou implementaci serverové aplikace. Podle návrhu struktury zdrojového kódu 5.2.2 se ve složce nachází moduly odpovídající navrženým částem. Jedná se o moduly **Api**, **Database**, **Configuration** a **Service**. Detailním popisům jednotlivých částí se věnují následující sekce. Součinnost zmíněných modulů je zobrazena na obrázku 6.1, kde jsou naznačeny i procesy, které mezi jednotlivými částmi probíhají, včetně těch mezi klientem, databází a serverem.

Třetí část se nachází ve složce **test**, ve které jsou soubory pro testování. Podrobnému popisu testování se věnuje sekce 6.7.



Obrázek 6.1: Diagram architektury

6.3 Database

Modul **Database** se zaměřuje na práci s databází. To zahrnuje vytvoření modelu databáze a následný přístup k datům. Model, který vychází z návrhu databázového modelu (sekce 5.3), je definován v souboru `Model.hs`. Pro vytvoření databázového schématu je využit balíček `persistent-template`, který poskytuje nástroje pro definování databázového schématu, které je následně použito pro generování datových typů a migrací. [57] Ve výpisu kódu 1 je zobrazena část definovaného modelu. Obsahuje dvě tabulky `User` a `Team`. Následuje definice sloupců tabulky společně s jejich datovými typy. Pro definování vazeb mezi tabulkami je možné jako datový typ uvést identifikátor tabulky, na kterou je navázána. Příklad definované vazby je možné vidět v tabulce `User` na řádce `teamId TeamId`. Datový typ `TeamId` je automaticky vytvořen při zavedení tabulky `Team`. Pomocí definovaného schématu jsou s využitím zmíněného balíčku prováděny akce jako migrace a vytváření tabulek s definovanými omezeními a závislostmi. Uživatel balíčku `persistent-template` nemusí zadávat ručně SQL dotazy, aby byly zmíněné operace provedeny.

Pro každou tabulku, která je definovaná v databázovém schématu, existuje modul, který má na starosti práci s konkrétní tabulkou. Moduly jsou pojmenované podle jednotlivých tabulek s příponou DAO (database access object), což v překladu znamená objekt pro přístup do databáze. V nich se nachází implementace jednotlivých operací s databázovými objekty, jako je například vytvoření, smazání, aktualizace a filtrování. Využívají k tomu funkce z balíčku `persistent-template`, které jsou kompatibilní s definovaným databázovým schématem a slouží k přístupu bez zadávání SQL dotazů. Všechny funkce využívají společnou funkci `runDB` (výpis kódu 2), pomocí které jsou prováděny databázové dotazy. Funkce přijímá parametr *query*, který slouží k definování databázové operace, jež je potřeba provést. Příklady operací jsou *insert* (vložení), *delete* (smazání), *update* (aktualizace) nebo *selectList* (filtrování).

```
    share
      [mkPersist sqlSettings, mkMigrate "migrateAll"]
      [persistLowerCase|
    User
      uuid      String
      firstName String
      lastName  String
      email     String
      password  String
      teamId    TeamId
      role      UserRole
      deriving Eq Read Show

    Team
      uuid      String
      teamName  String
      deriving Eq Read Show
    |]
```

Výpis kódu 1: Příklad definování databázového modelu v jazyce Haskell

```
runDB :: (Control.Monad.Reader.Class.MonadReader s m,
          Control.Monad.IO.Class.MonadIO m,
          BackendCompatible SqlBackend backend,
          HasPool s (Data.Pool.Pool backend)) =>
          Control.Monad.Trans.Reader.ReaderT backend IO b
          -> m b
runDB query = do
  context <- ask
  liftIO $ runSqlPool query (context ^. pool)
```

Výpis kódu 2: Funkce runDB

6.4 Service

Modul **Service** obsahuje modul pro každou tabulku, která je definována v databázovém schématu. Pro každou tabulku je implementován modul `Service` (například `TeamSerice`), který slouží ke složitějším operacím s daty. Obecně zde dochází k volání funkcí definovaných v `DAO` modulech a následně k validaci návratových hodnot. Návratovými hodnotami jsou datové typy `DTO` (data transfer object), což v překladu znamená objekt pro přenos dat. Tyto typy slouží pro komunikaci mezi serverem a klientem a jsou blíže popsány v sekci 6.5. Pro mapování databázových objektů na objekty pro přenos dat slouží moduly `Mapper` (například `TeamMapper`), které definují funkce pro mapování oběma směry. Příklad mapování pro entitu tým je zobrazen ve výpisu kódu 3.

