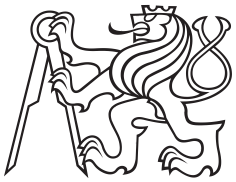
Diplomová práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

Integrační platforma FEL Hub

Bc. Adam Kovář

Vedoucí: Ing. Pavel Náplava, Ph.D.
Obor: Otevřená informatika
Studijní program: Softwarové inženýrství
Květen 2023

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Kovář** Jméno: **Adam** Osobní číslo: **457186**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Softwarové inženýrství**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Integrační platforma FEL Hub

Název diplomové práce anglicky:

Integration platform FEL Hub

Pokyny pro vypracování:

Analyzujte integrační platformu Hub. Zmapujte slabé stránky, navrhněte způsoby vylepšení, vedoucí k zvýšení kvality procesu integrace systémů:
• definujte pojem integrace, krátce porovnejte existující způsoby,
• popište platformu Hub,
• analyzujte a popište problémy, které stávající přístup k integraci současná verze platformy obsahuje,
• navrhněte kritéria hodnocení kvality, vycházející z IEEE Standard for a Software Quality Metrics Methodology a reflektující „developer experience“,
• navržená kritéria aplikujte na stávající verzi platformy a definujte optimální hodnoty, vedoucí ke snížení náročnosti integrace,
• navrhněte úpravy platformy, vedoucí k naplnění definovaných hodnot kritérií,
• navržené úpravy implementujte,
• provedené změny platformy otestujte prostřednictvím integrace testovací aplikace, kterou pro tyto účely vytvoříte, a aplikujte navržená kritéria kvality,
• výsledné hodnoty porovnejte s výchozími hodnotami, výsledky zhodnoťte a definujte standard integračního procesu.

Seznam doporučené literatury:

"ISO/IEC/IEEE Systems and software engineering -- Architecture description," in ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000) , vol., no., pp.1-46, 1 Dec. 2011, doi: 10.1109/IEEESTD.2011.6129467. Dostupné z https://ieeexplore.ieee.org/document/6129467
"IEEE Standard for a Software Quality Metrics Methodology," in IEEE Std 1061-1992 , vol., no., pp.1-96, 12 March 1993, doi: 10.1109/IEEESTD.1993.115124. Dostupné z https://ieeexplore.ieee.org/document/237006
Fagerholm, Fabian & Münch, Jürgen. (2012). "Developer Experience: Concept and Definition." Dostupné z https://www.researchgate.net/publication/258881147_Developer_Experience_Concept_and_Definition
K. Bakshi, "Microservices-based software architecture and approaches," 2017 IEEE Aerospace Conference, 2017, pp. 1-8, doi: 10.1109/AERO.2017.7943959. Dostupné z https://ieeexplore.ieee.org/document/7943959

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Pavel Náplava, Ph.D. Centrum znalostního managementu FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **30.01.2023** Termín odevzdání diplomové práce: **26.05.2023**

Platnost zadání diplomové práce: **22.09.2024**

Ing. Pavel Náplava, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. | Datum převzetí zadání

Podpis studenta

Poděkování

Děkuji Ing. Pavlovi Náplavovi, Ph. D. za vřelý přístup, ochotu, angažovanost a profesionalitu při vedení této práce. Dále děkuji přátelům z Centra znalostního managementu za jejich pomoc jak v technické, tak i osobní rovině.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, 22. května 2023

Abstrakt

Hub je platforma pro integraci systémů Elektrotechnické fakulty ČVUT, která vzniká od roku 2020. Probíhající integrace ukazují, že připojení systému v rámci platformy je poměrně náročný proces. Cílem práce je tento proces zjednodušit.

Prostřednictvím analýzy systému, proběhlých integrací a konzultací se stakeholdery byla definována konkrétní kritéria kvality. Vybraná kritéria zachycují použitelnost autorizačního systému platformy. Pro zlepšení hodnot těchto kriterií byly navrženy změny architektury a formátu autorizačního schématu, které byly úspěšně implementovány. Hodnoty metrik těchto kritérií se podařilo zlepšit oproti jejich výchozímu stavu.

Nové autorizační schéma spolu s backendovou a frontendovou implementací se podařilo propojit s integrovanými systémy a plně tak nahrazuje současné řešení. Nasazení na testovací prostředí nyní brání pouze dosud probíhající vývoj GraphQL federace v rámci platformy.

Klíčová slova: Integrační platforma, Hub, developer experience, DX, GraphQL, microservice

Vedoucí: Ing. Pavel Náplava, Ph.D.

Abstract

The Hub platform is an integration solution for systems of the Faculty of Electrical Engeneering at CTU, which is being developed since 2020. Finished and currently ongoing integrations show that the process of connecting a system via Hub is rather cumbersome. Aim of this thesis is to simplify this process.

Specific quality criteria were defined through an analysis of the system, review of previous integrations and consultations with stakeholders of the project. We decided to focus on criteria depicting usability of the authorization scheme of the platform, as they have been selected as the most impactful. To optimize values of relevant metrics, we proposed a solution comprising of changes to the authorization format, as well as the authorization architecture. The solution have been successfully implemented and values of the given metrics have been impacted positively.

The implemented solution has been successfully tested by connecting it to systems integrated within the platform, which means it is ready to be deployed (once the platform's GraphQL federation gateway is fully functional), replacing the current version.

Keywords: Integration platform, Hub, developer experience, DX, GraphQL, microservice

Title translation: Integration platform FEL Hub

Obsah

Obrázky

Tabulky

Kapitola 1

Úvod

Práce se věnuje platformě Hub, která slouží k integraci systémů Elektrotechnické fakulty. Platforma vzniká od roku 2020 a v současné době probíhá integrace několika fakultních systémů. Probíhající integrace však ukazují, že proces integrace systému je poměrně náročný - dosavadní vývoj platformy probíhal především s ohledem na funkcionality pro koncového uživatele (uživatele integrovaných systémů) a ne pro uživatele platformy (uživatele integrujícího svůj systém). Cílem této práce je proto analyzovat aktuální stav platformy, identifikovat její slabé stránky a navrhnout způsoby vylepšení, vedoucí k zvýšení kvality procesu integrace systémů.

Pro uvedení čtenáře do problematiky věnujeme úvodní kapitolu obecnému přehledu motivací a charakteristik integračních procesů. Zbytek práce je koncepčně rozdělen do dvou částí. Část první obsahuje analýzu platformy, která se skládá z popisu celého projektu a systémové analýzy platformy. Na tyto kapitoly navazujeme rozborem proběhlé integrace systému pro hodnocení zaměstnanců, ve kterém se zaměřujeme na nedostatky platformy, které negativně ovlivnily vývoj systému a jeho integraci.

Před samotným návrhem řešení identifikovaných problémů definujeme metriky kvality prostřednictvím metodologie standardu IEEE [3], abychom pomocí porovnání jejich výchozích hodnot a hodnot po aplikaci řešení ověřili výsledky implementace. Závěrem první části popisujeme návrh vylepšení platformy s ohledem na vliv na hodnoty definovaných metrik.

Druhou část věnujeme popisu implementace navržených řešení, jejich testování a validaci výsledků. Validace výsledků se skládá z hodnocení projektovým manažerem platformy a porovnání hodnot metrik oproti hodnotám výchozího stavu spolu s teoretickými hodnotami řešení. Závěrem hodnocení uvádíme úvahu nad dopadem řešení na integrační proces spolu s navazujícími aktivitami.

Kapitola 2

Pojem integrace

Pro uvedení do obecné problematiky integrací IT systémů věnujeme následující kapitolu popisu existujících typů integrací a jejich charakterisktik.

> „Using different IT components for different tasks is a common practice. But as business functions expand, companies may become overwhelmed by lots of disjointed tools that can't share data and work together. That's when system integration comes to the rescue.“

Citace z článku System Integration: Types, Approaches, and Implementation Steps [4] vhodně uvádí obecné motivace k integrování systémů (konkrétními motivacemi projektu Hub se zabýváme v následující kapitole). Systémovou integraci můžeme definovat [4] jako proces skládání rozdílných softwarových a hardwarových modulů do jedné pospolité infrastruktury, umožňující jednotlivým komponentám fungovat jako celek. Od takového sjednocení pak lze očekávat následující přínosy:

Zvýšená produktivita. Integrované systémy umožňují centralizovat kontrolu nad relevantními procesy, což přidává na efektivitě pracovních postupů. Společnost zvládne větší množství práce během kratší doby díky sjednocenému přístupu k datům a aplikacím, které zaměstnanci potřebují.

Přesnější a věrohodnější data. Datové položky se obnovují naráz napříč všemi komponentami a poskytují tak aktuální vhled pro všechny zúčastněné skupiny.

Rychlejší rozhodování. Z propojených systémů lze snadno získat relevantní data či holistický pohled pro vytváření businessových analýz. Tím, že informace nejsou roztroušeny napříč odstíněnými úložišti, se odstranila nutnost pro manuální extrakci a import do centrálního úložiště, nad kterým by bylo až následně možné provádět analytické operace.

Optimalizace nákladů. Integrace systému spotřebuje zpravidla nižší náklady než výměna všech součástí za nový, a to nehledě na vytváření nové IT infrastruktury.

Integrace mohou však zároveň vytvořit problémy jako např. nová bezpečnostní rizika [5]. V případě integrovaného celku může vniknutí do systému znamenat proniknutí do všech jeho komponent, na rozdíl od stavu kdy od sebe byly odděleny. Vytvořením komunikačních kanálu napříč komponentami navíc vznikají nová slabá místa k napadnutí. Článek [6] dále uvádí, že nižší náklady se rozhodně neprojeví zpočátku integrace. Naopak, úspěšný integrační proces vyžaduje nákladnou investici zdrojů, která se může vyplatit až v rozmezí let. Plně integrovaný systém je navíc díky přidané vrstvě abstrakcí poměrně komplexní. Udržování takového systému v provozu může být velmi náročné a vyžaduje zkušený tým.

Integrace systémů probíhají v mnoha různých prostředích, žádné dvě IT infrastruktury nejsou stejné a každá probíhající integrace je unikátní proces. Existuje množství kategorií integrací rozdělených na základě různých úrovní detailu. Uveďme zde kompilát [4] [6] kategorií z businessového pohledu:

Enterprise Application Integration. Cílem [4] je sjednocení subsystémů v jednom businessovém prostředí. Subsystémy typicky vznikaly nezávisle a postupně nashromáždily velké objemy vzájemně nepřístupných dat. Řešením [6] je propojení databází a workflow subsystémů tak, aby byla zajištěna integrita dat celého systému.

Electronic Data Integration/Interchange. EDI se soustředí na výměnu digitalizovaných dokumentů napříč subjekty (firmy, oddělení společnosti...) ve standardizovaném formátu. Tato integrace nahrazuje papírové dokumenty a automatizuje dané procesy [6]. V případě integrace napříč společnostmi se užívá i termín Business-to-business integration [4].

Data Integration. Cílem DI je sjednotit data a poskytnutí uceleného pohledu. Z tohoto pohledu lze poté čerpat analytická data, přičemž metody pro integraci dat se mohou lišit [6].

Third-party system integration. Integrace systémů třetích stran probíhá především pro rozšíření systému o nové funkcionality, které není výhodné či možné vytvářet proprietárně [4].

Legacy system integration. Velké množství organizací používá zastaralý software, který je zásadní pro jejich byznys. Tento kód nelze kvůli jeho dopadu odstranit či nahradit a místo toho se systémy modernizují propojením s novějšími IT systémy a technologiemi [4].

Z technického pohledu je zásadním lišícím atributem způsob komunikace mezi integrovanými systémy. Mezi nejrozšířenější kanály patří API, jakožto

nejpoužívanější princip propojení [4]. Dále middleware vrstvy, webhooky či tzv. *electronic data exchage*, formát typický pro EDI. Komunikační kanály jsou uspořádány na základě vhodné architektury, která musí odpovídat interním detailům jednotlivých komponent a způsobu jejich komunikace. Mezi obecné metody přístupu k integraci patří [6]:

Point-to-point integration. Poměrně limitovaná forma integrace, která propojuje přímo jeden systém s druhým. Tento způsob se využívá spíše pro specifické, menší integrace. Pro větší systémy mohou být velmi náročné na udržitelnost.

Vertical integration. Tento způsob uplatňuje point-to-point propojení komponent, které jsou úzce spřízněné svou funkcí. Pro konkrétní oblast poté vzniká řetězec propojených služeb. Tyto řetězce vytváří nezávislé struktury, ze kterých vzniká jednoduchý model systému, který ovšem může být velmi neflexibilní [5].

Star integration. Jedná se o další uplatnění point-to-point integrace, ovšem ve větší míře propojení a počtu systémů. Vzájemně komunikující systémy tak schématicky vytvářejí tvar hvězdy. Vysoké množství ledabylých propojení může nicméně vytvářet chaotické uspořádání, velice náročné na údržbu. K tomuto způsobu připojení se proto také váže pejorativní název „spaghetti integration“.

Horizontal integration. Horizontální integrace vzniká uplatněním jednoho subsystému jakožto centrálního komunikačního bodu. Ostatní komponenty jsou propojeny pouze s tímto bodem, což zamezuje komplexním komunikačním schématům jako v předešlém způsobu.

Common data format integration. Datové položky různých systémů a formátů se transformují do jednotného sdíleného formátu. Transformace dat odstraňuje komplexitu rozdílných systémů, které by jinak nemohly společně fungovat.

Jako každý projekt, integrace systémů je unikátní proces. Vyžaduje pečlivé zvážení jak existujících technických detailů, architektury, metodiky, tak i motivací, rizik a přínosů pro danou organizaci.

Projektem Hub a konkrétní podobě integrací do vznikající platformy se zabýváme v následujících kapitolách. Použitím zmíněných charakteristik lze platformu popsat jako aplikaci mixu horizontální a hvězdicové metodiky s použitím převážně REST API jako komunikačního kanálu mezi subsystémy.

Část I

Analýza současného stavu platformy

Kapitola 3

Analýza platformy Hub

V kapitole se věnujeme analytickému popisu platformy. K popisu zde přistupujeme především z pohledu integrujícího uživatele, nezabředáme proto do příliš specifických implementačních detailů komponent platformy a zaměřujeme se na jejich účel/produkt, případně procesy, které ovlivňují jejich vývoj.

3.1 Obecný popis projektu

Pro vysvětlení původu platformy a zasazení do kontextu uvádíme krátký přehled průběhu projektu od jeho vzniku až po současnost.

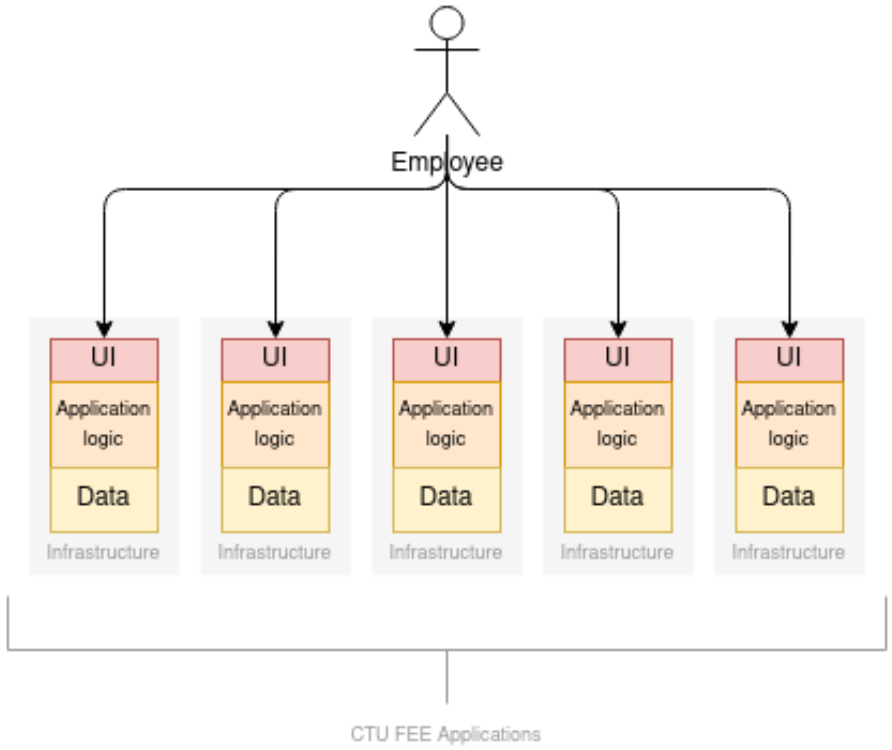
3.1.1 Motivace projektu

Během první poloviny roku 2020 registrovalo pracoviště CZM opakované stížnosti od zaměstnanců fakulty [7] ohledně systémů, které musí používat k náplni své pracovní agendy. Tyto stížnosti se konkrétně týkaly velkého počtu systémů (nikoli jejich funkcionalit). Řada pracovních úkonů totiž vyžaduje interakci s více systémy a střídání mezi nimi zásadě komplikuje práci a prodlužuje jejich trvání.

Tato kritika pochází jak od širšího vedení školy, tak od zaměstnanců, kteří zastávají větší množství pracovních rolí. V reakci na tyto podněty byla vypracována analýza [7], ze které vyplynula následující zhodnocení:

- Fakulta úspěšně elektronizuje své agendy (doktorské studium, semestrální projekty, odevzdávání úloh, studijní dokumenty…).
- Tato řešení vznikaly a nadále vznikají odděleně (obr. 3.1).
- Každá implementovaná elektronizace agendy má své rozdílné uživatelské rozhraní.

- Komunikace zaměstnanců není systematická. Zaměstnanci si vzájemně předávají úkoly zejména prostřednictvím emailu, který neslouží jako vhodná evidence, či zpětná dokumentace k jejich vypracování.
- Chybí centrální, obecný přehled elektronizovaných agend, ke kterému by měli zaměstnanci přístup.



Obrázek 3.1: Ilustrativní schéma uživatelských interakcí s aplikacemi FEL ČVUT

Pro řešení těchto problémů byla navržena integrační platforma, která by měla umožnit sloučení jednotlivých systémů pod jedno uživatelské rozhraní. Přínosy této platformy by měly vzniknout jednak pro koncového uživatele (zaměstnance fakulty) a jednak i pro správce integrovaných systémů. Platforma by totiž měla poskytnout sdílenou infrastrukturu (obr. 3.2), která předejde opětovné implementaci některých částí systémů a vytvořit prostředí, ve kterém by měly jednodušeji vznikat nové aplikace. Pro upřesnění citujme původní vizi [8] z roku 2020 s názvem „Jednotný systém“:

> „Hlavní požadavek je zpříjemnit interakci s různými systémy pro uživatele, kterými jsou především zaměstnanci fakulty. Není požadavek, aby toto zpříjemnění muselo být způsobem integrace toho množství systémů, které máme na fakultě, do jediného systému. Ale

já (Ing. Lukáš Zoubek, pozn. autora) bych volil raději přístup tvorby jednoho centrálního systému, který obsahuje modulovitě funkce a agendy, které se opakují napříč systémy, jako je například vyhledávání, úkoly, notifikace, procesy... Kde tyto funkce jsou řešeny obecně a je možné na ně navázat další systémy pomocí vytvořených a ustálených metodik a rozhraní. Které jsou vyvíjeny za účelem sjednocení fungování různých systémů z pohledu uživatele. A sloužící jako šablony neboli knihovny které musí nově vznikající systémy podporovat, anebo alespoň uvážit jejich podporu.

Také je cílem snížit nutnost tvorby samostatných nových systémů řešících určité nové agendy. A to tím způsobem, že se provede analýza, jestli není možné tyto agendy integrovat pomocí obecných modulů Jednotného Systému. Jako například: „Jestli nová agenda není vlastně určitý proces, který je možné integrovat pomocí procesního modulu. Anebo když agenda vyžaduje určitou novou funkcionalitu, jestli není ekonomičtější o tuto funkcionalitu rozšířit např. ten procesní modul.“

Na základě této představy fungování je možné usoudit, že tento projekt, i když časově náročný, dokáže dlouhodobě snižovat čas a potřebné na adopci nové agendy nebo nové technologie. A to i z pohledu uživatele, který díky jednotnému dizajnu a robustnímu propojení centrálního systému s ostatními systémy (po uživatelské stránce - po technické nemusí být zrovna pevně propojeny) zvykne na používání a přístup k nové agendě výrazně rychleji, jako je tomu doposud. (sic)“

Vize zachycuje snahu autorů o nastavení dlouhodobé koncepce, jejíž realizace by kromě řešení stávajících problémů měla také předcházet jejich vzniku v důsledku elektronizace dalších agend Elektrotechnické fakulty.

3.1.2 Business cile projektu a jejich naplnění

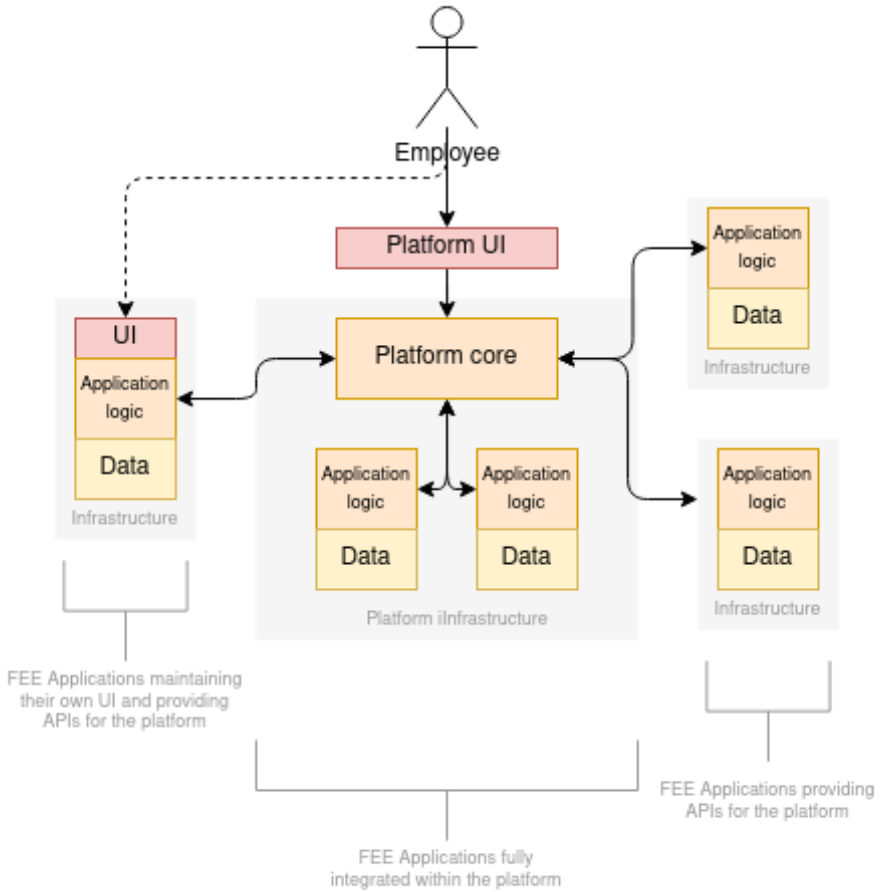
Business cíle projektu byly stanoveny jakožto řešení výše zmíněných problémů a požadavků. V této části vycházíme opět doslova z dostupných projektových dokumentací. V průvodní analýze projektu z roku 2020 [9] byl zadefinován cíl projektu takto:

„Zjednodušit koncovým uživatelům práci s fakultními aplikacemi. Těmito uživateli jsou studenti a zaměstnanci FEL ČVUT. Projekt by měl nejprve položit základ koncepce pro celý Portál FEL (tím je myšleno seskupení všech fakultních aplikací). Dále by se projekt měl podrobněji zabývat zaměstnaneckými aplikacemi, které jsou odkazovány ve školní a jiné legislativě.“

Znění business cílů se podle projektového plánu[10] z roku 2022 s průběhem vývoje projektu postupně upřesnilo na

1. zefektivnění administrativních procesů,
2. centralizaci agend,
3. integrační platformu.