Mimo strukturu databázových objektů stojí modul `Authorization`, který obsahuje pravidla pro ověření uživatelů. Prvním krokem je autentizace sloužící k ověření, zda přijatý autorizační token odpovídá buď existujícímu platnému přihlašovacímu tokenu, nebo platnému API klíči. Přístup ke zdrojům není

```haskell
fromCreateDTO :: String -> TeamCreateDTO -> Team
fromCreateDTO newUuid createDto =
  Team
    newUuid
    (createDto ^. teamName)

toDTO :: Team -> TeamDTO
toDTO Team {..} =
  TeamDTO
    { _teamDTOUuid = teamUuid,
      _teamDTOTeamName = teamTeamName
    }
```

Výpis kódu 3: Transformační funkce pro tým

v některých případech umožněn všem uživatelům, proto někdy následuje druhý krok – autorizace. Zde je kontrolováno, zda má uživatel dostatečná oprávnění provést požadovanou operaci s daným zdrojem. Kontrola probíhá buď podle příslušnosti k danému týmu, nebo podle uživatelské role. Administrátor může manipulovat se všemi zdroji, na rozdíl od běžného uživatele. Ten má přístup pouze k takovým zdrojům, které patří jeho týmu nebo jsou dostupné všem uživatelům.

Nejdůležitější a nejzajímavější částí je validace zpětné vazby podle šablony. Zpětná vazba obsahuje atribut *customs*, což je JSON obsahující dodatečné informace. Jeho obsah se může řídit atributem *schema* obsaženým v šabloně. Jedná se o JSON schéma, pomocí kterého je možné validovat obsah JSON. Pro validaci byly prozkoumány balíčky jazyka Haskell, které se snaží validaci pomocí JSON schéma implementovat. Bohužel žádný z vyzkoušených balíčků není kompatibilní s GHC verze 9.8.2, který je použit pro kompilaci projektu. Důvodem je, že balíčky nejsou několik let udržované, často proto, že se autorům nepodařilo balíčky implementovat tak, aby dokázaly flexibilně reagovat na vývoj JSON schéma. Balíčky byly vybrány i podle recenze [58], kde jsou shrnuty možnosti pro práci s JSON schéma v jazyce Haskell. Článek je z roku 2022, takže není překvapivé, že ani doporučení z něj nevedlo k úspěšné validaci za použití některého z balíčků. Následuje seznam vyzkoušených balíčků:

- schematic [59] – poslední aktualizace před 4 lety,
- hschema-aeson [60] – poslední aktulizace před 2 lety,
- json-schema [61] – poslední aktualizace před 6 lety.

Z důvodu neexistence vyhovujícího Haskell balíčku pro validaci JSON pomocí JSON schéma bylo nutné provést validaci jiným způsobem. Jedním z nich by byla vlastní implementace. Tato varianta byla zavržena, protože nespadá do rozsahu práce. Implementace validace JSON schéma by mohla vystačit na samostatnou diplomovou práci. Další variantou bylo využít jiný programovací jazyk, jehož kód je možné zavolat jako podproces. Jako kandidát byl vybrán jazyk Python, protože podle [53] pro něj existují oficiální podporované implementace validace pomocí JSON schéma a jedná se o velmi rozšířený jazyk, ve

kterém je možné psát jednoduché skripty. Z nabízených možností, jak v jazyce Python provádět požadovanou validaci, byl vybrán balíček `jsonschema` [62], který poskytuje přímočaré rozhraní pro validaci. Skript pro validaci je zobrazen ve výpisu kódu 4.

```
import sys
import json
from jsonschema import validate

if __name__ == "__main__":
    if len(sys.argv) != 3:
        print("Usage: python validation.py <schema> <instance>")
        sys.exit(1)

    schema = json.loads(sys.argv[1])
    instance = json.loads(sys.argv[2])
    try:
        validate(instance=instance, schema=schema)
    except Exception as error:
        sys.stderr.write(str(error))
        sys.exit(1)
```

Výpis kódu 4: Python skript pro validaci JSON pomocí JSON schéma

Skript přijímá dva parametry ve formě JSON. První parametr je JSON schéma, druhý je JSON, u kterého má být provedena validace. Pomocí metody `json.loads()` jsou načteny vstupní JSON objekty do JSON reprezentace v jazyce Python. Ty jsou následně využity jako parametry v metodě `validate()`. Pokud validace není úspěšná, metoda vyhodí výjimku, která je odchycena a následně zapsána na standardní chybový výstup. Návratový kód skriptu je v případě jakékoliv chyby číslo 1. V opačném případě je navrácen kód s hodnotou 0.

Spuštění skriptu, které se nachází ve validační funkci jazyka Haskell, je zobrazeno ve výpisu kódu 5. Návratová hodnota True (pravda) nebo False (nepravda) odpovídá výsledku validace. Pokud není validace úspěšná, není vytvořen nový záznam zpětné vazby.

6.5 API

Modul **API** obsahuje zdrojový kód, který zajišťuje komunikaci s klientskou částí, jak je patrné z obrázku 6.1. Jsou zde implementovány API endpointy, společně s DTO typy (data transfer object), které slouží pro definování struktury dat, pomocí kterých je komunikováno. Pro každý zdroj, se kterým je manipulováno pomocí API, je implementován datový typ `CreateDTO` (například `TemplateCreateDTO`), který slouží k vytváření a aktualizaci nového zdroje, a `DTO` (například `TemplateDTO`), který reprezentuje zdroj se všemi informacemi a který slouží jako návratový typ při volání API. Jelikož data o uživateli obsahují citlivé informace, konkrétně heslo, není žádoucí z pohledu bezpečnosti, aby se v návratové hodnotě vyskytovala informace o heslu uživatele. Proto je im-

```haskell
validateFeedbackCustoms :: Maybe T.Text
    -> TemplateId
    -> AppContextM Bool
validateFeedbackCustoms mCustoms templateId = do
  case mCustoms of
    Nothing -> return False
    Just customs -> do
      mTemplate <- Template.getByKey templateId
      case mTemplate of
        Nothing -> return False
        Just template -> do
          let schema = templateSchema template
          path <- Utils.getValidationFilePath
          (code, _, _) <- liftIO $
            readCreateProcessWithExitCode
                (proc path
                    [unpack schema, unpack customs])
                ""
          case code of
            ExitSuccess -> return True
            ExitFailure _ -> return False
```

Výpis kódu 5: Validace zpětné vazby

plementován navíc datový typ `UserSafeDTO`, který citlivé informace o uživateli neobsahuje.

Implementace endpointů je rozdělena do jednotlivých modulů podle zdroje, s nimiž manipulují, a podle úrovně oprávnění, která je pro přístup k nim vyžadována. Příkladem jsou moduly `TeamApi` a `TeamAdminApi`. První modul obsahuje implementaci endpointů pro manipulaci s týmy, které vycházejí z návrhu pro běžného uživatele (sekce 5.2.6.3). Druhý implementuje endpointy pro administrátora.

Pro implementaci endpointů v jazyce Haskell byl vybrán balíček `servant`. Jedná se o soubor knihoven v jazyce Haskell pro psaní webových aplikací, které jsou type-safe, ale můžou sloužit i pro odvození klientů nebo generování dokumentace. Toho je dosaženo tím, že jako vstup vezme popis webového API ve formátu Haskell typu. `Servant` dokáže ověřit, že serverová část řešící požadavky správně implementuje webové API. Dále umí generovat popis ve formátu OpenAPI nebo kód pro klientské funkce v jiných programovacích jazycích. [7]

Pro demonstraci byla vybrána implementace manipulace se šablonami (template). Ve výpisu kódu 6 se nachází typ `List_GET`, který slouží jako popis konkrétního endpointu. Hodnoty *API* a *templates* určují podobu URI zdroje, následují definice nepovinného parametru *querry* a hlavičky s hodnotou *authorization*. Poslední hodnotou je definována HTTP metoda s parametry určující formát a obsah návratové hodnoty. Podle popisu se řídí funkce `list_GET`, která má na starost konkrétní implementaci endpointu pro získání všech šablon.

V API modulech se nachází implementace všech endpointů, které spolu souvisí. Každý modul zavádí souhrnný popis svých API endpointů včetně

```haskell
-- GET /API/templates?q={query}
type List_GET = "API"
    :> "templates"
    :> QueryParam "querry" String
    :> Header "authorization" String
    :> Get '[JSON] [TemplateDTO]

list_GET :: Maybe String
    -> Maybe String
    -> AppContextM [TemplateDTO]
list_GET query mToken = do
  authorized <- AUTH.authorized mToken
  case authorized of
    False -> throwError $ err401
    True -> getAllTemplates query
```

Výpis kódu 6: Endpoint /API/templates pro získání všech šablon

kompozice, která určuje, jak mají být požadavky zpracovány. Příklad souhrnného popisu pomocí kompozice typů je zobrazen ve výpisu kódu 6, kde typ `TemplateAPI` definuje popis a funkce `templateServer` definuje zpracování.

```haskell
-- Template API composition
type TemplateAPI = List_GET
    :<|> List_POST
    :<|> Detail_GET

templateAPI :: Proxy TemplateAPI
templateAPI = Proxy

templateServer :: ServerT TemplateAPI AppContextM
templateServer = list_GET
    :<|> list_POST
    :<|> detail_GET
```

Výpis kódu 7: Kompozice typů popisující API endpointy pro práci se šablonami

Ze všech souhrnných popisů a ze všech funkcí, které implementují zpracování, jsou vytvořeny kompozice. Ty pak slouží jako vstupní parametry pro funkce balíčku `servant`, které řeší vytvoření webové aplikace.

6.6 Konfigurace

Serverová aplikace obsahuje konfigurovatelné proměnné. Pro správu konfigurace slouží modul `FeedbackRegisterConfiguration`, který umožňuje nastavit následující hodnoty.

- **Port** – Číslo portu, na kterém má aplikační server běžet.

- **Databáze** – Jméno databáze, která bude využita. Aktuální řešení podporuje databáze PostgreSQL a SQLite.
- **Connection string** (nepovinný) – Textový řetězec obsahující informace o tom, jak se připojit k databázi. Pro databázi SQLite není potřeba.
- **Výchozí e-mail administrátora** (nepovinný) – Výchozí e-mail pro uživatele s rolí Admin, který je vytvořen při prvním spuštění aplikace.
- **Výchozí heslo administrátora** (nepovinný) – Výchozí hodnota hesla pro uživatele s rolí Admin, který je vytvořen při prvním spuštění aplikace.
- **Cesta k validačnímu souboru** – Cesta ke spustitelnému souboru, který provádí validaci JSON pomocí JSON schéma.