V době psaní této práce se projekt pohybuje ve své prvé fázi nazvané dle projektového plánu[10] „První verze aplikace“. Výstupem této fáze je nasazená stabilní verze aplikace Hub s funkcí pro správu úkolů a notifikační službou, která kombinuje notifikace v aplikaci s rozesíláním e-mailů. Dále je v aplikaci integrovaný systém pro elektronizaci procesů (dále jen „Eprocesy“), systém pro správu závěrečných prací (dále jen „Témata“) a systém pro podporu hodnocení zaměstnanců (dále jen „Evaluace“).



Obrázek 3.2: Ilustrativní schéma uživatelských interakcí s platformou HUB

Hlavními přínosy těchto výstupů je dle projektového plánu [10] přehledné uživatelské rozhraní se sdílenými ovládacími prvky, sjednocené informování uživatele o jeho úkolech a přehlednější administrace těchto úkolů. Interním přínosem této fáze je především hmatatelný výsledek, který pomůže k prosazení dalších fází vývoje a zajistí uživatelskou zpětnou vazbu.

Aktuální stav platformy

V tuto chvíli je již nasazena a zveřejněna produkční verze aplikace, která obsahuje systém pro hodnocení zaměstnanců, napojený na notifikační službu platformy. V reakci na zveřejnění aplikace probíhá oprava chyb a zapracování uživatelských požadavku. Zároveň probíhá integrace Témat, Eprocesů a vývoj modulu pro správu úkolů.

Vývoj platformy probíhá pod záštitou Centra znalostního managementu, podílí se na něm tedy především studenti ČVUT v rámci stážového programu. Personální struktura projektu se dle dokumentace [11] skládá z šesti týmů: architektura, backend, frontend, UX/UI, analýza a testování. Vedle těchto týmů se na projektu podílejí skupiny zodpovědné za vývoj integrovaných systémů. Tyto skupiny zpravidla komunikují se všemi týmy projektu.

3.2 Systémová analýza platformy

Uživatel přistupuje k platformě prostřednictvím webové aplikace ve stylu SPA[12]. Data poskytuje komplexní serverová služba, která propojuje integrované systémy.

Architekturu softwarového systému můžeme definovat jako strukturu komponent programu či systému a jejich propojení, spolu s principy a pravidly, kterými se řídí návrh a evoluce systému v průběhu času [13]. Tímto pohledem architekturu můžeme vnímat i z pohledu procesů, které nutně zavádějí změny jak do implementace, tak jeho návrhu. Architekturu lze pak rozdělit do části makroarchitektury, zkoumající prostředí systému, a části mikroarchitektury, která popisuje jeho vnitřní strukturu.

Mikroarchitektura systému se skládá ze dvou koncepčně i technologicky rozdílných částí: uživatelského rozhraní, tzv. „frontend“ a serverové služby, neboli „backend“.

3.2.1 Backendová architektura platformy

Architektura backendové části platformy se odvíjí od modulárního rozdělení jejích služeb a integrovaných systémů. Je proto založená na návrhovém vzoru „microservice“ architektury [14].

Obecně se jedná o přístup k vývoji aplikace jako ke skupině menších služeb, které fungují odděleně a komunikují mezi sebou prostřednictvím protokolů pro vzdálený přístup, např. HTTP. Služby je v tomto ohledu potřeba vnímat jako víceméně jednoúčelové komponenty, které obstarávají pouze dílčí část systému. Přínosem takového rozdělení je vysoká vnitřní pospolitost služeb a zároveň nízký průnik zodpovědností služeb rozdílných. Prakticky to znamená, že služba A nevytváří řešení mimo účely své zodpovědnosti a využívá řešení přes API služby B. Vnitřní implementační detaily služeb jsou vzájemně naprosto irelevantní, stačí aby dodržovaly sdílený kontrakt (API) a byly dostupné. Takto oddělené služby je možné nasazovat odděleně, což může mít masivní vliv na zrychlení deploymentu systému - při změně uvnitř služby stačí nasadit pouze její novou verzi bez nutnosti nasazení i zbytku celého systému. Služby platformy se koncepčně rozdělují do pěti kategorií:

Služby platformní jsou klíčovou součástí platformy, plní její základní funkce a sdílené funkce napříč integrovanými systémy.

Plně integrované systémy jsou služby poskytující funkce uživatelských agend. Jedná se především o systémy které vznikají nově v rámci platformy.

Částečně integrované systémy plní funkce uživatelských agend, ale na rozdíl od plně integrovaných systémů poskytují některé své služby i mimo platformu.

Externí systémy jsou již existující aplikace, které fungují mimo platformu a poskytují jí data skrz API.

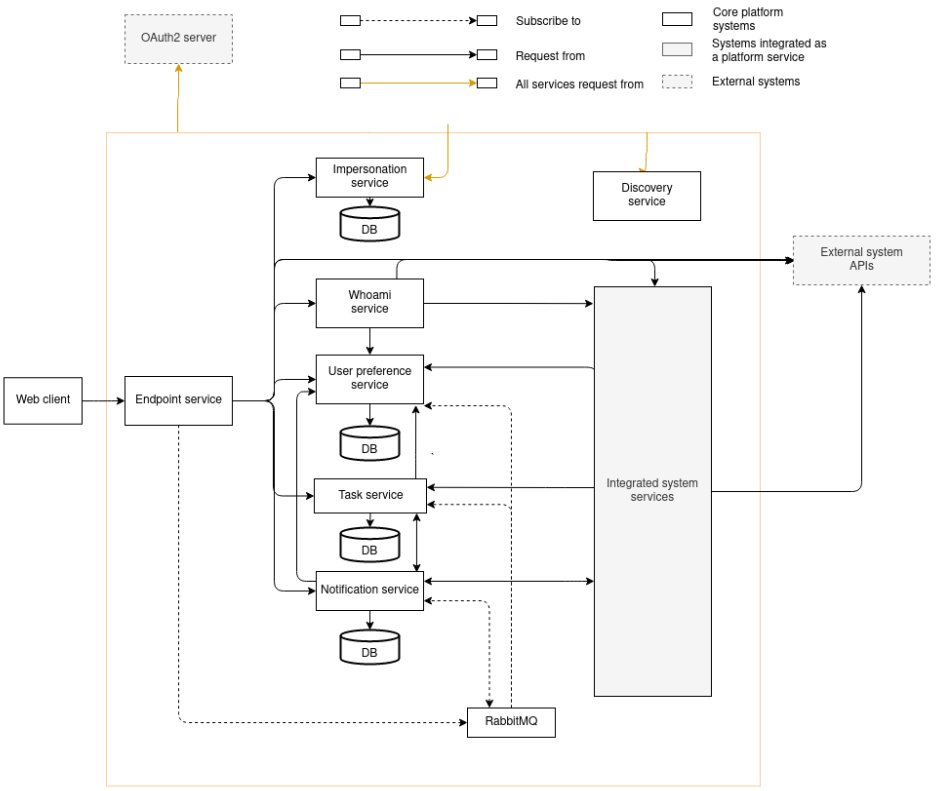
Volně integrované systémy jsou aplikace využívající některé API platformy, např. posílání notifikací.

Komunikace mezi službami

Služby mezi sebou komunikují prostřednictvím HTTP nebo pomocí RabbitMQ[15] implementace asynchronní fronty zpráv. Fronta slouží k propagaci změn napříč službami v návaznosti na nějakou událost. HTTP se používá pro veškerou komunikaci probíhající ve stylu dotaz/odpověď.

Každá služba platformy během spouštění registruje svou adresu v tzv. „discovery“ službě a zároveň od ní získává informace o zbylých službách, které se registrovaly. Discovery služba je implementovaná pomocí Eureka serveru [16].

Pro vzájemnou komunikaci přes HTTP poskytuje každá služba REST API. Tato rozhraní jsou specifikované ve formátu OpenAPI[17]. OpenAPI specifikace umožňuje generování kódu podle dokumentace a to jak na straně



Obrázek 3.3: Schéma služeb a jejich komunikace

poskytovatele API (tzv. „contract-first“ přístup), tak příjemce. Dále je možné vygenerovat specifikaci z již napsaného kódu. Většina služeb v současném stavu generuje specifikaci svého API z kódu - opačný přístup generování kódu ze specifikace se začal používat až v pozdější fázi vývoje.

V tomto ohledu je specifická služba Endpoint service, která poskytuje „bránu“ k celé backendové infrastruktuře. Ta vystavuje jak REST tak GraphQL API, prostřednictvím kterého komunikuje s frontend částí systému a její požadavky předává službám uvnitř infrastruktury.

Implementace služeb

Platformní a plně integrované služby jsou vyvíjeny pomocí frameworku Spring Boot [18]. Každá služba používá sdílenou knihovnu LibCommon, která tvoří její půdorys a poskytuje sdílené funkcionality pro připojení služby v rámci infrastruktury.

Knihovna LibCommon. LibCommon je knihovna sdílená prostřednictvím GitLab Package Registry [19] a integrovatelná do zdrojového kódu služeb

pomocí balíkovacích nástrojů jako např. Maven [20]. Její verze v době psaní práce je *0.1.13_stable*.

Klíčovým produktem knihovny je abstraktní třída *AbstractFelHubApplication*, která poskytuje základní konfigurace:

- registrace k discovery službě,
- nastavení monitorování výpisů služby a trasování požadavků,
- čtení a přesměrování autentizačních tokenů,
- načítání proměnných Docker Swarm prostředí,
- serializace časového formátu,
- konfigurace OAuth2 zabezpečení a falešného zabezpečení pro lokální vývoj služeb.

Pro použití *AbstractFelHubApplication* pak stačí, aby od ní tzv. *main* třída konkrétní služby dědila pomocí klíčového slova *extends*.

Zabezpečení. Všechny požadavky od uživatele se autorizují pomocí protokolu OAuth 2.0 [21]. Komunikace iniciovaná službou se zpravidla řídí http-basic [22] autentizací. Služby používají několik aplikačních profilů, které rozlišují způsob ověření: vývojový, testovací a produkční.

Falešné tokeny. Tyto tokeny jsou aktivní pouze ve vývojových profilech. Slouží především ke snadné impersonaci a ověřování během vývoje služeb. Pro získání falešného tokenu není potřeba žádná autorizační služba. Vývojáři stačí pro HTTP požadavky nastavit autorizační hlavičku ve formátu OAuth 2.0, kde namísto tokenu vyplní řetězec s údaji uživatele, za kterého chce být autorizován.

OAuth2. Mimo vývojové prostředí jsou služby zabezpečené prostřednictvím OAuth 2.0 protokolu. Produkční prostředí využívá ČVUT SSO autorizační server a testovací prostředí se připojuje ke Keycloak [23] serveru, který umožňuje impersonaci. LibCommon konfigurace zajišťuje, že si služba při spouštění vyžádá veřejný klíč autorizačního serveru. Pomocí tohoto klíče pak lze z přijatého tokenu dešifrovat informace o uživateli.

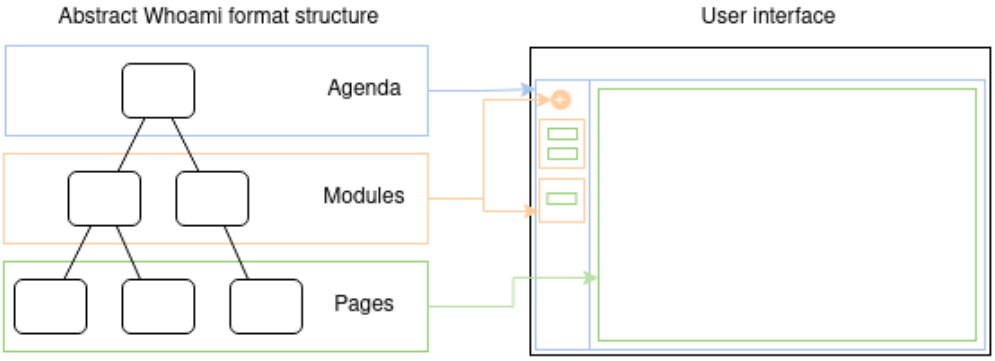
Platformní služby

V popisu služeb se věnujeme jejich rozhraní a způsobu komunikace s ostatními službami.

Endpoint service. Služba Endpoint service je tzv. „API gateway“, tedy brána pro komunikaci mezi službami a webovým klientem. Služba poskytuje GraphQL[24], REST a WebSocket API [25]. REST API poskytuje endpointy pro autentizaci, autorizaci, práci se soubory a úkoly. Skrz GraphQL API klient přistupuje k aplikačním datům služeb. GraphQL spolu s WebSocket API umožňuje real-time komunikaci ve formě GraphQL Subscriptions [26], které slouží např. k posílání upozornění uživateli. Endpoint service se dále věnujeme v analýze frontend architektury.

Whoami service. Whoami služba poskytuje informace o tom, k jakým částem aplikace má uživatel přístup. Jedná se proto o zásadní prvek celé platformy. Služba nemá vlastní databázi a slouží jako zprostředkovatel informací o integrovaných systémech, od kterých získává informace o přístupech uživatele v tzv. „Whoami“ formátu.

Whoami formát. Formát specifikuje, k jakým částem systému má uživatel přístup. Odvíjí se od koncepčního rozdělení systémů a jejich částí do tzv. „agend“, „modulů“ a „stránek“. Agenda je množina modulů, z pohledu uživatele jí lze chápat jako integrovanou aplikaci v rámci platformy. Modul je množina stránek. Stránka představuje hlavní obsah aplikace, typicky se jedná o formulář nebo pohledy na data ve formátu tabulky.



Obrázek 3.4: Ilustrace struktury Whoami formátu a její mapování na UI

Každý z integrovaných systémů musí vystavovat REST endpoint `/whoami`. Dotazujícímu uživateli vrací podmnožinu modulů a stránek, ke kterým má uživatel přístup. Odpověď posílá jako JSON objekt.

Všechny datové typy spadající do Whoami formátu obsahují identifikátor *id*. Objekt odpovědi obsahuje identifikátor systému a atribut *modules* - pole objektů typu *Module*. Typ *Module* obsahuje identifikátor *plusButton* a pole stránek *pages*, tedy objektů typu *Page*. Atribut *plusButton* udává přístup k datovým mutacím na úrovni modulu - v aktuální implementaci se ovšem zatím nepoužívá. *Page* typ obsahuje kromě *id* dále:

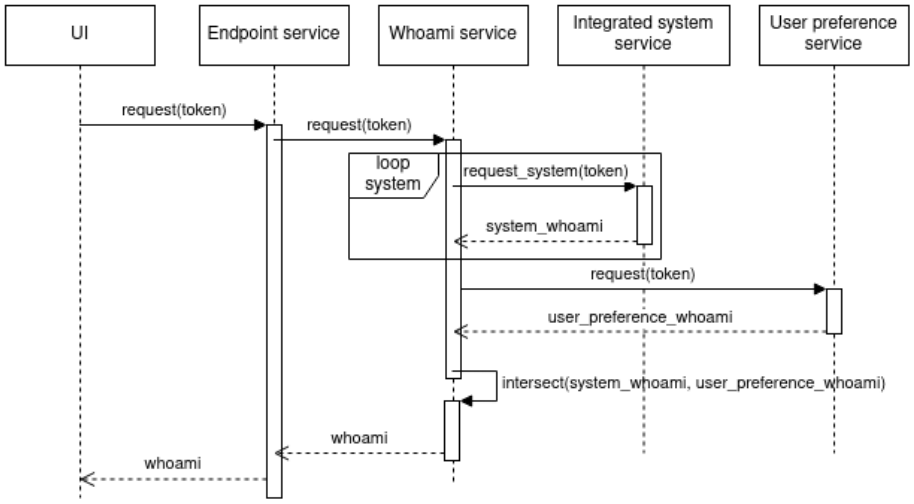
visibleInMenu - pravdivostní položku udávající jestli je stránka přístupná z navigačního seznamu v UI,

content - objekt typu *Content*, který udává typ a přístup k částem stránky,

subPages - pole typu *Page* udávající rozdělení stránky do dalších záložek.

UI v současné době z typu *Page* používá pouze vlastnost *id*.

Whoami požadavek. Služba poskytuje jediný REST endpoint `/user/whoami`. Při přijetí požadavku služba získá informace o uživateli přístupných agendách provoláním `/whoami` endpointu všech integrovaných systémů. Poté se dotáže User preference služby na agendy v jejím úložišti. Průnikem dat od systémů a User preference service vzniká odpověď ve Whoami formátu. Na základě tohoto výsledku se uživateli zobrazí jemu autorizované části UI.



Obrázek 3.5: Sekvenční diagram odbavení Whoami požadavku

User preference service. User preference služba slouží k uchovávání různých uživatelských nastavení. Smyslem služby je umožnit uživateli upravit si chování a rozlišení prvků aplikace. Nastavení se týkají jazyka, notifikací, e-mailů, úkolů a uživatelem definovaných agend.

Notifikace. Datový typ *NotificationPreferences* představuje nastavení upozornění na úkoly uživatele. Součástí jsou vypnutí/povolení upozornění na vytvoření, vyřešení, smazání či úpravu úkolu. Dále je možné definovat upozornění na blížící se datum splnění úkolu.

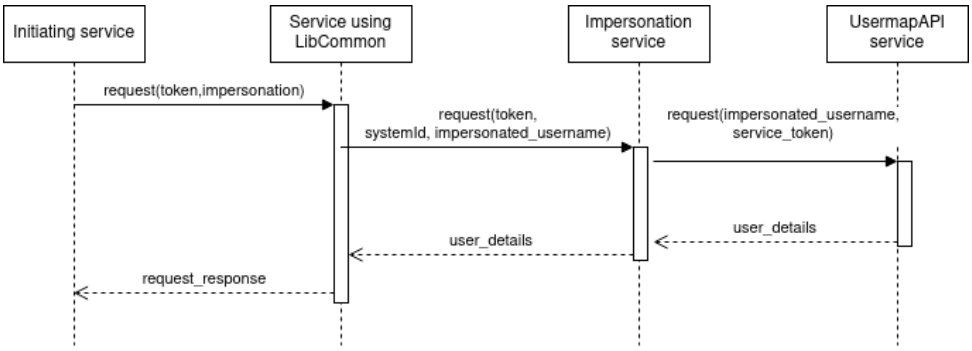
E-maily. Typ *EmailPreferences* obsahuje nastavení odesílání denních a týdenních e-mailů uživateli. Jedná se o vypnutí/zapnutí, denní dobu a dny odeslání.

Agendy. Tato konfigurace slouží k libovolné uživatelské definici agend. Služba umožňuje vytvořit si nové agendy, které mohou obsahovat moduly různých služeb. Agenda dále obsahuje název, popis a typ, který může být buď *CUSTOM* - uživatelem definovaná agenda, nebo *DEFAULT*. Agenda typu *DEFAULT* je předem definovaná a uživatel ji nemůže upravit, pouze nastavit zdali se má schovat z výběru v UI.

Uživatelské rozhraní k nastavování preferencí není v aktuální verzi implementováno, protože doposud nebyly konkrétně definované use-casy a z nich vycházející UI návrhy.

Impersonation service. Služba spravuje přístupy k impersonaci uživatelů, resp. obsahuje informace o tom, jaký uživatel může impersonovat a koho. Tyto práva jsou dále rozdělena podle systémů. Ostatní služby si u ní při přijetí impersonovaného požadavku ověřují, zda má uživatel právo požadavek za uvedenou osobu provést. Služba dále poskytuje „Management“ API pro možnosti upravování oprávnění uživatelů.

Impersonační požadavky. Impersonační požadavky jsou definovány jako HTTP požadavky obsahující v hlavičce klíč *impersonation* s hodnotou uživatelského jména, za které se má požadavek provést. Knihovna LibCommon poskytuje ověřování těchto požadavků prostřednictvím komunikace s Impersonation service.

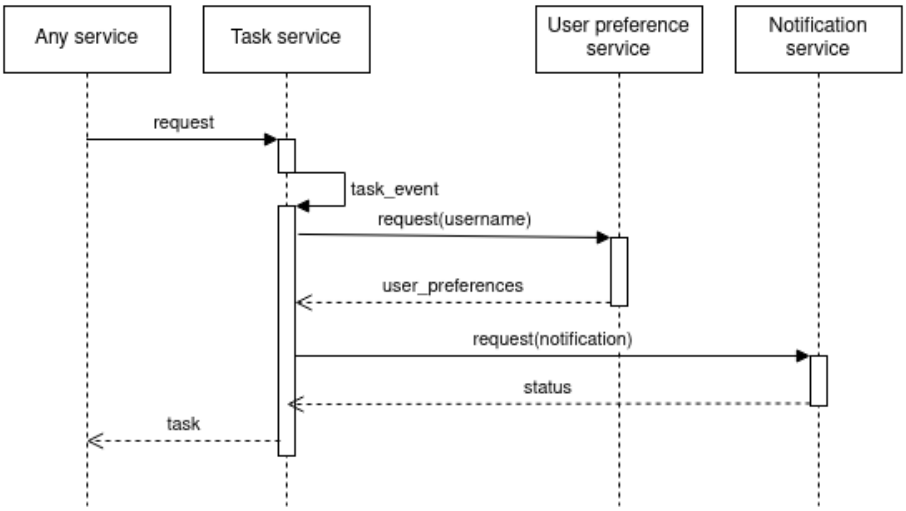


Obrázek 3.6: Sekvenční diagram odbavení požadavku s impersonací

Task service. Služba Task service poskytuje funkce ke správě uživatelských úkolů. Úkol reprezentuje požadavek na uživatelskou akci pocházející z jakéhokoli systému komunikujícího s platformou. UI zobrazuje upozornění na existující úkoly a jejich detail.

API služby poskytuje prostředky k CRUD operacím nad úkoly, které ukládá ve vlastní databázi. Úkoly jsou koncepčně spjaté s notifikacemi a uživatelským nastavením, proto služba intenzivně komunikuje s Notification a User preference service. Při vytvoření, smazání, či úpravě se vytvoří příslušné upo-

zornění. Upozornění se však vytvoří pouze v případě, kdy jeho typ odpovídá povoleným typům notifikací nastaveným uživatelem v User preference service.



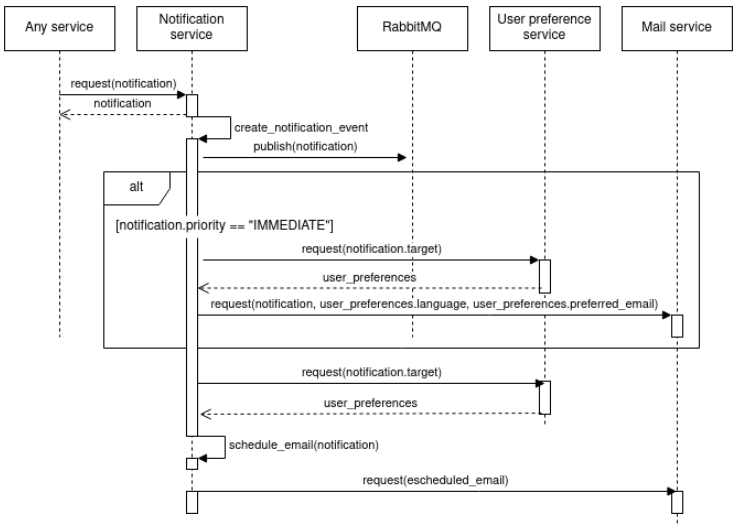
Obrázek 3.7: Sekvenční diagram odbavení požadavku na Task service

Notification service. Služba Notification service slouží k posílání libovolných upozornění uživateli. Umožňuje získávat, vytvářet, mazat notifikace a označovat je za přečtené. Kromě ukládání notifikací slouží služba jako brána k rozesílání upozornění prostřednictvím vícero kanálů - sdílené asynchronní fronty a e-mailů. Každou vytvořenou notifikaci odešle služba do RabbitMQ fronty. Pokud je notifikace typu *IMMEDIATE*, odešle služba požadavek na odeslání e-mailu do Mail service, v opačném případě se odeslání naplánuje podle uživatelského nastavení z User preference service.

Služby integrovaných systémů

Jelikož většina ostatních služeb figuruje v roli konzumenta platformy, nebudeme se jim v této analýze věnovat. Systém Evaluací dále popisujeme v rozboru proběhlých integrací. Na diagramech 3.8 a 3.6 jsou vidět UsermapApi a Email service, kterak odbavují požadavky platformních služeb. Tyto služby vznikly v rámci projektu Eprocesy. S ohledem na to, že na nich závisí logika platformních služeb, se plánuje jejich oddělení od systému a zařazení mezi služby platformní.

UsermapApi service. Služba slouží jako adaptér pro sbírání dat o osobách v ČVUT ze zdrojů jako je LDAP či Usermap. Dále poskytuje informace o uživatelských rolích a organizačních jednotkách v rámci FEL.



Obrázek 3.8: Sekvenční diagram odesílání notifikací přes Notification service

Email service. Služba slouží jako prostředník pro snadné posílání e-mailů přes REST API.

3.2.2 Deployment prostředí

Všechny komponenty platformy se nasazují prostřednictvím GitLab CI/CD [27]. Git repozitáře služeb obsahují specifické větve, ze kterých se zkompilované aplikace dostávají do příslušných prostředí - vývojové, testovací, produkční. Do vývojového prostředí lze nasadit verzi z jakékoli větve repozitáře, kromě větví určených k testovacím a produkčním releasům. Nasazení probíhá v následujících krocích:

1. vystavění docker image [28],
2. spuštění pipeline v projektu „main“,
3. pipeline main projektu pro příslušné prostředí nasadí kontejnery do docker swarmu [29].

Projekt main obsahuje compose [30] script pro jednotlivá prostředí a spolu s jejich GitLab CI konfiguracemi[31].

Monitoring

Pro sběr informací o svém stavu využívá platforma nástroj Prometheus [32], který soustavně monitoruje a ukládá data z běžících služeb jako míra využití

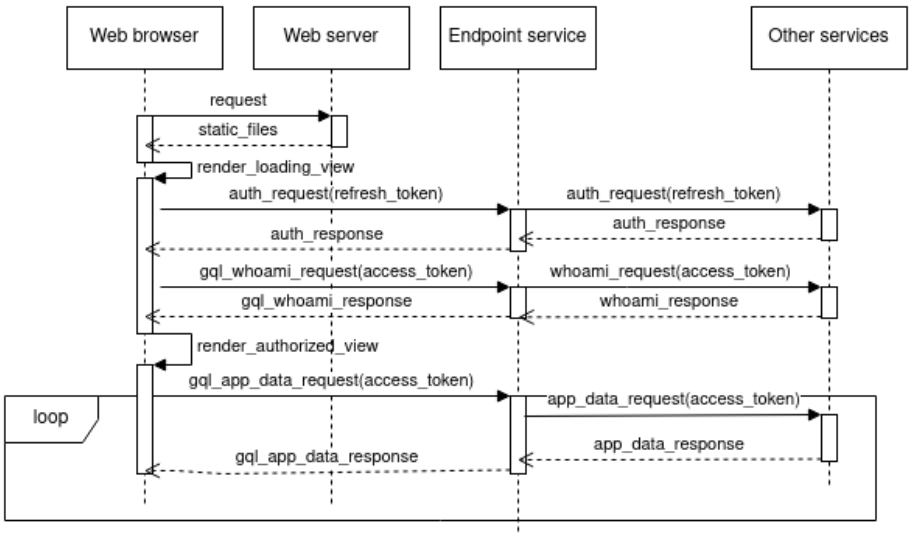
CPU, doba zpracování požadavků apod.

Pro sledování toku požadavků mezi službami je k dispozici tzv. trasování, které je implementováno v knihovně LibCommon pomocí knihovny Sleuth [33].

Veškerá monitorovací data jsou pak vývojářům přístupná skrz nástroje Grafana [34] a Kibana [35].

3.2.3 Frontendová architektura platformy

Frontendová část systému, neboli UI je vyvíjena formou webové SPA. Nad webovými technologiemi HTML, CSS a JavaScript je vyvíjena v jazyce TypeScript s použitím knihovny React [36]. Aplikace je renderována v prohlížeči. S backendovými službami komunikuje výhradně přes Endpoint service, čímž vzniká velmi malé provázání zdrojů backendu a frontendu.



Obrázek 3.9: Sekvenční diagram renderování aplikace v prohlížeči

Komunikace s Endpoint service probíhá především prostřednictvím formátu GraphQL. REST volání se dotazují pouze endpointů `/token` pro autorizaci a `/files` pro nahrávání/stahování souborů.

Analogicky k backendové architektuře je ta frontendová rozdělena do několika na sobě nezávislých aplikací, poskytujících UI pro integrované systémy. Tyto aplikace jsou koncepčně i vizuálně vsazeny do hlavního UI, které jim slouží víceméně jako knihovna poskytující sdílené funkce a komponenty. Na rozdíl od backendových služeb, UI agend sdílí stejný repozitář a jsou nasazovány v hromadném balíku.

Jednotlivé aplikace jsou dynamicky načítány až když je uživatel navštíví. Povolení k přístupu do aplikací je rozhodnuto na základě odpovědi ve Whoami formátu, na kterou se aplikace dotazuje při prvním načtení. Strom UI komponent se porovnává se stromem Whoami dat a uživateli se zobrazí pouze takové komponenty, jejichž identifikátor je ve Whoami datech obsažen. V současné době se toto porovnávání provádí na úrovni aplikací (agend), modulů a stránek. Podrobnější přístupy k datovým položkám se dále řeší na úrovni autorizace endpointů služeb.

3.2.4 Makroarchitektura platformy

Makroarchitekturou systému popisujeme jeho prostředí, které v našem případě chápeme jako vlivy zanášející změny do stavu implementace systému - resp. lidský faktor. Popisujeme zde rozdělení rolí a způsob jejich spolupráce. Popis procesů a komunikace je vyvozen z osobních zkušeností a konzultací (nejsou definovány v projektové dokumentaci). Jak bylo uvedeno v popisu projektu, na vývoji platformy se podílí šest týmů (testování nadále vynecháme, jelikož v době psaní práce tuto pozici nikdo nevykonává), spolu s týmy zodpovídajícími za integrované systémy (dále tým „integrace“). V týmech figurují následující role (v integračním týmu se mohou lišit, uvádíme tedy role které byly dosud napříč projekty společné):

architektura	ar1	správce infrastruktury
	ar2	hlavní vývojář/architekt
	ar3	vývojář/architekt
	ar4	konzultant
backend	b1	vedoucí backend vývojář
	b2	backend vývojář
frontend	f1	vedoucí frontend vývojář
	f2	frontend vývojář
UX/UI	u1	vedoucí designer
	u2	designer
analýza	an1	projektový manažer platformy
	an2	vedoucí analytik
	an3	analytik
integrace	i1	projektový manažer
	i2	analytik
	i3	backend vývojář

Standardní proces přidání funkcionality integrovaného systému začíná vypracováním analýzy analytickým týmem integrace, ze kterého vyplyne zadání pro tým UX/UI a backend. Na základě grafických návrhů a definovaného GraphQL schématu vyvíjí tým frontend uživatelské rozhraní, přičemž je zpětně konzultuje se členy backend týmu (platformní i integrační) a UX/UI týmu. Tento proces podrobněji zachycuje diagram 3.10. Pro upřesnění uvádíme v tabulce 3.1 schéma nezbytné komunikace mezi platformními a integračním týmem (komunikace v praxi probíhá i nad rámec uvedený v tabulce, zachycujeme zde teoreticky nejmenší nutnou podmnožinu). Platformní týmy mají zpravidla týdenní pravidelné schůze, kterých se účastní i projektový manažer platformy. Schůzí integračního projektu se účastní i členové platformních týmů, zejména UX/UI a frontend, kteří jsou k danému projektu přiřazeni.

	ar1	ar2	ar3	ar4	b1	b2	f1	f2	u1	u2	an1	an2	an3
i1	x		x	x		x		x		x	x		x
i2								x		x			
i3			x		x			x					

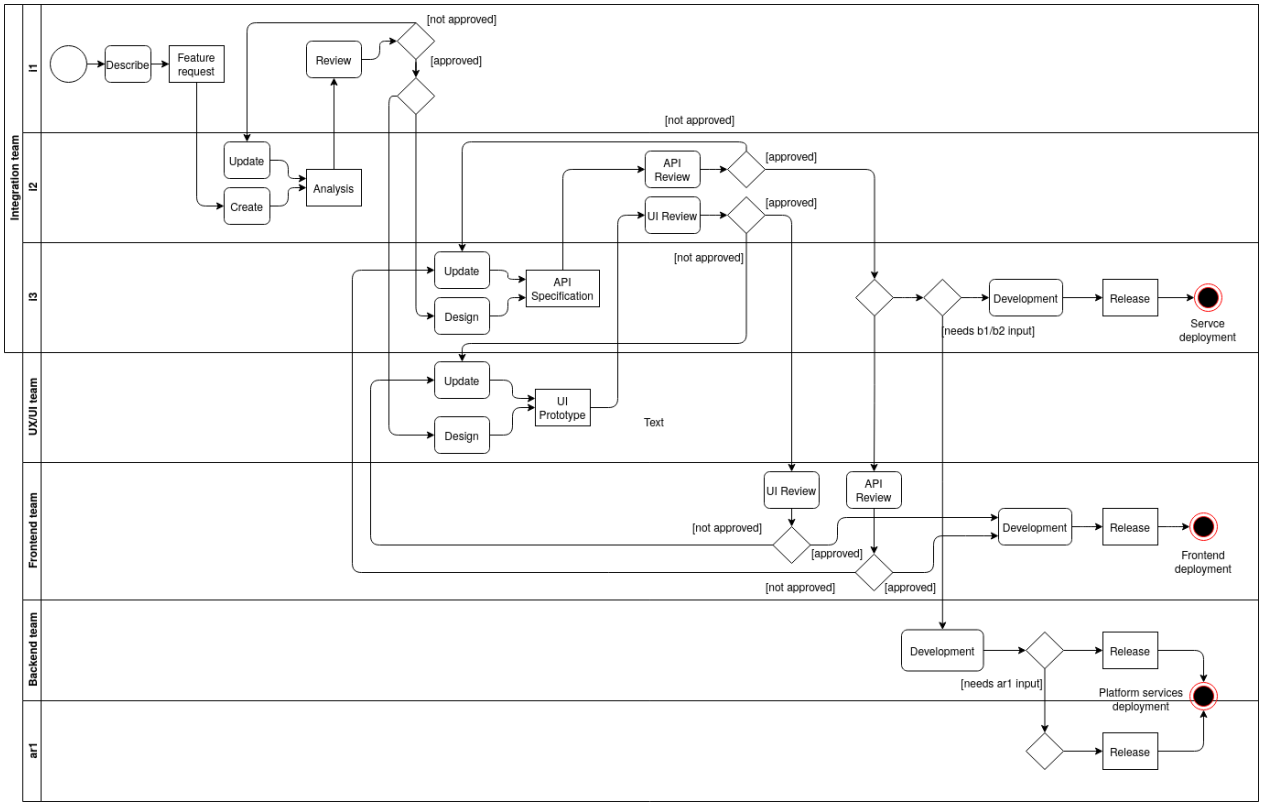
Tabulka 3.1: Matice nutné komunikace napříč týmy platformy a týmem integračním

Členové týmu backend se podílí především poskytováním konzultací ohledně služeb a napojení GraphQL vrstvy s REST API integrovaného systému. Správce infrastruktury se podílí nastavováním CI/CD, monitorovacích konfigurací a přidělováním přístupů. Projektový manažer platformy dohlíží na splnění integračních požadavků platformy, jako např. správná definice Whoami agend apod.

Release management

Vydání nových verzí služeb (spolu s celou jejich Git workflow) se musí řídit společným standardem [37]. Každá zveřejnitelná verze kódu musí být v GitLab repozitáři označená tagem [38], jehož název musí odpovídat tzv. sémantickému verzování [39]. Označený stav repozitáře se nasazuje z „release“ větve, ta je terminální, resp. případné změny zavedené do této větve se již nespojují s hlavní větví. Release větev musí nést název ve formátu *release-<major>-<minor>*, kde *major* a *minor* jsou čísla odpovídající major a minor části příslušné verze vydání.

Nasazení nových vydání služeb vyžaduje jistou míru synchronizace napříč týmy v závislosti na typu změn a jejich provázanosti. Webová aplikace je závislá na stavu GraphqL API v Endpoint service. Pokud GraphQL schéma obsahuje potřebné operace, může se frontendová část nasazovat téměř nezávisle. V případě implementace nové uživatelské funkcionality napříč UI i službami je nutné nasadit nové verze dotčených služeb, Endpoint service (kvůli mapování nových endpointů do GraphQL) i UI zároveň. Synchronizace



Obrázek 3.10: Obecný průběh přidání funkcionality z pohledu rolí.

mezi backendovými službami je nutná pouze v případě změn, které nejsou zpětně kompatibilní s předchozí verzí (zpravidla změna API definice).

Activity	Responsible	Accountable	Consulted	Informed
Platform service release announcement	an1	an1	ar1, ar2, ar4, b1	i1, i3, f1
Integrated service release announcement	i1	i1	i3, ar1, ar2, b1	an1, i1, b1
Platform services release	b2	b1	ar2, ar4, i1, an1	i1, i3
Frontend application release	f2	f1	an2, an1, i2, i1	an1, i1
Integrated services release	i2	i1	i1, b1, ar2, ar1	an1, b1, ar2
Logging, motitoring	ar1	ar1	i2, b1, ar2, ar4, an1	an1, i1, i3
CI/CD configuration	ar1	ar1	i2, b1, ar2, ar4, an1, f1	an1, i1, i3, f1, b1

Tabulka 3.2: RACI matice odpovědností rolí za aktivity spojené s vydáním nových verzí služeb

Pro produkční nasazování je proto zřízen komunikační kanál, kde zodpovědný člen 3.2 týmu musí ohlásit budoucí záměr o nasazení a domluvit se na časovém intervalu, během kterého budou členové dotčených týmů také nasazovat či věnovat zvýšenou pozornost událostem v produkčním prostředí.

Systémovou analýzou poskytujeme vhled do stavu platformy jak z pohledu technické implementace, tak z hlediska lidského faktoru. Platforma Hub je komplexní systém skládající se z více či méně nezávislých komponent, které spravují různé týmy a role. Ty vycházejí z technické podstaty platformy (mikroarchitektury) a pro jejich efektivní součinnost vznikly makroarchitekturní standardy, které cílí na ukotvení produktivních procesů. Platformu jsme popsali bez ohledu na konkrétní integrované systémy, kterým se věnujeme v následující kapitole, kde rozebíráme problémy procesu jejich začlenění do platformy.

Kapitola 4

Analýza integrace systémů a jejích problémů

Abychom zachytili konkrétní slabé stránky platformy, věnujeme následující kapitolu problémům, se kterými se potýkali vývojáři během integrace systémů a které byly zapříčiněny jejími nedostatky. V současné době je v rámci platformy integrovaný systém Evaluací, spolu s ním probíhá integrace Eprocesů a Témat. Integrace Eprocesů a Témat probíhá téměř od počátků vývoje platformy, jsou s ní proto poměrně úzce spjaté a neposkytují validní náhled na to, jak vypadá kompletní integrace z pohledu uživatele. Z tohoto důvodu se věnujeme pouze integraci Evaluací. Obsah této kapitoly vznikl na základě konzultací s tehdejším projektovým manažerem a autorem systému.

4.1 Integrace systému Evaluací

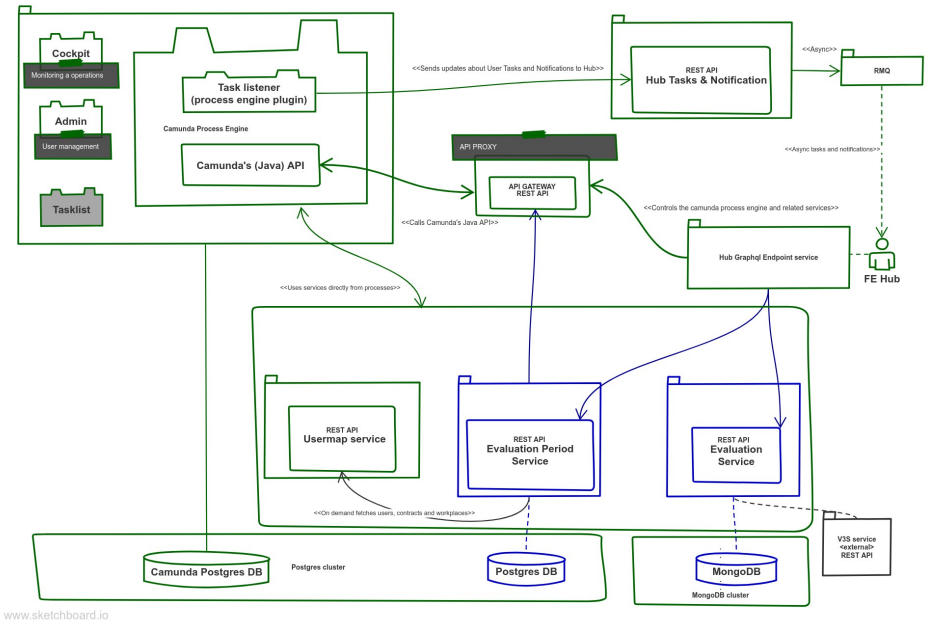
Aplikace pro podporu hodnocení zaměstnanců („Evaluace“) vznikla v roce 2022 v rámci diplomové práce [1] a nasazena byla spolu s prvním nasazením platformy v průběhu srpna 2022. Evaluace jsou prvním integrovaným systémem nasazeným v produkčním prostředí. V této kapitole se zaměříme zejména na problémy vzniklé během integrace aplikace kvůli nedostatkům platformy, opomíjíme zde proto bližší popis projektu.

4.1.1 Průběh integrace

Úvodem projektu Evaluací bylo potřeba se seznámit s technologiemi platformy spolu s integrovanými službami a ověřit, zdali bude jejich využití v rámci projektu schůdné. Zásadní je pro Evaluace nástroj pro správu a automatizaci digitalizovaných procesů Camunda [40], který byl zaveden do platformy probíhající integrací Eprocesů. Dále rozsah a způsob implementace UI - platforma poskytuje dynamické renderování Camunda formulářů, které ovšem

nevyhovovaly všem požadavkům Evaluací a byl proto zvolen přístup vytvoření nových komponent. Následně vznikla hrubá představa agendy Evaluací v rámci platformy.

Na základě výstupů analýzy [1] vznikal návrh aplikace (obr. 4.1), který byl průběžně kontrolován a konzultován s architekturním týmem, správcem infrastruktury a týmem Eprocesů. Hotovým návrhem odstartovala realizace projektu. Jako první produkty vývoje vznikly UI prototypy (obr. 4.2) a specifikace rozhraní služeb v OpenAPI formátu. V této fázi pracoval paralelně backendový a UX/UI tým, přičemž vývoj koordinoval projektový manažer (a zároveň autor) projektu. Design UI vznikal na základě předchozí aplikace pro hodnocení zaměstnanců a UML diagramů (spolu s formátem dat), iterativně se přitom upravoval na základě zpětné vazby dotázaných stakehholderů. Backendový tým vyvíjel služby aplikace Evaluation Period service a Evaluation Form service.