Pro nastavení konfigurovatelných hodnot je potřeba soubor `config.yaml`. V něm se nachází dvojice hodnot, kdy první udává jméno proměnné a druhá její hodnotu. Příklad nastavení čísla portu je zobrazen ve výpisu kódu 8. Tato konkrétní konfigurace umožňuje nastavení pomocí proměnných prostředí. Pokud je nastavena proměnná prostředí se jménem `FEEDBACK_REGISTER_PORT`, bude použita právě její hodnota. Pokud ale nastavena nebude, bude využita hodnota definovaná v konfiguraci.

```
feedback-register-port: _env:FEEDBACK_REGISTER_PORT:3000
```

Výpis kódu 8: Příklad konfigurace portu

6.7 Testování

Testování je nezbytným prvkem v procesu vývoje softwarových aplikací, neboť umožňuje ověřit správnost a spolehlivost implementovaného kódu. Testování může být prováděno různými způsoby. Existují testy unit (jednotek), integrační, funkční, regresní, výkonnostní, bezpečnostní nebo akceptační. Každý druh testuje jiné aspekty a uspokojuje různé potřeby, které jsou na aplikaci kladeny.

Pro otestování aplikace byly zvoleny automatizované integrační testy. Integrační testy API endpointů testují nejen správnost návratových hodnot při jejich volání, ale také spolupráci mezi jednotlivými moduly. Pokud by jeden z využívaných modulů nefungoval správně, integrační test nemůže být úspěšný. Právě z toho důvodu byly vybrány integrační testy, protože ověří správnost více modulů najednou včetně jejich spolupráce. Pro implementaci testů byl vybrán balíček `hspec-wai` [63], který umožňuje volat API endpointy přímo ze zdrojového kódu a kontrolovat návratové HTTP status kódy. Pro spuštění testů je využit nástroj `stack`.

Celkově bylo implementováno 106 testovacích scénářů, které mají za úkol otestovat běžné operace, které je možné provést přes API. Každý dostupný endpoint byl otestován, zda vrací odpovídající HTTP status kódy v různých situacích. U každého endpointu je také ověřeno, že je zabezpečený autentizací a pokud požadováno tak i autorizací. Implementované scénáře simulují integrační testy mezi klientem a serverem. Před provedením každého scénáře

jsou do testovací databáze vložena počáteční data, nad kterými jsou následně prováděny operace.

Během testování bylo odhaleno celkem 18 chyb, které se podařilo úspěšně odstranit. Ze zmíněného počtu bylo 14 chyb s velmi nízkou závažností. 5× se jednalo o překlep v URI adrese endpointu a 7× o nesprávný návratový HTTP status kód, kdy například u metody PUT byla místo hodnoty 200 navrácena hodnota 201. Zbylé 4 chyby byly však kritické a bez jejich opravení by nebylo možné službu využívat. Jedna z nich se týkala autorizace, přesněji kontroly přístupu k záznamům zpětné vazby, kdy uživatel chybně neměl přístup k záznamům patřícím k jeho týmu. Dalším kritickým problémem bylo nesprávné navrácení chybového status kódu v případě, kdy zadaná zpětná vazba neobsahovala nepovinný parametr *templateId* reprezentující identifikátor šablony. Nejzávažnějším problémem, který byl odhalený při testování, bylo nefunkční validování obsahu zpětné vazby. Ve zdrojovém kódu byla chyba, která způsobovala, že se validace nikdy nespouštěla, a proto každá zpětná vazba byla označena jako validní. Posledním objeveným kritickým problémem byla záměna dvou funkcí, kdy místo funkce pro aktualizaci zpětné vazby byla použita funkce pro její vytvoření.

6.8 Docker

Pro nasazení aplikace do Dockeru slouží `Dockerfile` a `docker-compose.yml`. `Dockerfile` obsahuje instrukce pro sestavení Docker image. Slouží k popisu konfigurace prostředí, ve kterém bude aplikace běžet, a definuje kroky potřebné k vytvoření tohoto prostředí v Dockerovém kontejneru. Pomocí souboru je možné určit závislosti, které jsou potřebné pro běh aplikace, nastavit pracovní adresář, nakopírovat zdrojový kód do kontejneru, provést instalaci softwaru, nastavit proměnné prostředí a definovat spouštěcí příkazy pro aplikaci.

`docker-compose.yml` slouží k definování a spouštění multi-container aplikací. Tento soubor obsahuje konfiguraci pro jednotlivé služby, které tvoří aplikaci, včetně informací o tom, jaké Dockeru image použít, jaké kontejnery spustit, jaké porty mapovat a jaké další konfigurační volby použít. Dále umožňuje snadno definovat a spustit celou aplikaci s více kontejnery pomocí jediného konfiguračního souboru. Tím se zjednodušuje vývoj, nasazení a správa distribuovaných aplikací, které se skládají z více částí.

6.9 Automatizace

Pro konfiguraci automatizovaných procesů slouží `gitlab-ci.yml`. Jedná se o konfigurační soubor pro Gitlab CI/CD pipeline [64]. Slouží k definování kroků a operací, které mají být provedeny při spouštění pipeline v reakci na určité události, jako je například push do repozitáře.