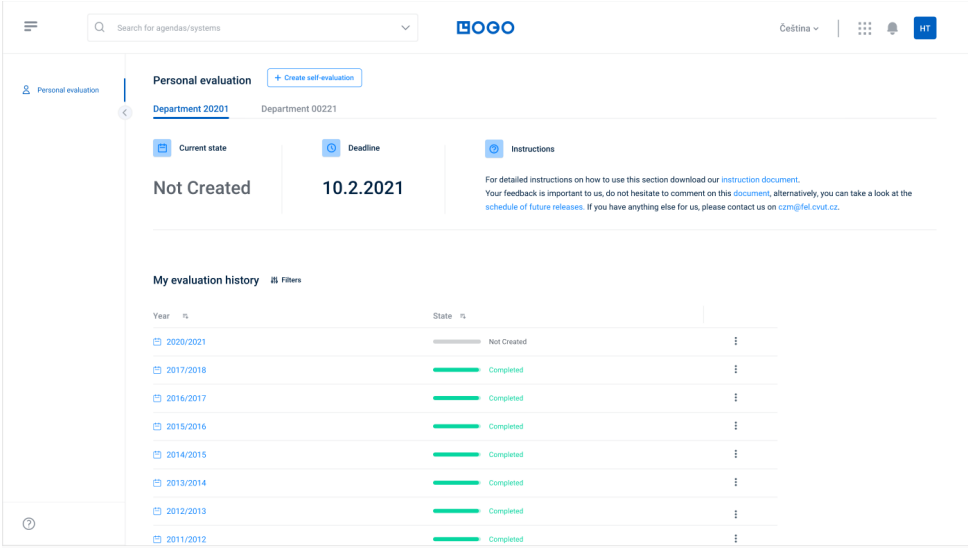


Obrázek 4.1: Nákres architektury aplikace systému Evaluací [1]

Implementace služeb se řídila podle OpenAPI specifikace, následně se definovalo GraphQL schéma a nasadilo na Endpoint service s mockovanými odpověďmi (zatím nepropojeno se službami Evaluací). Mezi implementací a API specifikací neexistoval žádný automatizovaný krok - kód se negeneroval ze specifikace a vice versa také ne. Analytik tedy vydefinoval specifikaci, která se zkopírovala do služeb, které podle ní byly implementovány. Obdobně probíhala integrace do Endpoint service, ve které se vydefinovalo schéma ve sdíleném souboru a následně se implementovaly tzv. resolvery, které mapují GraphQL dotazy na volání REST rozhraní služeb. Toto mapování nejen že přineslo další duplikaci, ale bylo velmi závislé na hlavním vývojáři platformy, který musel vývojáře zaučovat (s dosud GraphQL neměli zkušenosti) a revidovat

změny schématu a Endpoint service.

Podle prototypů a GraphQL schématu začal frontendový tým, paralelně s ostatními, vyvíjet UI agendy Evaluací. S probíhajícími iteracemi a novými požadavky se poté měnily jak UI návrhy, tak jejich frontendová implementace. Během UI implementace se dále měnily i požadavky frontendového týmu na podobu schématu. Po propojení služeb s Endpoint service a odstavením mockovaných odpovědí proběhlo refaktorování UI, protože mockované schéma v tu dobu již nebylo aktuální. Propojení služeb dále zahrnulo Usermap service a služby Camundy, během kterého bylo nutné předělat autorizaci služeb Evaluací. Původní autorizace služeb fungovala na základě jiného uživatelského identifikátoru než ostatní služby (chybělo jednotné schéma autorizace), což pokrývalo velkou oblast API. Každá změna se promítla do několika specifikací a následných implementací (popsáno výše).



Obrázek 4.2: Návrh jedné ze stránek UI Evaluací [1]

Služby Camundy v tuto dobu rovněž procházely vývojem, což mělo za následek jejich časté výpadky. Výpadky ovlivnily nejen frontendový tým, který vyvíjel oproti nasazenému „dev“ prostředí, ale také vývojáře backendu, kteří pro otestování museli na to samé prostředí nasazovat své změny kvůli složitému lokálnímu zprovoznění služeb. Stejně ovlivněno bylo i samotné vyvíjení a testování procesu prostřednictvím Camundy. Pro nedostatečnou zpětnou vazbu systému byla pro získání vhledu to stavu služeb Camundy potřebná součinnost jejich vývojáře, který se dokázal připojit a poskytnout zalogované hlášky programu. Tento problém se v průběhu vývoje vyřešil zprovozněním monitorovacích služeb popsaných v předchozí kapitole.

Výčet problémů integrace

Vývoj systému Evaluací probíhal podle zavedených praktik softwarového inženýrství, před samotnou implementací nejprve proběhla důkladná analýza nástrojů, poté následoval návrh architektury a následně definice artefaktů jako jsou databázová schémata, API specifikace, návrhy UI (spolu s uživatelským testováním prototypů) apod. Zároveň i aktivity potřebné k integrování systému do platformy probíhaly podle definovaných standardů. I přesto byl celý proces negativně ovlivněn nedostatky platformy, které tehdejší projektový manažer shrnul následovně:

Chybí technický garant platformy Neexistuje osoba, která by měla dostatečný přehled o platformě do takové míry, aby mohla efektivně vymáhat standardy a vyznala se ve všech zavedených technologiích. Řada stážistů je poměrně juniorní a s použitými nástroji se setkala poprvé.

Chybí SoC napříč projekty Služby Evaluací využívají funkcionality služeb projektu Eprocesů, které nebyly koncipovány samostatně, např. UsermapAPI service. Na těchto službách navíc probíhal vývoj, což vynucovalo soustavnou komunikaci napříč projekty a slaďování nasazených verzí.

Meziprojektová komunikace Po integrujících týmech nebyly vynucovány sdílené technické standardy, což vedlo k nesrovnalostem v datových položkách, např. uživatelského identifikátoru. Dalším aspektem je chybějící komunikace ohledně nasazování, které nebylo řádně zkoordinováno (v návaznosti vznikla koncepce release managementu popsaná v rámci makroarchitektury).

Vyvíjení na *dev* prostředí Pro integrující vývojáře je obtížné zprovoznit služby lokálně a k testování implementace nasazují své změny na *dev* prostředí.

Chybí impersonace V době produkčního nasazování Evaluací nebyla implementovaná impersonace, takže nebylo možné kontrolovat stav integrované aplikace skrz UI.

Whoami formát není srozumitelný Whoami funkcionalita Evaluací se implementovala až závěrem, přičemž členům integračního týmu nebyl zřejmý její účel a bylo nutné vícero konzultací se členy platformních týmů. Whoami definice se navíc musí definovat na více místech - primárním zdrojem je dokumentace platformy, ta se musí zreprodukovat do User preference service a na úrovni služeb probíhá samotná autorizace.

Monitoring Teprve v průběhu vývoje Evaluací se zavedly monitorovací služby, přičemž nebylo dostatečně předáno know-how ohledně jejich použití. V logovacích záznamech se složitě vyhledává a členům integračních týmů chybí přístupy.

API služeb je chápáno jako výsledný produkt Vývojáři často testovali pouze rozhraní služeb nehledě na výsledné propojení s Endpoint service přes GraphQL. Proto chyby v propojení často odhalili až členové frontend týmu.

Pomalé odstraňování chyb Při zaznamenání chyby v nasazeném prostředí je složité dohledat její původ.

Špatný vhled Ke zjištění stavu běžících služeb je potřeba spolupráce členů jiných týmů či správce infrastruktury. K API nasazených služeb se složitě přistupuje.

Složitý rollback Platforma nemá implementovanou strategii pro uvedení deploymentu do předchozí verze.

Pomalé dotazy přes GraphQL Dosavadní GraphQL implementace se nepotýkala s větším množstvím dotazů které by předávala napříč službami. Pro služby Evaluaci proto nebyl poskytnut žádný způsob optimalizace (např. „batchování“ dotazů) a některé dotazy trvají v řádu desítek sekund (některým dokonce vyprší životnost).

Duplikace GraphQL Neexistuje žádný automatizační krok mezi implementací a dokumentací - schéma je upravováno v jednom sdíleném souboru a následně jsou implementovány resolvery.

Duplikace OpenAPI Dokumentace API služeb se přidává do sdíleného repozitáře dokumentací. Z něj se následně kopíruje do služeb, ve kterých se z podle dokumentace implementuje API či vygeneruje klient.

Malá konzultace frontendovým týmem Frontend tým nedostatečně komunikoval své požadavky na podobu GraphQL schématu, což vedlo k refaktorování kódu služeb i UI.

Chybí podpora content managementu Platforma neobsahuje způsob efektivního předávání informací uživatelům o výjimečných stavech aplikace. Při výpadku infrastruktury bylo potřeba ukázat na stránkách Evaluací zprávu o dočasných problémech a jediný způsob jak komponentu zobrazit bylo nasadit novou verzi frontendu.

Na pokrytí těchto slabých stránek platformy se od doby integrace Evaluací intenzivně pracuje. Ze systémové analýzy platformy (viz druhou kapitolu) si lze povšimnout, že aktuální stav část z nich již neobsahuje.

4.1.2 Aktuální stav řešení problémů integrace

Pro standardizaci propojení služeb byl vydefinován proces release managementu, který zavedl pravidla pro vydávání verzí služeb a tím umožňuje

konzumentům služeb se spoléhat na stabilní podobu jejich rozhraní. S tím dále souvisí i usnadnění rollbackování služeb.

Součástí systému Evaluací je procesní engine Camunda, který je spravován týmem Eprocesů. Aby se umožnilo systémům využívat tento engine bez závislosti na stavu služeb Eprocesů, byl navržen koncept tzv. „embedded“ Camundy, který umožní službám zprovoznit vlastní instanci procesního enginu.

Další sdílené potřeby jako např. UsermapApi service se postupně přesouvají pod správu platformního týmu, aby mohly být provozovány jako stabilní funkcionalita platformy.

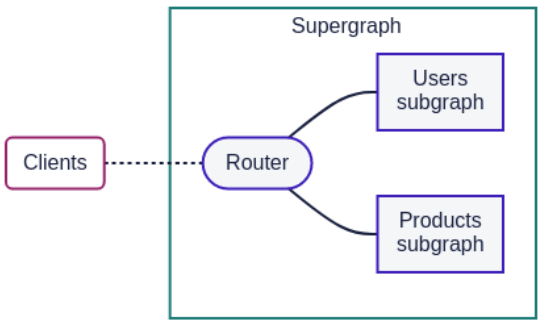
Pro usnadnění lokálního vývoje vznikl ustálený princip „lokálního developmentu“, který definuje postup připojení k službám infrastruktury běžícím v *dev* prostředí. Díky tomu stačí vývojáři integrovaného systému lokálně zprovoznit pouze své služby.

Chybějící funkcionalita impersonace byla implementována již zmíněnou Impersonation službou.

Pro zamezení duplikace OpenAPI specifikací byl vytvořen dokumentační nástroj, který automaticky načítá ze služeb jejich specifikace. Díky tomu jsou specifikace přístupny z jednoho zdroje jak k nahlížení, tak i stahování do projektů.

GraphQL federace

Významný podíl komplikací (duplikace kódu i specifikací, pomalé požadavky...) vycházel z GraphQL vrstvy a příslušné služby Endpoint service. Pro optimalizaci požadavků byla sice přidána podpora pro jejich batchování, nadále ovšem přetrvala duplikace rozhraní a nutnost spravování jednoho schématu uvnitř Endpoint service.



Obrázek 4.3: Ilustrace architektury Apollo Federation [2]

Pro udržitelnost platformy jsou tyto problémy zásadní a byla proto navržena

zásadní úprava architektury prostřednictvím Apollo Federace [2]. Jedná se o nástroj umožňující skládat GraphQL schéma („supergraf“) z menších celků, tzv. „subgrafů“ (obr. 4.3).

Díky federaci je vývojářům integrovaných systémů umožněno vyvíjet **pouze GraphQL rozhraní**, se kterým přímo komunikuje frontend. Vývojáři se tak zbaví nadbytečné implementaci REST rozhraní spolu s jejich napojením na GraphQL. Apollo Federation implementuje nová služba Hub Gateway, která bude sloužit jako nový vstupní bod pro rozhraní platformy. Tato služba bude skládat supergraf ze subgrafů služeb a Endpoint service, která zůstane v provozu kvůli pozvolné migraci a zpětné kompatibilitě.

Integrace systému Evaluací byla první plnohodnotná integrace systému do platformy Hub a jako taková odhalila řadu slabých míst. Některé z těchto problémů byly zapříčiněny „nevyzrálostí“ platformy (špatný přístup k logům, chybějící impersonace, tj. funkcionality, které nestihly být implementovány v průběhu integrace), jiné vycházely přímo z technického návrhu platformy (duplikace specifikací, pomalé GraphQL dotazy, nesrozumitelný Whoami formát…) nebo z organizačních nedostatků projektu (chybějící garant, nedostatečná meziprojektová komunikace…).

Uvedli jsme řadu problémů, některé se buď podařilo intenzivní snahou správců platformy vyřešit, nebo probíhá implementace jejich řešení. Prostřednictvím konzultací se stakeholdery jsme identifikovali přetrvávající nedostatky, které negativně ovlivňují integrační procesy a na které „má smysl“ (doposud neexistuje návrh jejich řešení) se v rámci této práce zaměřit:

1. chybějící podpora content managementu,
2. nesrozumitelnost Whoami formátu,
3. obtížná použitelnost monitorovacích služeb.

V následujících kapitolách se věnujeme návrhu jejich řešení.

Kapitola 5

Návrh kritérií hodnocení kvality řešení

Před samotným návrhem řešení identifikovaných nedostatků platformy nejprve definujeme metriky, kterými bude možné tato řešení kvantitativně ověřit vzhledem k výchozímu stavu. Při definici metrik vycházíme z metodologie „IEEE Standard for a Software Quality Metrics Methodology“ [3]. Zde je metrika kvality softwaru definována jako funkce, jejímiž vstupy jsou softwarová data a výstupem je numerická hodnota, kterou interpretujeme jako míru specifické vlastnosti softwaru ovlivňující jeho kvalitu. Dokument dále definuje softwarovou kvalitu:

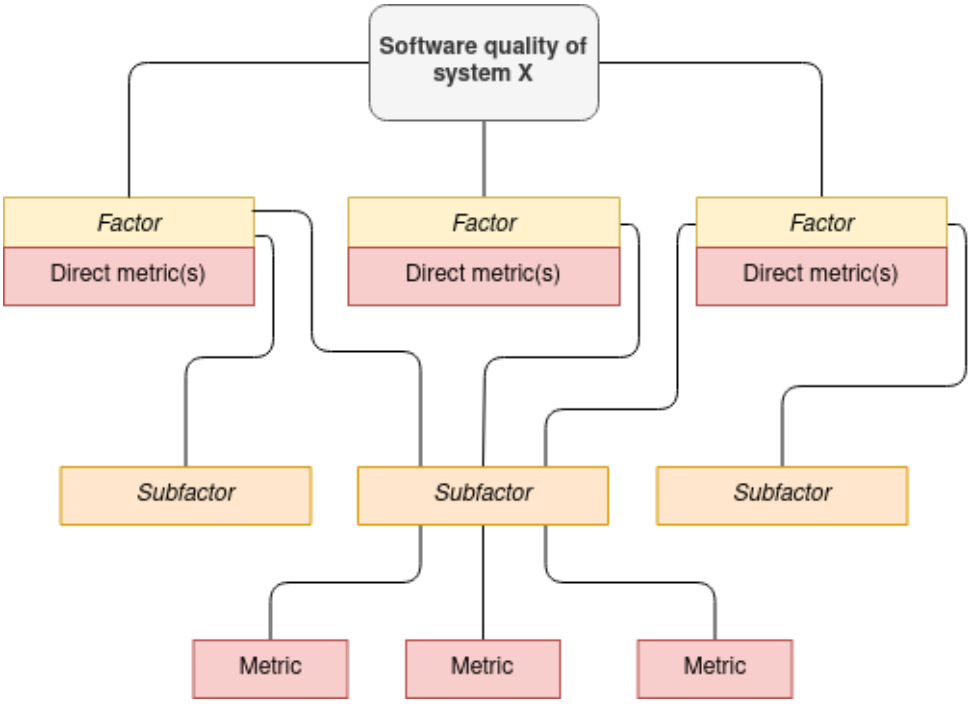
> „Software quality is the degree to which software possesses a desired combination of attributes. This desired combination of attributes shall be clearly defined; otherwise, assessment of quality is left to intuition. (...) In order to measure the software quality attributes an appropriate set of software metrics shall be identified.“

5.1 Metodologie definování metrik

Standard [3] definuje postup pro systematickou identifikaci metrik, jejich faktorů a podfaktorů. Výstup schématicky zobrazuje obr. 5.1.

Faktor. Faktor v metodologii představuje kritérium kvality, které je z projektového pohledu zaměřené na dopad na uživatele. Soubor těchto kritérií definuje softwarovou kvalitu systému. Každý faktor musí mít přiřazenou jednu či více přímých metrik, které udávají jeho kvantitativní ohodnocení, a každá z těchto metrik musí mít definovanou cílovou hodnotu.

Subfaktor. Faktorům je možné přiřadit subfaktory kvality, které představují softwarově zaměřená kritéria. Slouží především k dekomponování faktorů



Obrázek 5.1: Software quality metrics framework [3]

na menší měřitelné softwarové atributy - ty mohou být na sobě nezávislé a odpovídat více faktorům. Subfaktory mohou být rozčleněny na metriky odpovídající výstupům vývoje a procesům napříč vývojovými cykly. Na rozdíl od přímých (faktorových) metrik mohou být snadněji dostupné během vývoje, protože hodnoty přímých metrik mohou být obtížně získatelné či dokonce nedostupné.

Tento postup se dále užívá v pěti krocích procesu určenému dle metodologie [3] následovně:

1. *Identifikace kritéria kvality* - složení prioritizovaného seznamu faktorů.
2. *Identifikace metrik kvality* - relevantní metriky jsou zvoleny na základě postupu pro identifikaci metrik.
3. *Implementace metrik* - popis a sběr potřebných dat.
4. *Analýza výsledných hodnot metrik* - výpočet metrik, interpretace výsledku a ohodnocení kvality.
5. *Validace metrik* - porovnání výchozích a měřených metrik, zhodnocení zdali metriky řádně vypovídají o stavu daných faktorů.

5.2 Definice metrik

Dle metodologie nejprve identifikujeme kritéria kvality tvořená seznamem faktorů. Faktory jsme identifikovali na základě organizačních zkušeností, obecných standardů a uživatelské zpětné vazby. Seznam identifikovaných faktorů uvádí tabulka 5.1.

Faktor	Popis
Udržitelnost	Udává, jak složité je orientovat se v implementaci systému, zavádět do systému změny či rozšíření.
Použitelnost	Rozsah služeb poskytovaných systémem, obtížnost jejich používání a pokrytí uživatelských potřeb.
Spolehlivost	Schopnost systému poskytovat služby spolu se zachováním určité úrovně výkonnosti během daného období.
Agilita	Složitost procesu nasazování, možnost reprodukce jeho stavu.
Přehlednost	Možnost vhledu do aktuálního stavu systému.

Tabulka 5.1: Seznam faktorů tvořící kritéria kvality systému

Jako uživatele chápeme členy integračních týmů, identifikace faktorů proto proběhla s ohledem na DX, neboli „developer experience“ [41]:

> „DX consists of experiences relating to all kinds of artifacts and activities that a developer may encounter as part of their involvement in software development. These could roughly be divided into experiences regarding i) development infrastructure (e.g. development and management tools, programming languages, libraries, platforms, frameworks, processes, and methods, ii) feelings about work (e.g. respect, attachment, belonging), and iii) the value of one's own contribution (e.g. alignment of one's own goals with those of the project, plans, intentions, and commitment). (...)“

Termín „developer experience“ pokrývá soubor aktivit a vstupů, se kterými se vývojář (dle definice kdokoli podílející na vývoji) setká v rámci vývoje softwaru. Zlepšením DX tedy cílíme na celkový subjektivní dojem zúčastněných osob. O systému/nástroje s dobrým DX můžeme prohlásit, že

- vede uživatele k dosažení cílů správným[1] způsobem,
- odrazuje uživatele od vytváření řešení špatným[2] způsobem,

[1] Odpovídající zavedeným standardům a nejlepším praktikám řemesla.
[2] Odporující standardům či přímo poškozující kvalitu produktu.

- odstiňuje uživatele od repetitivních činností,
- poskytuje rychlou odezvu,
- poskytuje přesný vhled do jeho aktuálního stavu.

Z analýzy proběhlé integrace Evaluací je zřejmé, že aktivně probíhá vývoj platformy se snahou o pokrytí jejích nedostatků. Stav implementace je tedy poměrně dynamický. Abychom definovali metriky, které budou co nejméně ovlivněny již probíhajícími úpravami, musíme se zaměřit pouze na některé faktory kvality, resp. části systému.

V předchozí kapitole jsme definovali konkrétní problémy, kterými se budeme dále zabývat. Pro zachycení těchto problémů jsme vytvořili subfaktory faktorů „Použitelnost“, „Udržitelnost“ a „Přehlednost“, viz v tabulce 5.2.

Faktor	Subfaktor	Popis
Udržitelnost	Upravitelnost	Složitost strukturálního návrhu, míra provázání komponent a rozdělení jejich odpovědností, duplikace kódu, vhodné abstrakce [42]
Použitelnost	Kvalita zpětné vazby	Rychlost odezvy systému, složitost ověření zavedených změn, srozumitelnost odezvy
	Náročnost používání	Náročnost vykonávání aktivit prostřednictvím systému
Přehlednost	Monitorování	Dostupnost logů a relevantních dat o stavu systému.

Tabulka 5.2: Seznam subfaktorů kritérií kvality

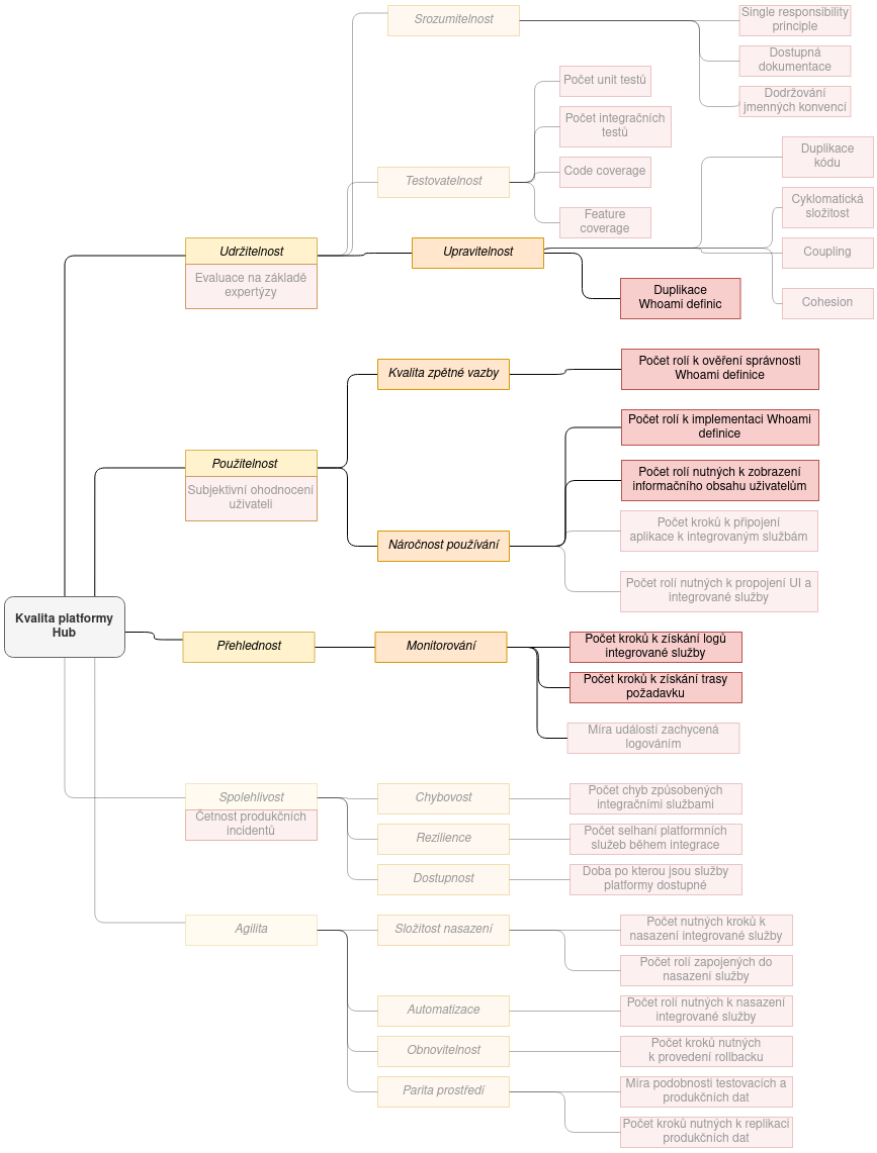
Subfaktor *Upravitelnost* popisuje metrika **Duplikace Whoami definic**, která udává počet míst, na kterých se duplikuje struktura Whoami komponent definovaných uživatelem. Ve výchozím stavu se jedná o dokumentaci platformy, frontendovou implementaci, backendovou implementaci integrované služby a záznam v Impersonation service, tedy čtyři místa nutné duplikace. Pro zlepšení faktoru *Udržitelnost* je nutné hodnotu snížit, optimální hodnota je jedna. Subfaktoru *Kvalita zpětné vazby* přísluší hodnota metriky **Počet rolí k ověření správnosti Whoami definice**. V současném stavu je při definování Whoami komponent potřeba, aby jejich správnost ověřil analytik platformního týmu, frontend vývojář a backend vývojář platformy. V ideálním případě by byl počet rolí nula a zpětnou vazbu poskytl přímo systém. Dále je subfaktor *Náročnost používání* určen metrikami:

Počet rolí k implementaci Whoami definice. Implementace znamená vytvoření definice a její zanesení do systému. Za implementaci Whoami formátu zodpovídá její autor - analytik integrujícího týmu, vývojář frontendu, platformní vývojář backendu a backendový vývojář integrujícího týmu. V ideálním případě by byl potřeba pouze její autor.

Počet rolí nutných k zobrazení informačního obsahu uživatelům. Jak již bylo popsáno, platforma dosud nemá podporu pro ovládání obsahu uživatelem, pro zobrazení dočasných informačních hlášek je tedy zapotřebí role analytika integračního týmu, UI/UX návrháře, frontendového vývojáře a hlavního frontendového vývojáře platformy, který zodpovídá za nasazení nové verze. Optimální hodnota této metriky je jedna, tedy pouze analytik integračního týmu.

Subfaktor *Monitoring* se skládá z metriky **Počet kroků k získání logů integrované služby**, která je dána počtem interakcí nutných k přečtení aktuálních logů služby v aplikaci Kibana. Jedná se o zvolení prostředí, zapnutí filtrování podle jména služby a vyplnění jména služby, tedy tři kroky. V ideálním případě by byl zapotřebí pouze jeden krok, který by uživatele dovedl přímo k danému datovému pohledu. Další metrika **Počet kroků k získání trasy požadavku** analogicky popisuje použití služby Grafana, které vyžaduje přesun na záložku „Explore“, výběr typu hledání, zadání jména služby, spuštění dotazu a následné dohledání požadavku podle jeho data. Jedná se tedy o pět kroků, přičemž v optimálním stavu by měl být pouze jeden.

Návrhy řešení vedoucích k optimalizaci těchto metrik popisujeme v následující kapitole (viz tabulka 6.1). Na schématu (obr. 5.2) jsou k ilustraci zobrazeny i další identifikované metriky, na které se z výše popsaných důvodů v této práci nezaměřujeme.



Obrázek 5.2: Přiřazení metrik k faktorům a subfaktorům kvality (měřené jsou pouze zvýrazněné položky)

Kapitola 6

Návrh optimalizace hodnot metrik

V následující kapitole se věnujeme návrhu konkrétních způsobů, jak optimalizovat hodnoty výše definovaných metrik.

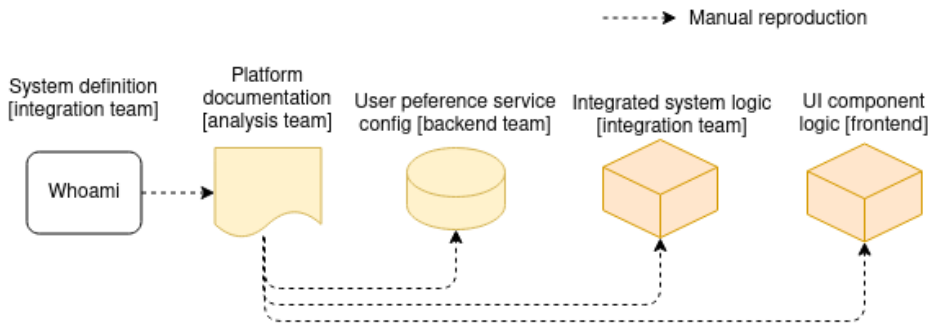
6.1 Whoami workflow

Ze systémové analýzy platformy vyplývá, že chce-li uživatel definovat formu Whoami pro svůj systém, musí být specifikována na několika místech:

1. ve sdílené dokumentaci platformy (ta je součástí repozitáře přístupného zaměstnancům CZM),
2. v User preference service, kde se ukládá členem platformního týmu do kódu,
3. v logice své aplikace, která poskytuje seznam modulů na základě práv uživatele,
4. ve frontendové implementaci,
5. (nemusí, ale v podstatě je to nutné) v analýze svého systému, která obsahuje pravidla pro autorizaci komponent.

Záznam v dokumentaci je primární, frontend tým podle něj definuje identifikátory komponent, které pak ověřuje podle Whoami záznamu. Zde je zřejmé, že jakákoli změna definice vyžaduje duplicitní zadávání (obr. 6.2) těch samých údajů a komunikaci napříč několika týmy - integrace, analýza, backend a frontend.

UI navíc pro vývojářské účely obsahuje přepínač, který pozastaví kontrolu Whoami a poskytne uživateli přístup do všech částí aplikace. Vývojáři nejsou systémem nuceni Whoami používat, takže integrace systémů může probíhat



Obrázek 6.1: Vizualizace aktuální přidávání Whoami definice systému

(a probíhala) bez ohledu na nesouvislosti Whoami definice či dokonce její implementaci - a to i přes to, že se jedná o klíčovou funkcionalitu platformy.

Z uživatelského hlediska proto navrhujeme zavedení těchto změn:

1. vytvoření jednotného zdroje Whoami definice systému,
2. vytvoření nástroje pro vkládání a úpravu Whoami definice systému.

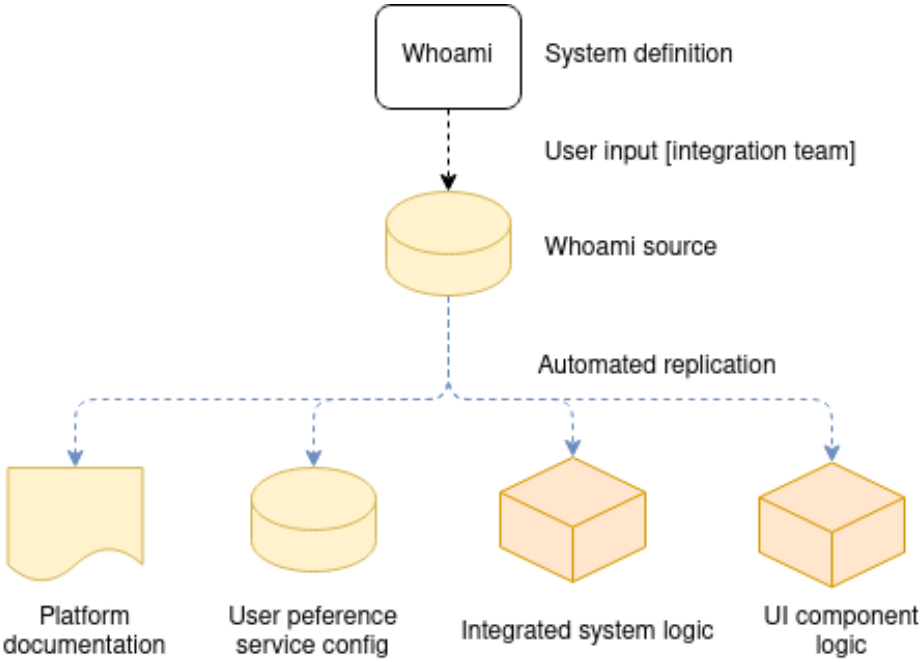
Ačkoli je primárním zdrojem definice dokumentace platformy, neexistuje žádný automatizovaný krok mezi touto dokumentací a záznamy v programu. Bod 1 tedy představuje implementaci řešení, které nebude po uživateli vyžadovat žádnou manuální replikaci definice.

Bodem 2 cílíme k omezení potřebné komunikace napříč týmy na základě následujícího předpokladu: *členové integračních týmů nerozumí funkci a struktuře Whoami formátu a je nutné jeho implementaci vícekrát konzultovat.* Prostřednictvím interaktivního nástroje lze uživatelům poskytnout zpětnou vazbu a vedení, které v ideálním případě nahradí jak dokumentaci, tak i nutnou výpomoc od členů platformních týmů. Nabízí se jednoduchá varianta implementace v podobě strukturovaného souboru v např. git repozitáři, ke kterému by uživatel získal přístup. Ta ovšem po uživateli vyžaduje porozumění formátu souboru (např. YAML či JSON). Samotný soubor dále neposkytuje zpětnou vazbu ohledně významu vyplněných dat a jejich korektnosti, kterou by v rámci nástroje šlo „staticky“ kontrolovat bez nutnosti nasazení či ověření členem platformního týmu.

Naopak nevýhodou uchování definic v aplikaci je ztráta automatické replikace napříč prostředími. Soubor spravovaný v repozitáři se totiž replikuje při „mergování“ větví a vytvoření nové verze. Pro tento účel bude potřeba uživatelům poskytnout jednoduchý způsob pro migraci celých definic.

6.1.1 Jednotný zdroj Whoami definice

Zdroj definice musí být strojově zpracovatelný (současná definice je vedená v markdown souborech) a úložiště musí poskytovat API pro získání a definic.

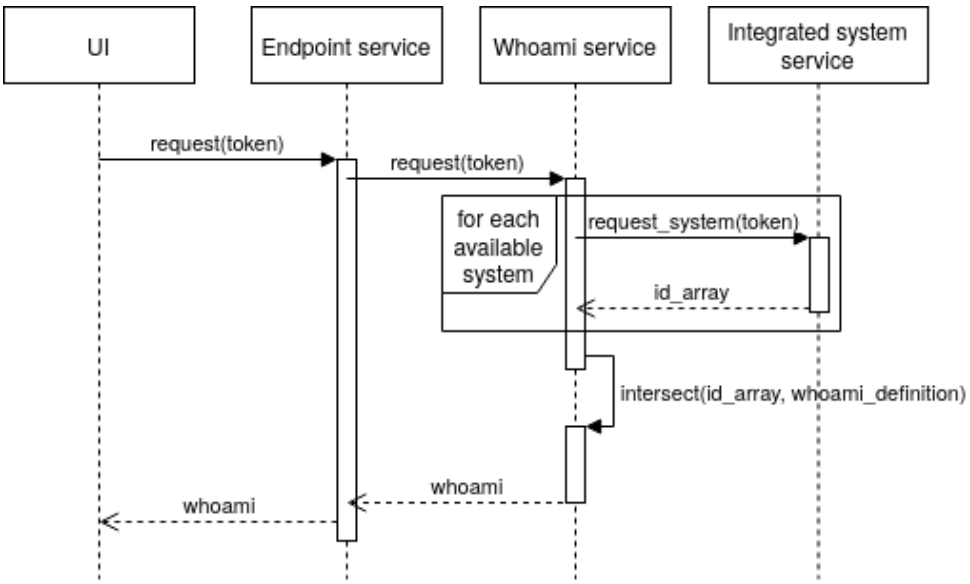


Obrázek 6.2: Návrh přidávání Whoami definice systému prostřednictvím jednotného zdroje

V aktuálním stavu definice agend vkládá do User preference služby, soubor ve formátu JSON obsahuje seznam agend a příslušných modulů. Agenda je totiž koncepčně zamýšlena i jako množina libovolných modulů, kterou si koncový uživatel může sestavit dle svého uvážení (tzv. „custom“ agenda). Tento use case však nebyl plně realizován a **pro ukládání definic proto navrhujeme použít službu Whoami service**.

Tímto se změní proces (obr. 3.5) Whoami dotazu tak, že Whoami service nebude žádat User preference o data, ale protne odpovědi dotázaných služeb přímo s uživatelskými definicemi. (V případě zavedení „custom“ agend lze User preference opět do procesu zapojit, jeho role se však změní tak, že bude poskytovat pouze tyto agendy, které rozšíří základní seznam).

Jisté míře duplikace se ovšem nevyhneme. Aby mohla aplikační logika integrované služby na endpointu `/whoami` validovat komponenty Whoami, musí v době kompilace znát alespoň jejich *id*. Uložením celé definice do Whoami service se však služby zbaví zodpovědnosti za strukturu komponent (viz systémovou analýzu - User preference obsahuje agendy a seznam jejich identifikátorů, struktura modulů je obsažena v JSON odpovědi služeb). Validační endpoint proto může vracet **pouze seznam** povolených identifikátorů bez



Obrázek 6.3: Sekvenční diagram odbavení Whoami požadavku bez User preference service

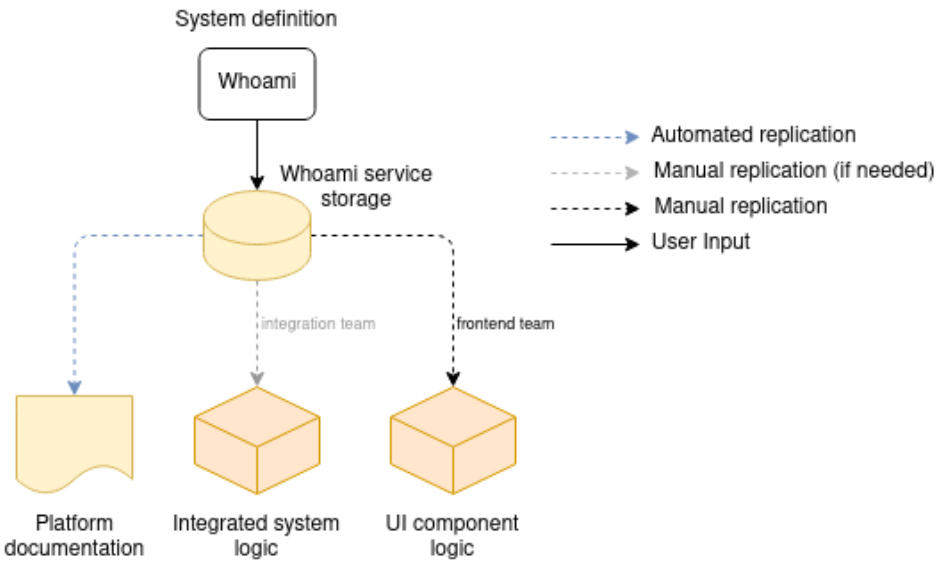
duplikace celé struktury. Validační logiku lze případně kompletně nahradit, čemuž se věnujeme v další části řešení.

Platformní dokumentace se udržuje v GitLab wiki repozitáři. V zásadě se jedná o git repozitář, který renderuje soubory ve formátu markdown do strukturované podoby wiki stránek. GitLab poskytuje REST API pro nahrávání commitů, tímto kanálem lze poměrně jednoduše replikovat definici z Whoami service do platformní dokumentace. Při každé změně definice z ní stačí automaticky vygenerovat markdown soubor a ten pak přes Commits API[43] odeslat. Tímto se dále vyřeší problémy vzniklé nesouhlasnými záznamy v dokumentaci a implementaci.

Pro validaci UI komponent na straně frontendu platí stejné omezení jako integrační služby. Hierarchie frontendových komponent odpovídá struktuře Whoami, což přináší duplikaci jak její struktury, tak identifikátorů. Pro zamezení duplikace by bylo možné změnit stromovou hierarchii implementace UI agend na mapování identifikátor - komponenta. Renderovací algoritmus by poté zobrazoval komponenty na základě URL identifikátoru v příslušné sekci.

Definici Whoami lze perzistentně ukládat jako soubor součástí zdrojových kódů služby (tak tomu je v User preference service) nebo v databázi. Výhodou jednoduchého souboru je, že v rámci repozitáře se poměrně snadno upravuje a verzuje. Nevýhodou je, že uživatel musí mít práva na publikování změn do repozitáře služby. Změny jednak vyžadují revizi zodpovědných vývojářů a udělování přístupů uživatelům platformy může představovat bezpečnostní riziko. Zavádění změn by navíc po uživateli vyžadovalo nároky v podobě znalosti implementačních detailů služby. Dalším nedostatkem tohoto řešení je, že pro propagaci změn do běžící aplikace je nutné nasazení nové verze.

Databáze naopak umožňuje zavádění změn v běžící aplikaci. Pro použití databáze je nutné implementovat logiku pro její připojení a vkládání dokumentací, ta nicméně odstiňuje uživatele od interních detailů a poskytuje úzké rozhraní. Proto navrhujeme definice ukládat v MongoDB[44] databázi, která je vhodná pro ukládání JSON dokumentů a již se používá v dalších platformních službách.



Obrázek 6.4: Návrh přidávání Whoami definice systému do Whoami service

Pro přidávání a úpravu definicí je nutné uživateli vystavit příslušné rozhraní, kterému se věnujeme v následující části.

Dopad na hodnoty metrik. Aplikací jednotného zdroje definic předpokládáme snížení hodnoty **Duplikace Whoami definic** z hodnoty čtyř na hodnotu jedna, protože strukturu Whoami pro integrovaný systém by mělo stačit ukládat pouze ve Whoami službě.

6.1.2 Nástroj pro správu Whoami definice systému

Nástroj by měl vyhovovat následujícím funkčním požadavkům:

1. Systém umožní vytváření a úpravu Whoami definic.
2. Systém poskytne kontrolu Whoami definice.
3. Systémem lze nastavit povolené Usermap role pro Whoami komponenty.
4. Systémem lze nastavit přístup konkrétním uživatelům pro Whoami komponenty.

5. Systémem lze ověřit přístup ke komponentám na základě uživatelských jmen.

6. Systém umožní správu přístupů k úpravě definic.

Bod 1 slouží jako rozhraní pro primární zdroj definice popsaný výše. Druhý funkční požadavek slouží jako interaktivní dokumentace (či návod), která by měla uživateli přiblížit účel Whoami komponent. Kontrola definice poskytne automatizaci aktivit, které je potřeba vykonat napříč několika týmy, např. kontrola správné struktury definice či výskytu duplikací identifikátorů.

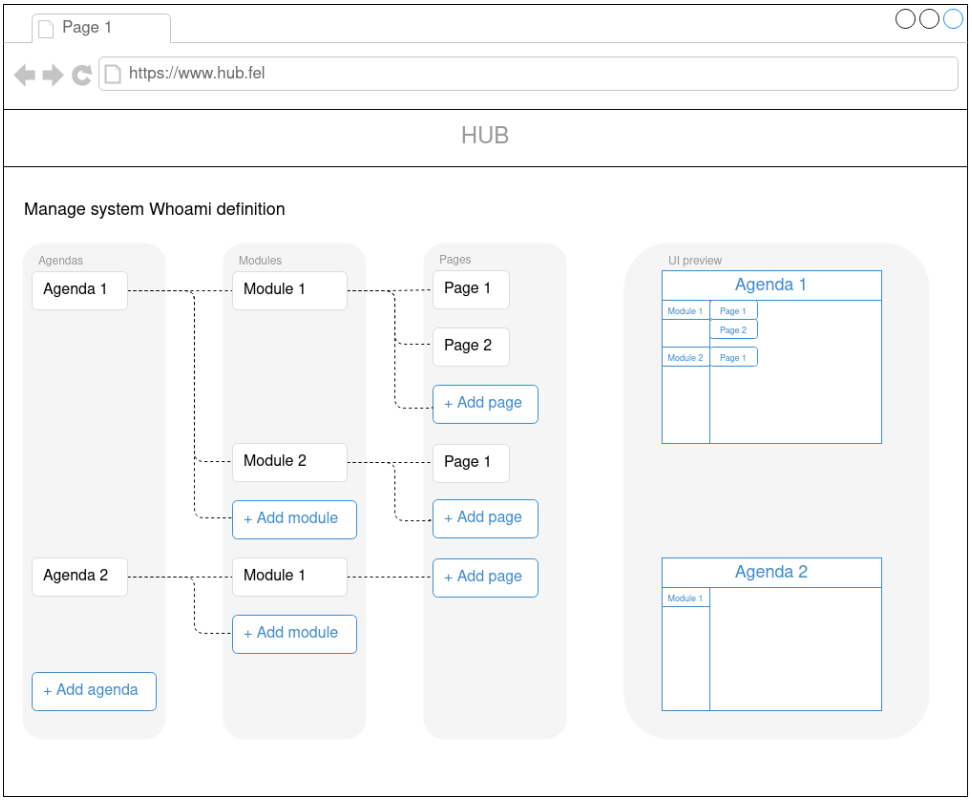
Třetí funkční požadavek vychází ze zkutečnosti, že většina implementací `/whoami` endpointů závisí na Usermap rolích přihlášených uživatelů. Nástroj pro správu definic nabízí prostor pro rozhraní k mapování těchto rolí k příslušným komponentám. Seznam mapovaných rolí lze pak ukládat (a ověřovat) ve Whoami service - čímž se (v systémech, které používají pouze Usermap role) může kompletně předejít implementaci endpointu a jinak nezbytné duplikace definice. Tento způsob implementace kontroly rolí navíc zbavuje zodpovědnosti vývojáře integračního týmu a za jejich zanesení zodpovídá analytik, který je i zároveň (z business pohledu) definuje. Čtvrtý požadavek rozšiřuje třetí požadavek granulárnější možností udělení přístupu (např. pro administrátory).

Přínosem autorizace na úrovni Whoami service je dále snížení počtu síťových požadavků. Máme-li n služeb, které využívají Usermap role pro kontrolu Whoami, pak během jednoho požadavku rozešle Whoami service n dotazů do těchto služeb, které dohromady odešlou přibližně (v závislosti na případných či chybějících optimalizací) n dotazů na Usermap API - ve výsledku $2n$ síťových dotazů. Při kontrole na úrovni Whoami service stačí pouze jeden dotaz na Usermap API, což je oproti $2n$ nejen zrychlení, ale i snížení náchylnosti na síťové chyby.

Pátý požadavek - kontrolování přístupů slouží ke zlepšení zpětné vazby prostřednictvím možnosti ověření autorizačních definic. V případě definování kontroly přes seznam Usermap rolí či uživatelských jmen lze totiž definici ověřit bez nutnosti implementace a nasazení aplikační logiky. Pro ostatní služby lze tímto ověřovat stávající implementaci.

Implementace UI nástroje vznikne v rámci existujícího rozhraní platformy, vzhledem k již existující infrastruktuře jako jsou frontendové komponenty a komunikace s backendovými službami. Pro tento účel vznikne v rámci platformy nová agenda sloužící pro administrativní účely. Obrázek 6.5 zobrazuje stromové znázornění agend, modulů a stránek spolu s tlačítky pro jejich přidávání.

Ke splnění funkčních požadavků nástroje a implementace jednotného zdroje je potřeba rozšířit Whoami service o následující funkce:



Obrázek 6.5: Wireframe funkcionality přidávání Whoami komponent

CRUD operace nad Whoami definicemi - základní funkce pro správu definic.

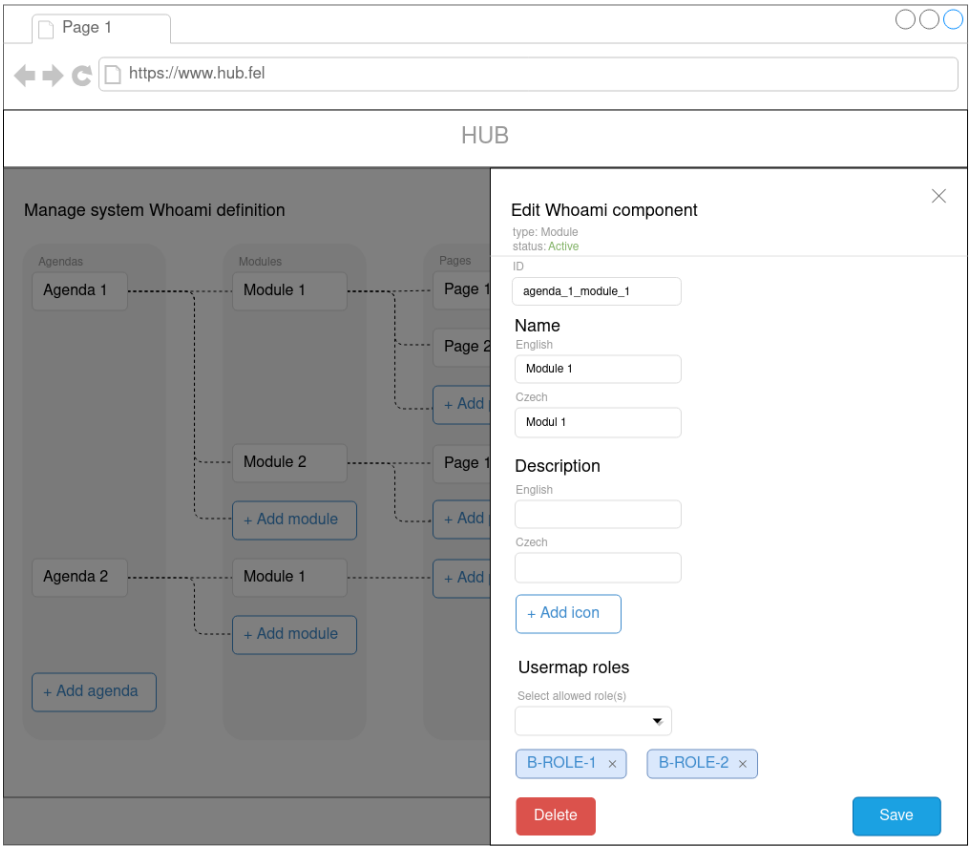
CRD operace pro úpravu přístupů - Přístup uživatelů k administrační agendě nelze udělit na základě vnějšího zdroje rolí. Je proto potřeba ukládat seznam uživatelských jmen s přístupy k této funkci.

CRUD operace pro úpravu vlastnictví definic - Uživatel by měl mít možnost upravovat pouze ty agendy, které vytvořil, nebo ke kterým mu byl udělen přístup. Dále by měl mít možnost upravit přístup uživatelů ke svým definicím.

Kontrola komponent na základě rolí - Funkcionalita endpointu `/user/whoami` se rozšíří o stahování rolí uživatele z Usermap service a jejich ověření oproti rolím přiřazeným k identifikátoru (pouze pokud nějaké přiřazeny jsou).

Validace komponent na základě uživatelského jména - Slouží k zobrazení komponent, ke kterým má daný uživatel přístup.

Nový formát Whoami endpointu. Díky novému zdroji definicí již nemusí služby posílat strukturovaný objekt, ale pouze seznam identifikátorů (pokud



Obrázek 6.6: Wireframe formuláře pro úpravu Whoami komponent

systémům nestačí pokrytí autorizačních pravidel pomocí Usermap rolí na úrovni Whoami služby). Pro funkcionalitu testování přístupů prostřednictvím nástroje pro správu whoami je ovšem potřeba, aby služby poskytovaly Whoami odpověď na základě uživatelského jména a ne pouze na základě tokenu. Pro tento účel navrhujeme nový formát jako `/whoami/<username>`, který vrací JSON pole řetězců.

Databázové záznamy definic je dále potřeba rozšířit o atribut agendy, který udává uživatelská jména uživatelů s přístupem ke správě definic. Podobě konkrétních endpointů a GraphQL schématu se v této části nebudeme věnovat, vychází ze zavedených REST a GraphQL standardů. Do uživatelského rozhraní platformy je pak potřeba implementovat:

Administrační agendu a stránku pro správu Whoami definic - Toto řešení respektuje koncepční rozdělení komponent platformy a vyžaduje pouze minimální konfiguraci.

Vstup pro vytváření Whoami komponent - Nástroj by měl vhodně zobrazit jejich hierarchii a stav. Tímto zobrazením by mělo být možné i ověřit, k jakým komponentám má jaký uživatel přístup.

Formulář pro správu detailů komponent - Formulář pro podrobnější úpravu atribut komponent (obr. 6.6) jako jsou povolené role, lokalizované názvy apod.

Dopad na hodnoty metrik. Vytvořením nástroje snížíme hodnotu metriky **Počet rolí k ověření správnosti Whoami definice** na nulu, protože nástroj by neměl umožnit vytvořit definice v chybném formátu. Díky nástroji není vyžadována implementace `/whoami` endpointu backendovým vývojářem integračního týmu, avšak frontendová implementace i po zmíněné úpravě vyžaduje alespoň zanesení identifikátorů. **Počet rolí k implementaci Whoami definice** se tedy může snížit minimálně na dvě role.

6.2 Přehled aktivních služeb

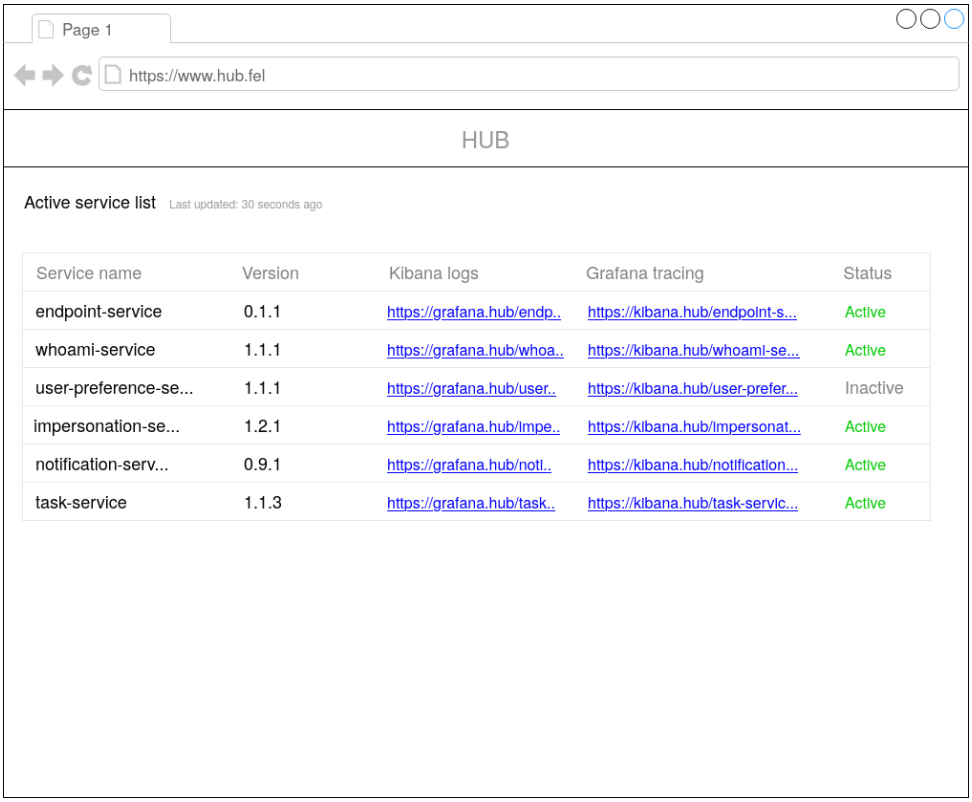
V současné době uživatel má přehled o aktivních službách v daném prostředí prostřednictvím monitorovacích služeb (má-li k nim přístup) nebo vyzkoušením UI funkcí, které jsou s ní propojené. Seznam aktivních služeb lze získat pomocí Eureka discovery service, která umožňuje periodické kontrolování registrovaných klientů a poskytuje jejich výčet prostřednictvím REST API. Pro vytvoření přehledu služeb tedy stačí namapovat volání daného endpointu na GraphQL a v UI zobrazit jednoduchou tabulku. Pro umístění tabulky se vybízí vytvořit novou stránku administrativní agendy navržené v předešlé části.

Přehled by šlo rozšířit i o informace jako je aktuální verze nasazených služeb, či odkazy na monitoring se specifickými pohledy na logy služeb a jejich dotazů. Specifickými odkazy lze usnadnit práci s monitorovacími nástroji, mají poměrně sofistikované rozhraní a pro uživatele bývá náročné se v nich zorientovat, obzvlášť pokud se nejedná o vývojáře, kteří je využívají častěji než např. analytici. K vytvoření odkazů stačí název služby a typ prostředí. Pro získání verze služby je potřeba poskytnout endpoint, který by ji předával z konfigurace služby. Tento endpoint lze implementovat v knihovně LibCommon, čímž by nevznikal žádný požadavek na vývojáře služby.

Dopad na hodnoty metrik. Pomocí přehledu se lze dostat na příslušné logy služby v aplikaci Kibana i Grafana dvěma kroky (přesměrování na přehled, kliknutí na odkaz logů), což upravuje hodnotu metriky **Počet kroků k získání logů integrované služby**.

6.3 Trasování požadavků od UI

Monitoring platformy poskytuje sledování („tracing“) průběhu požadavku napříč službami pomocí Spring Cloud Sleuth [33]. Dílčím požadavkům je



Obrázek 6.7: Wireframe rozšířené tabulky aktivních služeb

přidělen identifikátor *traceId*, podle kterého je možné je shlukovat a zobrazovat jejich návaznost v nástrojích Kibana či Grafana. V aktuálním stavu však pro uživatele neexistuje spojení mezi požadavky z UI a dílčími požadavky uvnitř platformy. Uživatel zjišťující průběh požadavku tedy musí návaznost dohledat mezi všemi proběhlými požadavky, což je nejen pracné, ale vyžaduje i jistou znalost implementace systému.

Pro usnadnění monitorování a propojení *traceId* s UI požadavky navrhujeme zveřejňovat tento identifikátor prostřednictvím hlaviček HTTP odpovědí Endpoint service. Kromě samotného identifikátoru lze navíc uživateli zobrazit přímo vygenerované odkazy (podobně jako v rozšířeném přehledu služeb) do monitorovacích nástrojů, což může výrazně usnadnit celý proces. V UI platformy pak lze v rámci vývojářských nástrojů zobrazit seznam požadavků, který k názvu GraphQL dotazu přiřadí příslušné odkazy.

Dopad na hodnoty metrik. Díky seznamu dotazů lze na pohled trasy požadavku v aplikaci Grafana přejít otevřením seznamu a kliknutím na příslušný odkaz, čímž dostaneme hodnotu dva pro metriku **Počet kroků k získání trasy požadavku**.

6.4 Podpora content managementu

Podporou content managementu rozumíme poskytování funkcionality pro upravování obsahu UI bez nutnosti technických znalostí a zásahu do implementace systému. Vycházíme zde z požadavků na typ obsahu a komponent, které vyplynuly z produkčního běhu platformy.

Jedná se především o komponenty sloužící k upozornění uživatele integrovaných aplikací na výluky či dodání nápomocných informací. Tento obsah byl pokaždé implementován frontendovým týmem a nasazen jako nová verze UI. Poskytnutím funkcionality pro správu tohoto obsahu je pak integrující uživatel schopen informovat uživatele své aplikace dle libosti bez zásahu jiného týmu a nasazování nové verze.

Obsah navrhujeme ukládat v nové službě Content service prostřednictvím MongoDB v následujícím schématu:

```
{
  "title": "Content",
  "properties": {
    "parentId": { "bsonType": "string"},
    "from": { "bsonType": "date"},
    "to": { "bsonType": "date"},
    "hidden": { "bsonType": "bool"},
    "type": { "bsonType": "string", "enum": ['Alert', 'Help'] },
    "properties": {
        "bsonType": "array",
        "items": {
            "bsonType": "object",
            "title": "ContentProperties",
            "properties": {
                "key": "string",
                "value": "string"
            }
        }
    }
  "required": ["parentId", "type", "properties"],
}
```

UI při renderování Whoami komponenty (např. modul) odešle dotaz na obsah, kde *parentId* odpovídá jejímu identifikátoru. Pokud nějaký obsah existuje, zobrazí komponentu příslušnou jeho typu. Pro položky lze nastavit od a do kdy má být obsah přístupný. Typy obsahu bude možné nadále přidávat dle potřeby, v rámci tohoto řešení se soustředíme na vytvoření základních komponent a poskytnutí prostředí pro budoucí rozšíření.

```
{
  "title": "Content",
  "properties": {
    "parentId": { "bsonType": "string"},
    "from": { "bsonType": "date"},
    "to": { "bsonType": "date"},
    "hidden": { "bsonType": "bool"},
```

```
    "type": { "bsonType": "string", "enum": ['Alert', 'Help'] },
    "properties": {
        "bsonType": "array",
        "items": {
            "bsonType": "object",
            "title": "ContentProperties",
            "properties": {
                "key": "string",
                "value": "string"
            }
        }
    }
  "required": ["parentId", "type", "properties"],
}
```

Dopad na hodnoty metrik. Díky funkcionalitě správy obsahu se hodnota metriky **Počet rolí nutných k zobrazení informačního obsahu uživatelům** sníží na jedna, protože již není potřeba navrhovat, implementovat a nasazovat dané UI komponenty.

6.5 Předpokládané hodnoty metrik řešení

Předpokládané hodnoty metrik vysvětlujeme výše v podrobném popisu jednotlivých řešení, dohromady je shrnuje tabulka 6.1. Implementovat všechna navržená řešení v rámci této práce v dostatečné kvalitě by pravděpodobně nebylo proveditelné. Řešením byla proto na základě konzultací s hlavním manažerem projektu Hub přiřazena priorita podle jejich dopadu (od *A* do *C*, kde *A* je nejvyšší). **Jako nejpřínosnější byl zvolen nástroj na správu Whoami definic** a implementaci proto zaměříme na optimalizaci metrik **Počet rolí k ověření správnosti Whoami definice**, **Počet rolí k implementaci Whoami definice** a **Duplikace Whoami definic**.

Název metriky	Priorita	Výchozí hodnota	Optimální hodnota	Teoretická hodnota řešení
Počet rolí k ověření správnosti Whoami definice	A	3	0	0
Počet rolí k implementaci Whoami definice	A	4	1	2
Duplikace Whoami definic	A	4	1	1
Počet rolí nutných k zobrazení informačního obsahu uživatelům	B	4	1	1
Počet kroků k získání logů integrované služby	C	3	1	2
Počet kroků k získání trasy požadavku	C	5	1	2

Tabulka 6.1: Přehled hodnot metrik (řádky seřazeny sestupně dle priority)

Část II

Implementace navržených řešení

V následující části popisujeme realizaci vybraných částí navržených řešení. Z důvodů zmíněných v předchozí části se věnujeme pouze implementaci nástroje pro správu Whoami definic. Ze zbylých návrhů lze případně vytvořit technická zadání pro vývojáře platformy.

Implementace nové správy Whoami představuje poměrně významný zásah jak do existující codebase, tak do přístupu k platformě z pohledu integrujícího uživatele. Od výsledného produktu je tedy vyžadována možnost pozvolné migrační strategie z předešlého řešení a zároveň zpětná kompatibilita se systémy, které pro migraci případně nemají dostupné prostředky.

V backendové infrastruktuře proto vznikne nová Whoami service, která bude akceptovat předešlý formát autorizačních odpovědí a nahradí tak původní službu bez dopadu na integrované systémy. Frontendová aplikace se kromě rozšíření o administrační agendu musí upravit tak, aby přijímala nový formát Whoami. Tento nový formát pouze rozšiřuje ten stávající, takže UI existujících agend by mělo také nadále fungovat bez zásahu. Vývojáři frontendu tedy mohou přecházet postupně refaktorováním příslušných komponent, čímž výsledně poskytnou plnou funkcionalitu nástroje pro správu definic. Cílem této implementace je vytvořit takovou sadu nástrojů (ať už API či kompletní UI), která bude snadnou použitelností motivovat uživatele k migraci ze starého formátu.

Kapitola 7

Nová Whoami service

Tato kapitola podrobně popisuje implementační detaily nové Whoami služby. Pro vytvoření nové služby (namísto rozšíření té stávající) jsme se rozhodli z těchto důvodů:

Rozdílné rozhraní Stávající služba poskytuje REST rozhraní pro endpoint service, kdežto ta budoucí již využije novou GraphQL federaci. Vyvíjíme tedy přímo GraphQL endpoint, což výrazně zjednodušuje celý proces. Tímto se ovšem zbavíme možnosti přepoužití REST endpointů stávající služby.

Statická definice klientů Stávající služba definuje Feign klient [45] pro každou integrovanou službu, které se dotazuje na `/whoami` endpoint. To má za následek, že při přidání nové služby je nutné vytvořit nový Feign klient soubor a rozšířit logiku pro stahování dat. V nové službě chceme tento proces zjednodušit a automaticky se dotazovat na služby, které specifikoval uživatel.

Resolver per klient Pro každý Feign klient definuje stávající služba i resolver pro zpracování jeho odpovědí. Tyto resolvery poskytovaly korekci dat v chybném formátu. Nová služba by měla odchytit tyto edge-casy uvnitř logiky dotazování, dokud se všechny služby nepřesunou na nový formát, který je výrazně jednodušší a měl by tak i zamezit podobným chybám.

Popsané části stávající služby jsou pro naše účely nepoužitelné a bude proto vhodné vytvořit zcela novou službu, která ji nahradí.

7.1 Základní prvky služby

Smyslem služby je uchovávání všech Whoami komponent a jejich autorizace. Předpis komponent je ukládán uvnitř tzv. Whoami definice:

Whoami definice. Definicí označujeme zásadní datovou položku, která obsahuje uživatelem definované komponenty. Kromě komponent obsahuje informaci o administrátorech (viz níže) a atributy pro připojení k integrovaným službám: jméno služby či URL.

Administrátor definice. Adminem definice je jakýkoli uživatel, který je uveden v seznamu administrátorů definice. Admin je autorizován k úpravě komponent, atributů definice či přidávání/odebírání administrátorů.

Root definice. Jako *Root* označujeme speciální definici, která obsahuje základní komponenty Whoami služby (s tím i administrační agendy). Atributy definice a komponent nelze upravovat, služba povoluje pouze úpravu autorizací komponent a přidávání/odebírání adminů.

V rámci služby neexistují žádné role jako např. „super-user“, které by opravňovaly uživatele ke specifickým akcím. Administrátor definice je jediný a dostačující autorizační prostředek, protože admin definice si může autorizovat jakoukoli z jeho komponent. Pokud bychom chtěli někomu udělit možnosti hypotetické „super-user“ role, stačí uživatele přidat do seznamu administrátorů *Root definice* (viz výše), ve které lze navolit práva k libovolné komponentě a tím i práva k jakékoli definici. Administrátory *Root* definice je také možné nakonfigurovat proměnnými prostředí. Služba pro autorizaci svých funkcí (viz dále) používá namísto rolí princip Whoami, tudíž funkce jsou spjaty s identifikátory komponent. Uživatel tedy může vykonat akci pouze v případě, že má přístup k dané Whoami komponentě.

7.2 GraphQL rozhraní

Stejně jako ostatní služby, nová Whoami service využívá již zmíněnou knihovnu LibCommon a autorizuje HTTP požadavky pomocí JWT tokenu v hlavičce. GraphQL rozhraní je jakožto REST endpoint kryt touto autorizací skrz Spring security [46] a do jeho handleru se dostanou pouze požadavky s platným tokenem a informaci o uživateli. V následujícím textu proto autorizací myslíme autorizaci až na úrovni business logiky.

Níže popisujeme schéma GraphQL rozhraní, které slouží jako API kontrakt mezi službou a jejími klienty (zatím pouze frontend). Pro vývoj rozhraní je použit framework DGS [47] spolu s pluginem pro generování kódu, který automaticky vytváří veškeré data transfer objekty definované schématem.

Služba poskytuje poměrně úzké rozhraní pro čtecí operace, neboli „GraphQL queries“:

my Query *my* slouží k získání dat aktuálně přihlášeného uživatele. Uživatel je reprezentován rozhraním *BaseUser*, které je implementováno typem

HubUser (jakýkoli uživatel) a *WhoamiServiceUser* (uživatel uložený v databázi služby, vážou se k nim definice či přístup do testovacího prostředí). *BaseUser* obsahuje pouze uživatelské jméno a seznam Whoami komponent. Vytváření seznamu popisujeme dále v rámci kapitoly o Whoami resolveru.

whoamiUserByUsername Query *whoamiUserByUsername* vrací uživatele podle uživatelského jména. Slouží především pro testování přístupu pro daného uživatele a dotaz je proto přístupný pouze pro administrátory definic.

whoamiServiceUsers Query *whoamiServiceUsers* vrací seznam uživatelů v databázi služby a je přístupné pouze pro uživatele, kteří mají přístup ke komponentě s identifikátorem *whoamiServiceUsers*.

whoamiDefinitionById Query *whoamiDefinitionById* vrací definici podle jejího identifikátoru. Dotaz je povolený pouze pro administrátora definice, či uživatele s přístupem ke komponentě *allDefinitions*.

allDefinitions Query *allDefinitions* vrací všechny definice v databázi služby. Dotaz je povolený pouze pro uživatele s přístupem ke stránce *allDefinitions*.

whoamiComponentById Query *whoamiComponentById* vrací Whoami komponentu podle jejího identifikátoru a je přístupné pouze pro administrátory definice, pod kterou komponenta patří. Typy komponent *WhoamiAgenda*, *WhoamiModule*, *WhoamiPage* a *WhoamiAction* implementují rozhraní *WhoamiComponent*.