Soubor obsahuje instrukce, aby při provedení operace push do repozitáře spustil sestavení projektu pomocí nástroje `stack`, sestavení do Dockeru a sestavení validačního skriptu do spustitelného binárního souboru. Při úspěšném dokončení jsou následně spuštěny implementované integrační testy.

6.10 OpenAPI

Aplikace při spuštění automaticky generuje pomocí nástroje Swagger [52] API dokumentaci ve formátu OpenAPI verze 2.0. Ta je dostupná na lokální adrese *localhost:3000/swagger.json* nebo na jiném portu, pokud byl změněn v konfiguraci. Zároveň je automaticky vygenerován webový klient, který usnadňuje používání serverového API. Ještě větší usnadnění nabízí PostMan kolekce, která se nachází v projektu a byla jednorázově vygenerována ze zmíněné dokumentace. Kolekci je možné nahrát do nástroje PostMan [47], který umožňuje pokročilejší operace při volání API endpointů, jako je například definování proměnných, které jsou platné pro celou kolekci.

Kapitola 7

Rozšíření

V této krátké kapitole se nachází návrhy pro budoucí rozšíření stávajícího řešení. Zdrojem jsou buď nerealizované požadavky z kategorií *Could have* a *Will not have* nebo návrhy, které vznikly při implementaci aplikace. Následuje výčet na sobě nezávislých možností, jak stávající řešení obohatit, který je seřazen sestupně podle aktuálnosti případných rozšíření.

- **Klientská aplikace pro DSW** – V návrhu práce se nachází doporučení, jak by mohla vypadat integrace do systému DSW. Navazující prací by na základě zmíněného návrhu mohla být implementace klientské části služby pro správu zpětné vazby pomocí funkcionálního jazyka Elm.
- **Validace pomocí JSON schéma v jazyce Haskell** – Při vývoji bylo zjištěno, že pro jazyk Haskell neexistuje implementace validace pomocí JSON schéma, která by byla podporována nejnovější verzí GHC. Aby byla serverová část čistě v jazyce Haskell, mohl by pro potřeby nejen tohoto projektu vzniknout balíček pro podporu nejnovějších verzí technologií (ať už JSON schéma nebo GHC), který by nabízel práci s JSON schéma včetně validace.
- **Statistiky a reporty** – Aktuální implementace serveru umožňuje jednoduché filtrování vybraných zdrojů. Dalšími operacemi nad agregovanými daty by se mohla zabývat navazující práce, která by poskytovala pokročilé statistiky o zpětné vazbě, ze kterých by bylo možné generovat například měsíční souhrnné zprávy.
- **Obohacení obsahu chybových návratových hodnot** – Server při zaznamenání jakéhokoliv problému během zpracování požadavku navrátí klientovi pouze HTTP status kód, ale nijak blíže nespecifikuje, z jakého důvodu k chybě došlo. Konkrétním místem pro vylepšení by mohlo být navrácení seznamu chyb, které byly nalezeny v průběhu validace pomocí JSON schéma. Klient by mohl díky znalosti validačních chyb upozornit uživatele, aby se pokusil chyby napravit.
- **Automatická pravidlo pro zpětnou vazbu** – Zaznamenaná zpětná vazba může mít díky definované šabloně pevně danou formu a obsah. Za takového předpokladu by mohla být implementována pravidla, která by ji po automatické analýze například přiřadila do definovaných kategorií.

Dále by mohla sloužit ke snížení duplicit nebo alespoň k automatickému označování zpětné vazby jako potenciálně duplicitní.

- **Integrace s Issue trackery** – Další možné rozšíření by se mohlo týkat připravení aplikace na možnou integraci s vybranými Issue trackery. To by umožnilo využít obohacujících funkcí, které Issue trackery nabízejí, a tak by nemusely být implementovány v aplikaci samotné.
- **Implementace unit testů** – Aplikace je otestována pomocí integračních testů, které jsou pro rozsah práce dostačující. Při budoucím rozšiřování zdrojového kódu může nastat situace, že některý z testů selže. Bude proto nezbytné nalézt zdroj chyby. Z integračních testů nemusí být jednoduché zjistit, ve kterém z volaných modulů se chyba nachází. Proto by mohly být implementovány unit testy, které se budou zaměřovat na kontrolu jednotlivých modulů a jejich funkcí.

Závěr

Zadáním práce bylo navrhnout službu pro sběr a správu zpětné vazby, kterou bude možné integrovat do systému DSW. Pro splnění zadání bylo potřeba provést analýzu stávajícího řešení sběru zpětné vazby v systému DSW a pojmenovat klíčové nedostatky. Dále byla provedena rešerše tématu zpětné vazby, která se zaměřuje nejen na technickou stránku věci, ale zohledňuje i uživatelské aspekty problematiky. Byla prozkoumána stávající řešení, která jsou využívána komerčně, nebo se jedná o teoretická doporučení, jak se zpětnou vazbou pracovat. Na základě rešerše, stávajících řešení a shromážděných požadavků, které vyplývají ze zadání práce, byl vytvořen návrh nové služby. V něm bylo rozhodnuto, že externí služby, které chtějí využívat novou službu, musí implementovat své vlastní uživatelské rozhraní, aby bylo vizuálně a technologicky sladěné se službou, do které se integruje. Pro systém DSW byl vytvořen návrh, jak by v něm nové uživatelské rozhraní mohlo vypadat. Důležitou částí návrhu je forma, jakou má zpětná vazba nabývat. Po porovnání navržených variant byla vybrána taková varianta, která umožňuje uživateli definovat vlastní strukturu zpětné vazby za využití komplexního validačního nástroje JSON schéma. V rámci implementace bylo nutné řešit technické a designové výzvy, včetně nedostatečné podpory JSON schéma v programovacím jazyce Haskell. Výsledkem implementace je serverová aplikace pro sběr a správu zpětné vazby, jež komunikuje přes REST API, které je zdokumentované pomocí nástroje Swagger. Aplikace byla úspěšně otestována pomocí automatizovaných integračních testů.

Všechny úkoly vyplývající ze zadání práce byly splněny. Nově navržené řešení umožňuje uživatelům lépe vyjadřovat jejich potřeby a poskytovat kvalitnější zpětnou vazbu nejen na vědecké projekty v rámci DSW. Zároveň oproti stávajícímu řešení centralizuje zdroj zpětné vazby, odstraňuje vysokou závislost na platformě GitHub a umožňuje využití komplexních forem zpětné vazby. V poslední kapitole byly zmíněny možné navazující práce, ze kterých je nejaktuálnější implementace klientské části aplikace pro integraci s novou službou v rámci systému DSW.

Literatura

[1] Data Stewardship Wizard community: Data Stewardship Wizard User Guide. [online], 2023 [cit. 2023-12-10]. Dostupné z: https://guide.ds-wizard.org

[2] Feedier: User Feedback. [online], 2023 [cit. 2024-3-16]. Dostupné z: https://feedier.com/blog/user-feedback/

[3] ReviewTrackers: CSAT vs NPS. [online], 2022 [cit. 2024-3-17]. Dostupné z: https://www.reviewtrackers.com/blog/csat-vs-nps/

[4] eBRÁNA system: 1. díl NPS - Net Promoter Score. [online], 2018 [cit. 2024-3-18]. Dostupné z: https://system.ebrana.cz/nps-spokojenost-klientu

[5] HubSpot: Customer Feedback. [online], 2024 [cit. 2024-3-19]. Dostupné z: https://www.hubspot.com/customer-feedback

[6] The Haskell community: Haskell Programming Language. [online], 2024 [cit. 2024-2-3]. Dostupné z: https://www.haskell.org

[7] Servant Community: Servant - The type-level web DSL for Haskell. [online], 2024 [cit. 2024-2-1]. Dostupné z: https://www.servant.dev

[8] Nielsen, H.; Fielding, R. T.; Berners-Lee, T.: Hypertext Transfer Protocol – HTTP/1.0. RFC 1945, Květen 1996, doi:10.17487/RFC1945. Dostupné z: https://www.rfc-editor.org/info/rfc1945

[9] PostgreSQL Global Development Group: About PostgreSQL. [online], 2024 [cit. 2024-2-14]. Dostupné z: https://www.postgresql.org/about

[10] Amazon Web Services: Amazon S3 User Guide. [online], 2024 [cit. 2024-3-26]. Dostupné z: https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html

[11] Elm: The Elm Programming Language. [online], 2024 [cit. 2024-3-23]. Dostupné z: https://elm-lang.org

[12] GitHub: GitHub Docs. [online], 2023 [cit. 2023-12-10]. Dostupné z: https://docs.github.com/en/get-started/quickstart/hello-world

[13] Git: Git - Fast, Scalable, and Distributed Version Control System. [online], 2024 [cit. 2024-1-20]. Dostupné z: https://git-scm.com

[14] FasterCapital: Customer Feedback and Continuous Improvement. [online], 2024 [cit. 2024-3-16]. Dostupné z: https://fastercapital.com/startup-topic/Customer-Feedback-and-Continuous-Improvement.html

[15] StoriesOnBoard: 8 ways to collect user feedback. [online], 2024 [cit. 2024-4-1]. Dostupné z: https://storiesonboard.com/blog/8-ways-to-collect-user-feedback

[16] PricewaterhouseCoopers: PwC. [online], 2024 [cit. 2024-3-8]. Dostupné z: https://www.pwc.com

[17] Exner, K.: The Complete Guide To Collecting Meaningful User Feedback. [online], 2024 [cit. 2024-3-8]. Dostupné z: https://theproductmanager.com/topics/user-feedback

[18] Kamburov-Niepewna, U.: 7 Reasons Why Customer Feedback is Important to Your Business. [online], 2024 [cit. 2024-3-8]. Dostupné z: https://www.startquestion.com/blog/7-reasons-why-customer-feedback-is-important-to-your-business/

[19] LinkedIn: How can you encourage customers to give feedback? [online], 2024 [cit. 2024-3-8]. Dostupné z: https://www.linkedin.com/advice/0/how-can-you-encourage-customers-give-g0hse