WhoamiComponent. Rozhraní *WhoamiComponent* představuje jakoukoli komponentu Whoami stromu. Kromě lokalizovaných názvů a popisů obsahuje komentář (určený vývojářům), seznam Usermap rolí a uživatelských jmen s přístupem k této komponentě a atribut *isDisabled*, který slouží k znepřístupnění komponenty.

Autorizace atributů BaseUser a WhoamiServiceUser. Atributy *whoami* a *definitions* typů implementujících *BaseUser* rozhraní jsou zatíženy autorizačními pravidly tak, aby uživatel dotazující se na data jiného uživatele dostal pouze relevantní data. Pokud se tedy dotazujeme na Whoami komponenty jiného uživatele, dostaneme pouze ty, které spadají pod naše definice. Podobně tomu je i u definicí, kde žádající uživatel může získat pouze takové definice jiných uživatelů, kterými jsou také administrátory.

Error handling. V současné době není ukotvený formát pro chybné stavy dotazů, který by se používal napříč službami, což částečně pochází z nejasných požadavků frontend týmu a rozdílných implementací integrovaných systémů. Specifikace GraphQL je v tomto ohledu navíc poměrně ambivalentní, na rozdíl

od REST architektury nespecifikuje konkrétní způsob indikace chybových stavů uživateli. Frontendový error handling musel doposud vycházet především ze stavových kódů REST volání mezi službami, které byly propagovány až do GraphQL. Nový způsob indikace errorů by měl vytvořit prvotřídní podporu pro předávání informací pocházející z business logiky implementované backendem.

V GraphQL schématu služby tedy definujeme rozhraní *Error*, které navrhujeme použít jako nový standard pro indikaci chybových stavů. *Error* obsahuje jediný atribut *cause*, který udává důvod vzniku chyby a může nabývat hodnot *UNAUTHORIZED*, či *INVALID_INPUT* (tyto hodnoty byly pro účely služby dostačující, případná rozšíření ale nevylučujeme). Případné dotazy obohacené o chybové informace pak obsahují atribut pro požadovaná data (s hodnotou *null* v případě chyby) a atribut *error* typu implementujícím *Error*, s hodnotou *null* v případě bezchybné odpovědi.

Typ *UserInputError* slouží k popisu chybného uživatelského vstupu a rozšiřuje rozhraní *Error* o atribut *fields*. Položky seznamu *fields* obsahují identifikátor vstupu (zpravidla název parametru) a lokalizovanou zprávu o chybě, která je určená koncovému uživateli. Tímto způsobem chceme „vměstnat“ chybné hlášky do backendové implementace, která je primárním zdrojem validační logiky.

7.2.1 Mutace

Pro účely správy definic poskytuje služba oproti čtecím dotazům větší množství mutací, t.j. dotazů, které slouží k modifikaci dat:

createWhoamiDefinition Mutace pro vytvoření definice přijímá vstup atributů definice a je autorizovaná pouze pro držitele oprávnění *myDefinitions_create*. Vstup typu *WhoamiDefinitionUpdateAttributes* je sdílený s mutací pro úpravu definice.

copyWhoamiDefinition Uživatelé s přístupem k *myDefinitions_create* mohou také vytvářet definice jejich „kopírováním“, resp. vstupem této mutace je kompletní předpis definice spolu se všemi komponentami.

updateWhoamiDefinition, deleteWhoamiDefinition Mutace pro úpravu definice přijímá identifikátor dané definice, stejně jako mutace pro vymazání. Tyto operace jsou přístupné pouze pro administrátory definice. Mutace pro úpravu přijímá objekt atributů definice.

addAdminToDefinition, removeAdminFromDefinition Mutace pro přidání/odebrání administrátora vloží či vymaže uživatelské jméno ze seznamu administrátorů definice. Operace může vykonat pouze administrátor definice, či uživatel s přístupem k *allDefinitions_update_admin* komponentě.

addAdminToDefinition, removeAdminFromDefinition Mutace pro přidání/odebrání administrátora vloží či vymaže uživatelské jméno ze seznamu administrátorů definice. Operace může vykonat pouze administrátor definice, či uživatel s přístupem k *allDefinitions_update_admin* komponentě.

createWhoamiComponent Mutace pro vytváření komponenty vyžaduje identifikátor rodiče, identifikátor komponenty a povinné atributy viditelné uživatelem. Identifikátor by měl být srozumitelný a přesně vystihovat funkci komponenty, proto není automaticky generovaný. Whoami komponenty tvoří stromovou hierarchii a každá potřebuje ukazatel na svého předka. Kořenová komponenta typu *Agenda* neukazuje rodičem na komponentu, ale na definici, které náleží. Pokud tedy vstupní identifikátor rodiče přísluší definici, vytvoří se *Agenda*, pokud komponentě typu *Module*, vytvoří se *Page*, a tak dále. Komponenty mohou vytvářet pouze administrátoři v rámci svých definic (toto pravidlo platí i pro zbytek mutací komponent).

updateWhoamiComponent Mutace pro úpravu komponenty vyžaduje identifikátor spolu s atributy typu *WhoamiComponentUpdateAttributes*, které sdílí s mutací pro vytváření. U komponent vázaných k *Root* definici lze upravovat pouze atributy přístupu, konkrétně *allowedUsernames* a *usermapRoles*.

deleteWhoamiComponent Mutace *deleteWhoamiComponent* slouží k vymazání komponenty podle jejího identifikátoru.

updateWhoamiComponentSubtree Pro usnadnění práce s aplikováním přístupových atributů slouží mutace *updateWhoamiComponentSubtree* k úpravě povolených rolí či uživatelských jmen pro *všechny* terminální potomky dané komponenty.

replaceWhoamiComponent Mutace *replaceWhoamiComponent* mění rodiče komponenty, čímž přesouvá i celý její podstrom. Komponentu lze přemístit pouze mezi rodiči stejného typu.

(create/update/delete)WhoamiServiceUser Mutace pro CUD operace nad uživateli služby jsou povolené pro držitele příslušných identifikátorů *whoamiServiceUsers_create*, *whoamiServiceUsers_update* a *whoamiServiceUsers_delete*.

7.3 REST rozhraní

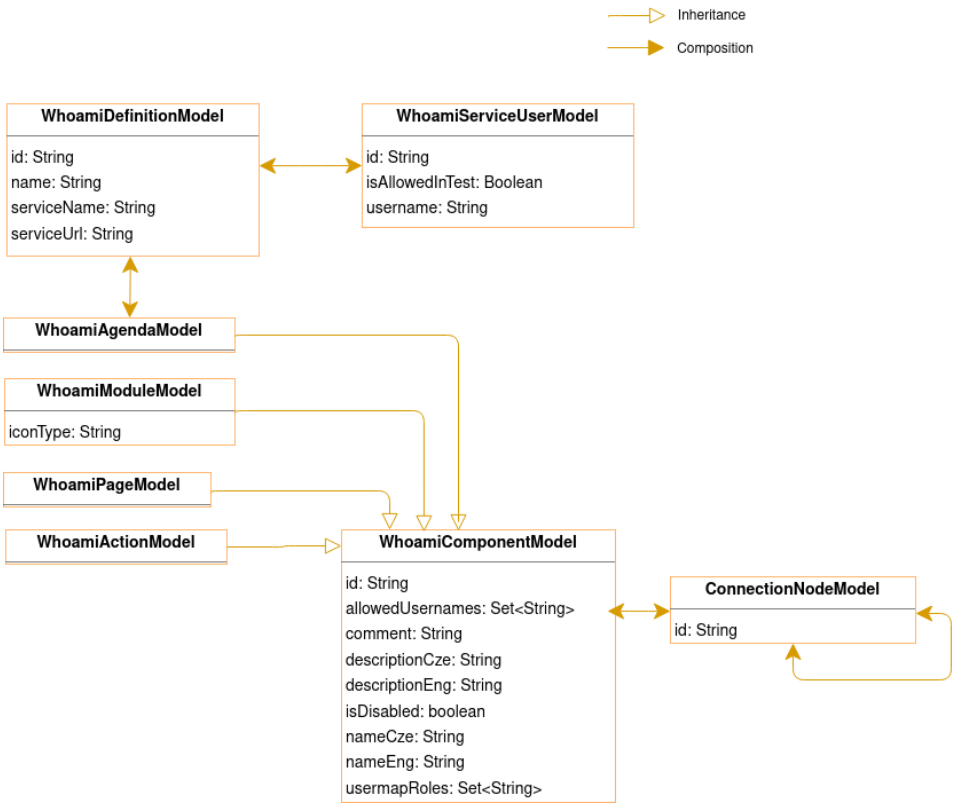
Služba poskytuje i rozhraní pro některé meziservisní požadavky na endpointech:

`admins/<username>` Nový formát whoami endpointů poskytuje seznam identifikátorů pro jakéhokoli uživatele. Pokud by integrovaná služba neměla poskytovat takové informace komukoli, může je autorizovat zavoláním endpointu `admins/<username>`, který vrací prázdnou odpověď se stavem 204, pokud je daný uživatel adminem některé definice ve službě Whoami (jiný uživatel by neměl mít potřebu dotazovat se na identifikátory jiné osoby než sebe).

`my/components/<componentId>` Tento endpoint vrací status 204, pokud má dotazující se uživatel přístup ke komponentě s identifikátorem *componentId*.

7.4 Entitní model

V původním návrhu služby byla pro ukládání dat zvolena dokumentová databáze MongoDB, nicméně kvůli relační struktuře komponent, definicí a administrátorů byla použita databáze PostgreSQL [48]. Implementaci PostgreSQL volíme kvůli jejímu použití v dalších integrovaných službách.

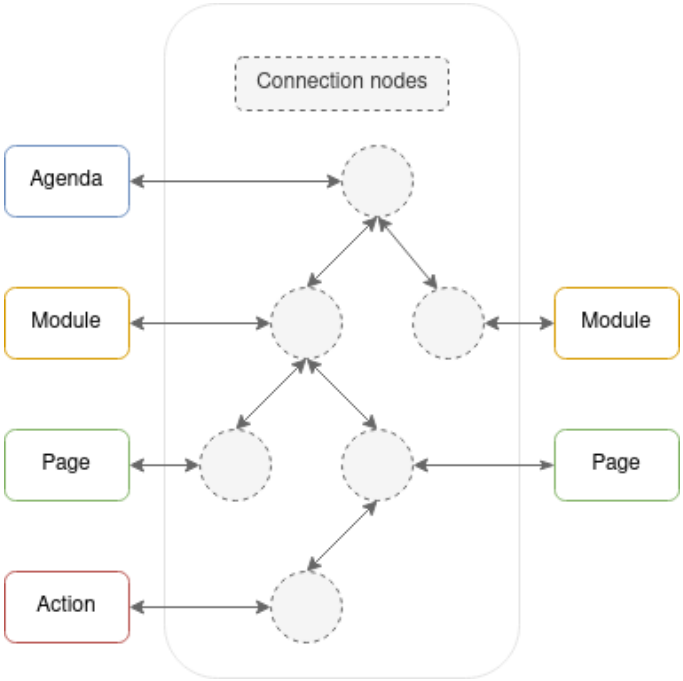


Obrázek 7.1: Entity relationship diagram backendového modelu

Diagram 7.4 zachycuje třídy persistentních entit. Protože DGS plugin generuje názvy typů tak, jak jsou definovány ve schématu, vzniklé data

transfer objekty nemohou obsahovat obvyklou příponu „Dto“. Názvy tříd modelů proto obsahují příponu *Model*, aby se tak odlišily od názvů DTO tříd.

Entity *WhoamiDefinitionModel*, *WhoamiServiceUserModel* a *WhoamiComponentModel* (spolu s potomky) víceméně odpovídají příslušným GraphQL typům, které popisujeme výše.



Obrázek 7.2: Schéma propojení uzlů *ConnectionNodeModel* a komponent

ConnectionNodeModel je implementační detail stromové struktury Whoami komponent, který nahrazuje rekurzivní vazbu mezi komponentami - ta ve spojení s dědičností komponent způsobovala řadu problémů s relačním mapováním objektů.

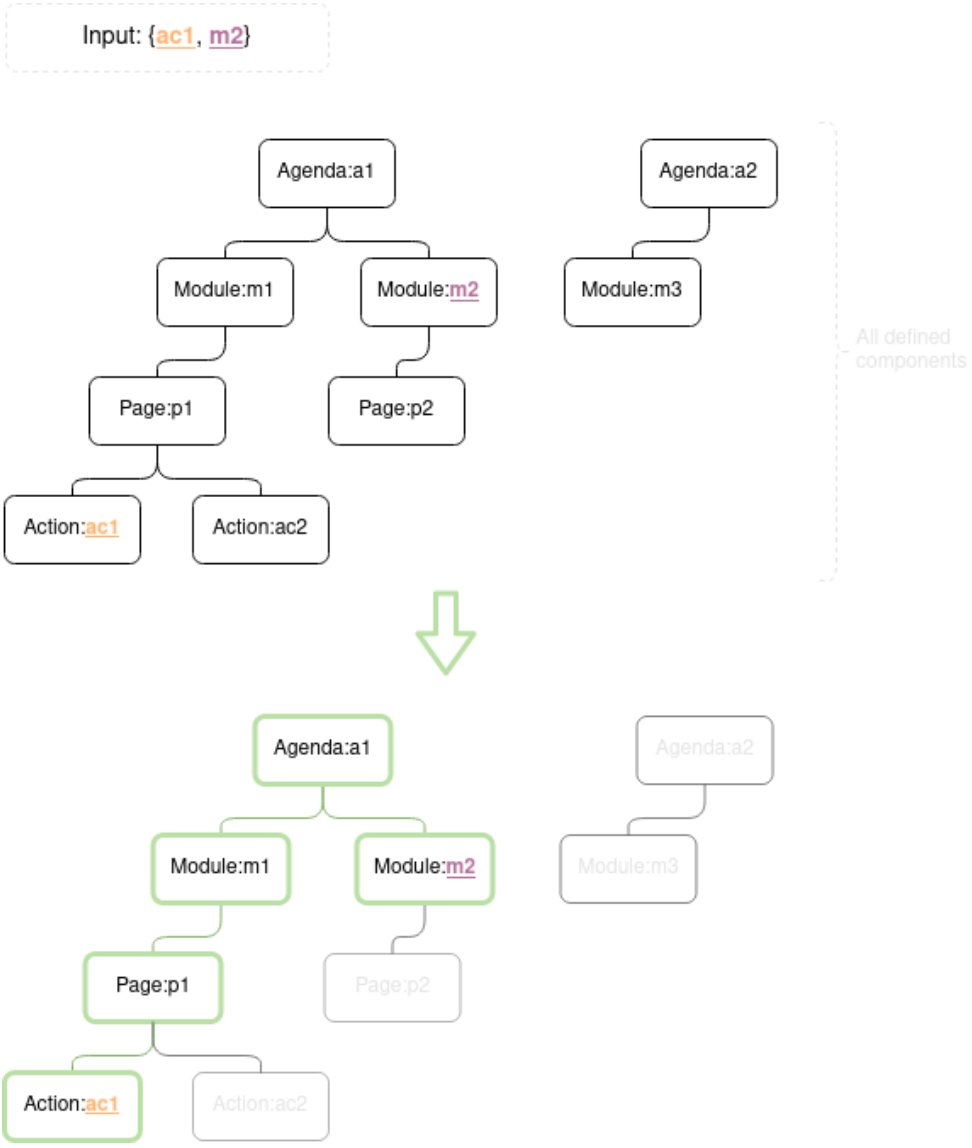
7.5 Whoami algoritmus

Algoritmus pro vytváření Whoami odpovědi přijímá jako vstup množinu identifikátorů a vrací stromy autorizovaných komponent. Autorizovaná komponenta je taková, pro kterou platí:

1. alespoň jedno z následujících tvrzení:
   a. Identifikátor komponenty je obsažen ve vstupní množině identifikátorů.
   b. Potomek komponenty je autorizován.

2. A zároveň
   a. komponenta či žádná z jejích předků nejsou deaktivovány.

Jednodušeji řečeno, pokud je komponenta na vstupu, autorizují se i všichni její předci (s ohledem na stav atributu *isDisabled*). Algoritmus ilustruje diagram 7.3.



Obrázek 7.3: Ilustrace Whoami algoritmu (zelené komponenty jsou autorizované)

Algoritmus implementuje třída *WhoamiResolver*, jejíž instance se vytváří se vstupními identifikátory. Zavoláním metody *getAgendas* se algoritmem vytvoří množina povolených komponent. Metoda poté vrátí seznam data transfer objektů *WhoamiAgenda*, které obsahují pouze agendy a potomky

přítomné v množině povolených komponent. V podstatě se jedná o jakýsi sofistikovanější mapper databázových entit na DTO.

7.5.1 Získání vstupních identifikátorů

Pro použití algoritmu je potřeba získat povolené identifikátory daného uživatele. Zdrojem jsou autorizační pravidla uložených komponent a integrované služby. Proces získávání vypadá následovně:

1. Načtení komponent, které v atributu *allowedUsernames* obsahují uživatelské jméno uživatele.
2. Načtení komponent, pro které existuje průnik mezi rolemi v atributu *usermapRoles* a rolemi uživatele, získané z UsermapAPI service.
3. Načtení seznamu identifikátorů z integrovaných služeb. Dotazování na `/whoami` endpoint probíhá pouze pro služby, jejichž jméno či adresa je obsažena v některé z Whoami definic.

Adresa dotazované služby je buď uživatelem definovaná URL, nebo jméno služby, na základě kterého se adresa získá prostřednictvím instance *DiscoveryClient*. Služba se nejprve pokouší přečíst data v novém formátu z endpointu `/whoami/<username>`, tedy v podobě pole identifikátorů. V případě neúspěchu se pokusí načíst data ve starém formátu.

V popisu GraphQL rozhraní uvádíme, že nová Whoami služba používá identifikátory k autorizaci některých vlastních akcí. V takovém případě se instance resolveru vytváří pouze s identifikátory uložených komponent (krok 1 a 2), předchází se tak nadbytečným síťovým dotazům mezi službami.

7.6 Testování služby

Mezi platformními službami dosud není stanovená jednotná koncepce přístupu ke psaní testů. Návrhem testovací strategie nové Whoami service tedy cílíme i na zavedení testovacích praktik pro ostatní služby. Při psaní testů prioritizujeme tyto vlastnosti:

1. Testování business logiky na úkor implementačních detailů.
2. Testování je „přirozená“ součást vývoje a nevnáší nadbytečnou práci.

Bodem č. 1 obhajujeme absenci jakýchkoli *unit* testů, t.j. granulárních testů interních funkcionalit jako jsou metody apod. Pro racionalizaci uveďme příklad:

Metoda *getAgendas* třídy *WhoamiResolver* původně navracela modifikované objekty databázových entit, na rozdíl od aktuálního stavu, kdy vrací DTO. Pokud bychom pokryli metodu unit testy, museli bychom kromě přepisování logiky zapracovat i změny do všech těchto testů. To znamená nadbytečnou práci, a to i přesto, že se refaktorováním z původního stavu nezměnila jediná funkcionalita z pohledu uživatele služby. Původní testy navíc neposlouží k ověření správnosti refaktorovaného řešení - vývojář je může použít až poté, co je upravil, což přináší riziko zanesení regresí do samotných testů. Testy tudíž po refaktorování nepřináší z podstaty věci jistotu, že klíčová logika funguje korektně.

Z těchto důvodů upřednostňujeme testy API celé služby, neboli *integrační* testy. Při uplatnění integračních testů bychom v předešlém případě nejen že nemuseli měnit stávající testy, dokonce by nám i posloužily k ověření správnosti refaktorované metody. Toto sice platí pouze za předpokladu, že byly kvalitně napsány, to je ovšem společný požadavek jak na integrační, tak i unit testy.

Bodem č. 2 zdůrazňujeme požadavek na testování, které není přítěží a co nejméně ovlivňuje vývojářskou workflow. Tento požadavek vychází z omezeného časového fondu vývojářů platformy (zpravidla se jedná o studenty), kteří musí zároveň spravovat několik platformních služeb. Testy proto vytváříme pomocí API platformy Postman [49], kterou vývojáři služeb běžně používají. Nástroj umožňuje poměrně jednoduchý způsob psaní testovacích skriptů v jazyce JavaScript. Skripty navíc nevyžadují žádné pokročilé znalosti jazyka, protože se zpravidla skládají pouze ze získání dat z HTTP odpovědi a ověření jejich hodnot. Pro ilustraci přikládáme test mutace *createWhoamiServiceUser*:

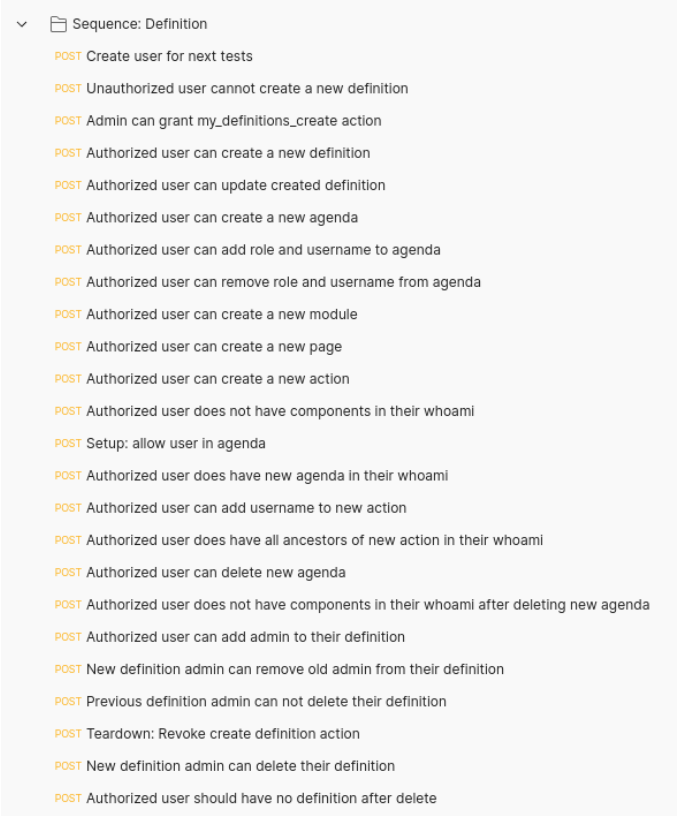
```
pm.test("User is created with correct data", () => {
    const data = utils.getData(pm.response)
    const variables = utils.getVariables(pm.request)
    const user = data.createWhoamiServiceUser.user;

    pm.expect(user.username).to.eq(variables.username);
    pm.expect(user.isAllowedInTest).to.eq(variables.attributes.
    isAllowedInTest);
    pm.expect(data.createWhoamiServiceUser.error).to.eq(null);
});
```

Tímto se klasický vývojářský proces ověřování API funkcí rozšíří pouze o doplnění jednoduché validační logiky, která se automaticky provede při každém dotazu. Skripty mohou navíc sdílet globální proměnné, takže je možné provádět sekvence operací, které přepoužívají své vstupy a výstupy. Tím se dají pokrýt komplexnější use-casy, ale zároveň i zautomatizovat sekvence volání API při vývoji (což může vývojáře přímo motivovat ke psaní testů).