[20] Çökeli, H.: How to Collect, Analyze, and Use User Feedback Effectively. [online], 2022 [cit. 2024-3-13]. Dostupné z: https://medium.com/mytake/how-to-collect-analyze-and-use-user-feedback-effectively-userguiding-22fd78e74b9a

[21] NICE: What is Indirect Feedback. [online], 2024 [cit. 2024-3-13]. Dostupné z: https://www.nice.com/glossary/what-is-indirect-feedback

[22] NICE: What is Direct Feedback. [online], 2024 [cit. 2024-3-13]. Dostupné z: https://www.nice.com/glossary/what-is-direct-feedback

[23] Johnson, T.: Motivating User Feedback Through Survey Links. [online], February 2019 [cit. 2024-3-16]. Dostupné z: https://idratherbewriting.com/2019/02/01/motivating-user-feedback-through-survey-links/

[24] Qualtrics: Net Promoter Score. [online], 2024 [cit. 2024-3-14]. Dostupné z: https://www.qualtrics.com/uk/experience-management/customer/net-promoter-score

[25] Živković, M.: Qualitative vs. Quantitative Feedback – Which is Better for your Product? [online], 2023 [cit. 2024-3-14]. Dostupné z: https://www.feedbear.com/blog/qualitative-vs-quantitative-feedback

[26] CGS Inc.: Using Qualitative Data for L&D ROI. [online], 2024 [cit. 2024-3-14]. Dostupné z: https://www.cgsinc.com/blog/using-qualitative-data-for-l-d-roi

[27] Moran, K.: Usability Testing 101. [online], 2019 [cit. 2024-3-17]. Dostupné z: https://www.nngroup.com/articles/usability-testing-101/

[28] Bika, N.: Conducting Structured Interviews. [online], 2023 [cit. 2024-3-17]. Dostupné z: https://resources.workable.com/tutorial/conduct-structured-interview

[29] Qualtrics: Open-Ended Survey Questions. [online], 2024 [cit. 2024-3-17]. Dostupné z: https://www.qualtrics.com/uk/experience-management/research/open-ended-survey-questions

[30] Usability.gov: Prototyping. [online], 2024 [cit. 2024-3-17]. Dostupné z: https://www.usability.gov/how-to-and-tools/methods/prototyping.html

[31] Tousley, S.: How to Do A/B Testing: 15 Steps for the Perfect Split Test. [online], 2024 [cit. 2024-3-18]. Dostupné z: https://blog.hubspot.com/marketing/how-to-do-a-b-testing

[32] BotPenguin: Feedback Loop. [online], 2024 [cit. 2024-3-19]. Dostupné z: https://botpenguin.com/glossary/feedback-loop

[33] Dawer, N.: The ACAF Customer Feedback Loop. [online], 2024 [cit. 2024-3-19]. Dostupné z: https://www.zonkafeedback.com/blog/the-acaf-customer-feedback-loop

[34] Lübke, D.; Schneider, K.: Leveraging Feedback on Processes in SOA Projects. In *Software Process Improvement*, editace I. Richardson; P. Runeson; R. Messnarz, Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, ISBN 978-3-540-47696-2, s. 195–206.

[35] Schneider, K.; Schwinn, T.: Maturing experience base concepts at Daimler-Chrysler. *Software Process: Improvement and Practice*, ročník 6, č. 2, 2001: s. 85–96, https://onlinelibrary.wiley.com/doi/pdf/10.1002/spip.140. Dostupné z: https://onlinelibrary.wiley.com/doi/abs/10.1002/spip.140

[36] Zimmermann, T.; Premraj, R.; Bettenburg, N.; aj.: What Makes a Good Bug Report? *IEEE Transactions on Software Engineering*, ročník 36, č. 5, 2010: s. 618–643, doi:10.1109/TSE.2010.63.

[37] Chaparro, O.: Improving Bug Reporting, Duplicate Detection, and Localization. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, 2017, s. 421–424, doi:10.1109/ICSE-C.2017.27.

[38] Huang, H.; Zhang, B.: *Text Indexing and Retrieval*. Boston, MA: Springer US, 2009, ISBN 978-0-387-39940-9, s. 3055–3058, doi:10.1007/978-0-387-39940-9_417. Dostupné z: https://doi.org/10.1007/978-0-387-39940-9_417

[39] Gomes, L. A. F.; da Silva Torres, R.; Côrtes, M. L.: Bug report severity level prediction in open source software: A survey and research opportunities. *Information and Software Technology*, ročník 115, 2019: s.