7.6.1 Integrační testy

Postman kolekce testů služby obsahuje cca 120 testů pro 60 dotazů. Dotazy jsou koncepčně rozděleny do složek. Každá složka představuje sekvenci dotazů, která by měla být po vykonání všech dotazů znovu spustitelná a nesmí ovlivnit běh ostatních složek.



Obrázek 7.4: Složka s dotazy sekvence pro testování Whoami definic

Celá kolekce testů je součástí repozitáře projektu ve formátu JSON. Tímto způsobem může testy zprovoznit a udržovat jakýkoli vývojář projektu. Testy se spouští v CI pipeline (podrobněji viz níže) při každém nahrání změn do repozitáře.

Mockování dat

Součástí logiky Whoami service je získávání dat z ostatních služeb infrastruktury. Abychom mohli „deterministicky“ testovat interakci s ostatními službami, bylo potřeba vytvořit mockovací služby, které budou poskytovat očekávaný výstup. Je sice možné se připojit na Usermap service a služby, které poskytují `/whoami` endpoint, nemáme však zaručeno, že budou v době testování v provozu a že vrátí data, která požadujeme. Testy by v tomto případě

mohly být tzv. „flaky“, t.j. jejich výstup by se mohl měnit bez ohledu na stav testovaného systému. Takové testy nemusí přinášet důvěryhodné výsledky.

Platforma Postman umožňuje uživateli vytvářet vlastní instance serverů, které poskytují definovaná data [50], nicméně stejně jako u připojení k platformním službám musíme spoléhat na to, že mock servery budou během testování v provozu. Pro vytváření mockovacích služeb jsme proto zvolili implementaci vlastního serveru v Node.js [51], který je možné spouštět dle potřeby. Node.js je serverový JavaScript runtime, který nám umožňuje napsat jednoduchý server jedním souborem (výsledný skript má cca. 40 řádků). Server poskytuje (na základě vstupního argumentu) data v novém i starém Whoami formátu, spolu s formátem specifickým pro službu Témat, která obsahuje chybu v názvu atributu.

7.6.2 Spouštění testů v CI

Aby testy správně proběhly, musí obsah dotazů odpovídat stavu testovaného systému. Pokud např. testujeme dotazy v roli administrátora, musíme mít k dispozici existující uživatelské jméno administrátora v aplikaci. Pro tyto účely vznikl shell skript, který na základě vstupních argumentů

1. nastaví hodnotu proměnných testovací kolekce,
2. spustí instance mockovacích serverů
3. vytvoří konfigurační `.properties` soubor aplikace,
4. spustí aplikaci,
5. spustí testy.

Testy se spouští nástrojem Newman [52] pro spouštění testovacích kolekcí z terminálu. Proměnné kolekce se načítají z JSON souboru s polem objektů, kde každý objekt představuje iteraci testů. Pole obsahuje tři objekty, které se liší pouze v proměnné s adresou mock serveru, skript totiž pouští tři servery s jiným formátem Whoami odpovědi, aby se tak otestovala i zpětná kompatibilita.

Tento skript především zjednodušuje testování a zprovoznění testů v rámci CI pipeline. Abychom mohli spouštění aplikace a testů zprovoznit v jednom skriptu, je potřeba aby aplikace běžela na pozadí, zatímco probíhá testování. Pro tyto účely poskytuje Spring Boot příkaz `spring-boot:start`, který spustí aplikaci a pošle její proces do pozadí. Proces na pozadí lze pak ukončit příkazem `spring-boot:stop`.

```
1 application test:
2   stage: Test
```

```
  allow_failure: false
  when: always
  tags:
    - test
  variables:
    # Postgresql service config
    POSTGRES_DB: $POSTGRES_DB
    POSTGRES_USER: $POSTGRES_USER
    POSTGRES_PASSWORD: $POSTGRES_PASSWORD
    POSTGRES_HOST_AUTH_METHOD: trust
  services:
    - postgres:12.2-alpine
  before_script:
    - apt-get update -y
    # Install node for newman
    - curl -sL https://deb.nodesource.com/setup_18.x | bash
    - apt-get install nodejs -yq
  script:
    - bash integration-tests/run-tests.sh adminuser 8080 8086
     $POSTGRES_DB $POSTGRES_USER $POSTGRES_PASSWORD postgres
```

V testovacím jobu pak stačí pouze poskytnout skriptu údaje k databázi a nainstalovat Node.js pro spouštění Newman aplikace. Testovací databáze je vytvořena pomocí samostatného Docker image prostřednictvím atributu *services* [53].

Celá git workflow služby spolu s CI nastavením se řídí standardy release managementu. Pro zjednodušení dodržování těchto standardů jsme vytvořili šablonu CI, která obsahuje joby automatizující předepsané git operace. Této šabloně se konkrétně věnujeme v následující kapitole.

Kapitola 8

CI template pro automatizaci release managementu

Zásady release managementu platformy [54] jsou podrobně definovány v interních wiki stránkách. Jejich dodržování je klíčové pro správu služeb a funkcionalitu závislou na jejich propojení. V současném stavu je ovšem jejich dodržení plně v rukou vývojářů, tudíž je nutné, aby si dokument důkladně přečetli, pochopili a soustavně se zásadami řídili. I přes pořádání dedikovaných workshopů a připomínání se však nedaří standard plošně vynutit (ne všechny týmy se jím řídí) a na projektech, které jej dodržují, vznikají nekonzistence či hrubá porušení, jako např. artefakty bez označení verzí. Pro zamezení těchto problémů jsme se pokusili automatizovat Git operace definované prostřednictvím CI šablony, která vznikla v rámci projektu nové Whoami služby a následně byl podán návrh na její začlenění do dalších konfiguračních souborů platformy.

Základem celého principu je verze, pro kterou se vytváří release. Abychom mohli implementovat CI funkcionality pro automatizaci release managementu, musíme definovat primární zdroj informace o dané verzi, který bude strojově zpracovatelný. Pro tento účel jsme definovali konfigurační soubor s názvem *.hub-info.yml*, který musí obsahovat atribut *version*, t.j. aktuální verzi ve formátu sémantického verzování a *release-candidate*, pořadí release kandidáta aktuální verze. Tento soubor musí být umístěn v kořenové složce projektu. Konfigurace se skládá z těchto jobů:

upsert release branch (manuální) Vytvoří novou release větev se jménem ve formátu `release-<major>-<minor>`, kde hodnoty *major* a *minor* odpovídají major a minor číslu verze v souboru *.hub-info.yml*. Do této větve pak merguje hlavní větev. Pokud větev s tímto jménem již existuje, pouze se merguje hlavní větev. Tento job se spouští pouze pro pipeline hlavní větve.

create rc tag (manuální) Tento job se spouští pouze pro pipeline release

větví. Na základě atributů info souboru vytvoří tzv. „release candidate tag“ ve formátu `<version>-rc<release-candidate>`.

create final tag (manuální) Tento job se spouští pouze pro pipeline tagů ve formátu release kandidátů a vytváří tzv. „final tag“ s názvem aktuální verze.

create release image Pro pipeline finálních či release candidate tagů automaticky vytvoří docker image s příslušným tagem. Tento image se poté nahraje do container registry daného projektu (z registry projektu se pak image stahují pro deployment).

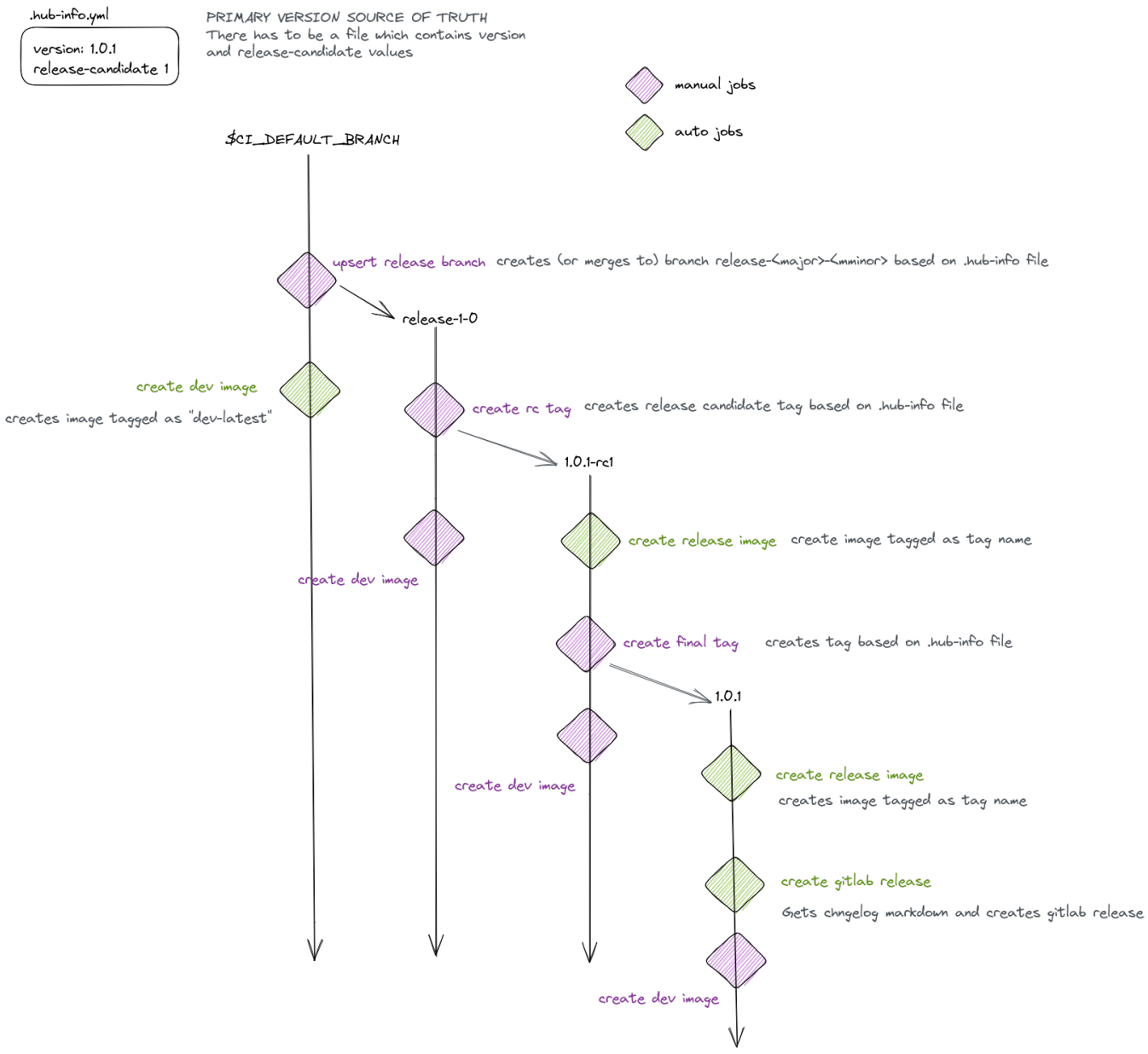
create dev image (manuální) Pro jakoukoli pipeline vytvoří image s tagem *dev-latest*. Image s tímto tagem se nasazují na *dev* prostředí.

create gitlab release Do šablony byl přidán také job pro vytváření changelog dokumentace na základě commit zpráv a záznamu pro GitLab release, který obsahuje vygenerovaný changelog a komprimované soubory projektu. Tento job se spouští po úspěšném jobu *create release image*

Sekvenci jobů ilustruje diagram 8.1:

Pro použití template je potřeba v `.gitlab-ci.yml` souboru projektu přidat jeho URL do atributu *includes* a rozšířit *stages* atribut o hodnoty *Image*, *Release*, *DevImage* a *Deploy*. Skripty používají GitLab API a klonování projektu přes HTTPS, ke kterým je potřeba v GitLab projektu nastavit proměnnou *CI_API_TOKEN* na hodnotu projektového tokenu s rolí *Developer* a scopem *api*.

Poté by měla z pohledu metodiky release managementu být díky této konfiguraci po vývojářích vyžadována pouze korektní definice verze v info souboru. Zbylé aktivity (od vytvoření release větve až po nahrání korektně otagovaného artefaktu) stačí pouze „odklikat“ v GitLab UI dle potřeby.



Obrázek 8.1: Ilustrace jobů CI template pro automatizaci release managementu

Kapitola 9

UI pro administraci Whoami

Frontendová část nástroje pro správu Whoami se skládá z nové administrační agendy a z upravení frontendového zpracování nového Whoami formátu.

Stávající implementace UI používá Whoami uživatele pouze pro „maskování“ sekcí aplikace. To znamená, že z existujících UI komponent povolí zobrazení pouze těch, ke kterým má uživatel přístup. Takové použití souhlasí se starým formátem Whoami, který o komponentách neposkytoval téměř žádná metadata. Definice struktury UI agend tedy vypadá následovně:

```
{
  whoamiManagement: {
    icon: <UsersIcon />,
    renderTitle: () => t`Whoami management`,
    pages: {
      myDefinitions: {
        id: 'myDefinitions',
        renderTitle: () => t`My Definitions`,
        component: <DefinitionsPage type="my" />,
        hasInnerRoutes: true,
      },
      allDefinitions: {
        id: 'allDefinitions',
        renderTitle: () => t`All Definitions`,
        component: <DefinitionsPage type="all" />,
        hasInnerRoutes: true,
      },
      whoamiServiceUsers: {
        id: 'whoamiServiceUsers',
        renderTitle: () => t`Whoami service user management`,
        component: <DefinitionsPage type="all" />,
        hasInnerRoutes: true,
      },
    },
  },
};
```

Tento konfigurační objekt se předává funkci *AppEntryPoint*, která podle něj vytvoří strom komponent s routováním podle URL pomocí knihovny

React Router [55]. V konfiguračním objektu lze pozorovat duplikaci struktury Whoami.

Díky novému formátu lze konfiguraci v UI značně zjednodušit, ten totiž obsahuje data, která musela být doposud definovaná na straně frontendu, jako jsou názvy komponent či ikony modulů v navigační liště. Pro implementaci UI je tedy teoreticky potřebné pouze přiřadit UI komponentu k Whoami komponentě.

Pro tento účel vznikla funkce *GenerateAppEntryPoint*, které stačí pouze mapování identifikátorů na React komponenty (tzv. *implmentory*). Tím se předchozí příklad zjednoduší na:

```
{
  myDefinitions: {
    render: () => <DefinitionsPage type="my" />,
  },
  allDefinitions: {
    render: () => <DefinitionsPage type="all" />,
  },
  whoamiServiceUsers: {
    render: () => <ServiceUsersPage />,
  },
};
```

Lze si všimnout, že nová konfigurace již neobsahuje stromovou strukturu. Kromě snížené duplikace je výhodou také to, že již nezáleží na typu Whoami komponenty, ani na jejím předkovi. Pokud by se v tomto případě změnila definice tak, že stránka *allDefinitions* patří pod jiný modul, navigační lišta spolu s routováním budou odpovídat nové definici bez nutnosti úpravy frontendové konfigurace. Abychom zamezili nekonzistentnímu UI, které by mohlo vzniknout v aplikacích s původním *AppEntryPoint*, byla tato funkce rozšířena tak, aby preferovala názvy komponent z Whoami odpovědi na úkor vstupního konfiguračního objektu.

GenerateAppEntryPoint vygeneruje layout aplikace spolu s rozcestníkem v levém navigačním menu stránky s použitím Whoami dat. Dále pak vloží do UI stromu rekurzivní funkci *WhoamiComponentRoute*, která z URL přečte identifikátor „své“ Whoami komponenty a vykreslí následující obsah:

1. Pokud existuje pro identifikátor aktuální komponenty implementor, vykreslí implementor.
2. *Switch* komponentu, do které vloží
   a. *WhoamiComponentRoute* pro následující url segment, pokud má Whoami komponenta nějakého potomka,
   b. *Redirect* na následující URL segment, pokud existuje potomek a neexistuje implementor.

3. Pokud neexistuje implementor ani potomek Whoami komponenty, vykreslí stránku se zprávou o nenalezeném obsahu.

Metoda implementoru *render* přijímá jako argument příslušnou Whoami komponentu a React funkci *children*. Implementor tedy může sloužit i jako „obal“ pro UI komponent jeho potomků:

```
{
  whoamiManagement: {
    render: (_, children) => <Layout>{children}</Layout>
  },
  myDefinitions: {
    render: () => <DefinitionsPage type="my" />,
  },
  ...
};
```

V tomto příkladu budou všechny komponenty stránek modulu *whoamiManagement* vyrenderovány uvnitř komponenty *Layout*, která může mít např. specifický vzhled, či poskytovat kontext svým potomkům.

Funkce *GenerateAppEntryPoint* dále přijímá identifikátor agendy, pro kterou generuje obsah. Teoreticky by bylo možné jednoduše implementovat i automatické vygenerování všech agend na základě Whoami odpovědi (implementory se nastaví v globálně přístupném objektu a seznam entrypointů se vyrenderuje podle agend uživatele), tento krok by ovšem vyžadoval přepsání všech existujících agend, což spadá mimo realizovatelný rozsah této práce. Zvolili jsme proto přístup pozvolné migrace, kde mohou vývojáři frontendu postupně přepisovat existující agendy dle potřeby. Výhodou tohoto přístupu je navíc i to, že je možné některé Whoami komponenty použít jako ovladače přístupu i pro jiné UI komponenty, než jsou agendy, moduly a stránky. Nevýhodou je naopak to, že při přidání Whoami agendy se změna ihned neprojeví v UI aplikace.

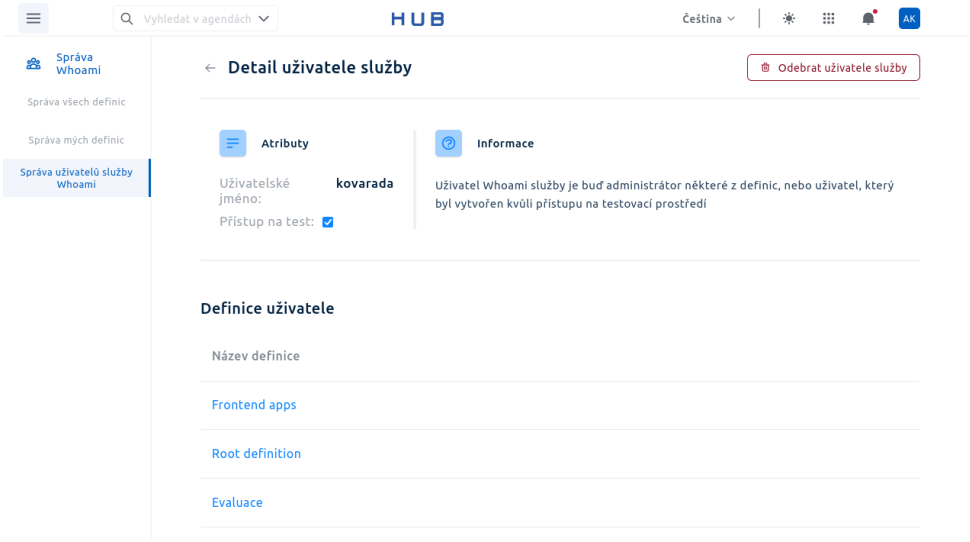
9.1 Administrační agenda

UI správy Whoami se drží stejného konceptu jako ostatní aplikace v rámci platformy. Nástroj je koncipován jako modul *Správa Whoami* se třemi stránkami, struktura a názvy jsou vidět v konfiguračním objektu výše. Modul je zasazen do nové agendy s názvem *Hub admninistrace*, která bude sloužit i dalším funkcionalitám pro správu platformy.

9.1.1 Správa uživatelů služby Whoami

Služba Whoami ve své databázi uchovává záznamy pro uživatele, kteří jsou buď administrátorem některé z definic, nebo byli přidán kvůli přístupu k

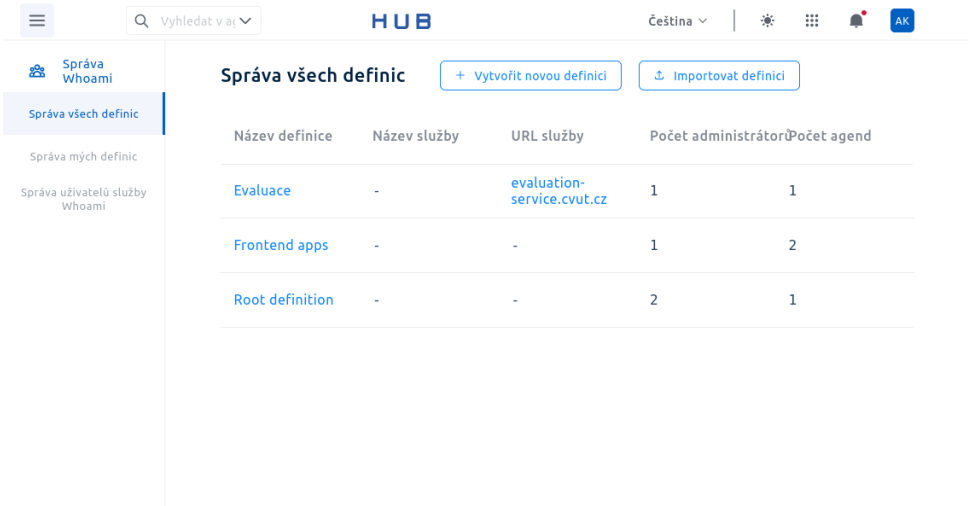
testovacímu prostředí. Pro správu těchto záznamů byla vytvořena jednoduchá stránka s tabulkou uživatelů a stránka pro detail uživatele s příslušnými funkcionalitami.



Obrázek 9.1: Detail uživatele služby Whoami

9.1.2 Správa definic

Stránky „Správa mých definic“ a „Správa všech definic“ obsahují tabulky s odkazy na detail definice a tlačítka na vytváření nových definic (pouze má-li uživatel přístup k akci *myDefinitions_create*).