58–78, ISSN 0950-5849, doi:https://doi.org/10.1016/j.infsof.2019.07.009. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0950584919301648

[40] Chris, B.; Alexander, M.: *Text Mining*. Springer Cham, 2014, ISBN 978-3-319-12655-5, doi:10.1007/978-3-319-12655-5. Dostupné z: https://doi.org/10.1007/978-3-319-12655-5

[41] Kelley, K.: What is GitLab and how to use it? [2023 edition]: Simplilearn. [online], 2024 [cit. 2024-4-1]. Dostupné z: https://www.simplilearn.com/tutorials/git-tutorial/what-is-gitlab

[42] Gitlab: Gitlab Issues. [online], 2024 [cit. 2024-4-10]. Dostupné z: https://docs.gitlab.com/ee/user/project/issues/

[43] Brush, K.: MoSCoW method. [online], 2023 [cit. 2024-4-1]. Dostupné z: https://www.techtarget.com/searchsoftwarequality/definition/MoSCoW-method

[44] Fulton, R.; Vandermolen, R.: *Chapter 4: Requirements - Writing Requirements*. CRC Press, 2017, ISBN 9781351831420, 89-93 s. Dostupné z: https://books.google.cz/books?id=ZQMvDwAAQBAJ

[45] AltexSoft Team: Non-Functional Requirements - AltexSoft Blog. [online], 2023 [cit. 2024-3-31]. Dostupné z: https://www.altexsoft.com/blog/non-functional-requirements/

[46] Swagger: OpenAPI Specification. [online], 2021 [cit. 2024-4-1]. Dostupné z: https://swagger.io/specification/

[47] Postman: Postman Collections. [online], 2024 [cit. 2024-4-1]. Dostupné z: https://www.postman.com/collection/

[48] Docker: Docker - Build, Ship, and Run Any App, Anywhere. [online], 2024 [cit. 2024-3-31]. Dostupné z: https://www.docker.com

[49] Hashemi-Pour, C.; Bigelow, S. J.; Courtemanche, M.: Docker Definition - TechTarget. [online], 2023 [cit. 2024-3-31]. Dostupné z: https://www.techtarget.com/searchitoperations/definition/Docker

[50] Čermák, M.: Vícevrstvá architektura: popis vrstev. [online], 2010 [cit. 2024-4-10]. Dostupné z: https://www.cleverandsmart.cz/vicevrstva-architektura-popis-vrstev/

[51] Quillin, B.: The Benefits of a Three-Layered Application Architecture. [online], 2023 [cit. 2024-4-10]. Dostupné z: https://vfunction.com/blog/the-benefits-of-a-three-layered-application-architecture/

[52] Swagger: Swagger Specification v2. [online], 2014 [cit. 2024-2-22]. Dostupné z: https://swagger.io/specification/v2/

[53] JSON Schema: JSON Schema. [online], 2024 [cit. 2024-2-22]. Dostupné z: https://json-schema.org

[54] Richardson, L.; Amundsen, M.; Ruby, S.: *RESTful web apis*. O'Reilly Media, Inc., 2013, ISBN 9781449359737.

[55] The Haskell community: The Glasgow Haskell Compiler. [online], 2024 [cit. 2024-4-29]. Dostupné z: https://www.haskell.org/ghc

[56] Stack contributors: The Haskell Tool Stack. [online], 2024 [cit. 2024-4-29]. Dostupné z: https://docs.haskellstack.org

[57] Haskell Community: Hackage: persistent-template. [online], 2024 [cit. 2024-4-25]. Dostupné z: https://hackage.haskell.org/package/persistent-template-2.9.1.0

[58] Shine, S.: A review of JSON Schema libraries for Haskell. [online], 2022 [cit. 2024-4-25]. Dostupné z: https://dev.to/sshine/a-review-of-json-schema-libraries-for-haskell-321

[59] Haskell Community: Hackage: schematic. [online], 2024 [cit. 2024-4-25]. Dostupné z: https://hackage.haskell.org/package/schematic

[60] Haskell Community: Hackage: hschema-aeson. [online], 2024 [cit. 2024-4-25]. Dostupné z: https://hackage.haskell.org/package/hschema-aeson

[61] Haskell Community: Hackage: json-schema. [online], 2024 [cit. 2024-4-25]. Dostupné z: https://hackage.haskell.org/package/json-schema-0.7.4.0

[62] Berman, J.: jsonschema 4.21.1 documentation. [online], 2024 [cit. 2024-4-26]. Dostupné z: https://python-jsonschema.readthedocs.io/

[63] Hengel, S.; Daisuke, F.: Experimental Hspec support for testing WAI applications. [online], 2020 [cit. 2024-4-20]. Dostupné z: https://hackage.haskell.org/package/hspec-wai-0.11.1

[64] Gitlab: Get started with GitLab CI/CD. [online], 2024 [cit. 2024-4-10]. Dostupné z: https://docs.gitlab.com/ee/ci/

Příloha A

Seznam použitých zkratek

API Application programming interface

CI/CD Continuous Integration and Continuous Delivery/ Deployment

CSAT Customer satisfaction score

DAO Data access object

DevOps Development and Operations

DMP Data management plan

DSW Data Stewardship Wizard

DTO Data transfer object

EB Expected behaviour

GHC Glasgow Haskell compiler

HTTP Hypertext transfer protocol

JSON JavaScript object notation

NPS Net promoter score

OB Observed behaviour

OS Operační systém

REST Representational state transfer

S2R Steps to reproduce

SQL Structured query language

UUID Universally unique identifier

Příloha **B**

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `src` ..... zdrojové soubory
  - `implementation.zip` ..... archiv zdrojových kódů implementace
  - `readme_implementation.txt` ..... popis použití zdrojových kódů
  - `thesis.zip` ..... archiv práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF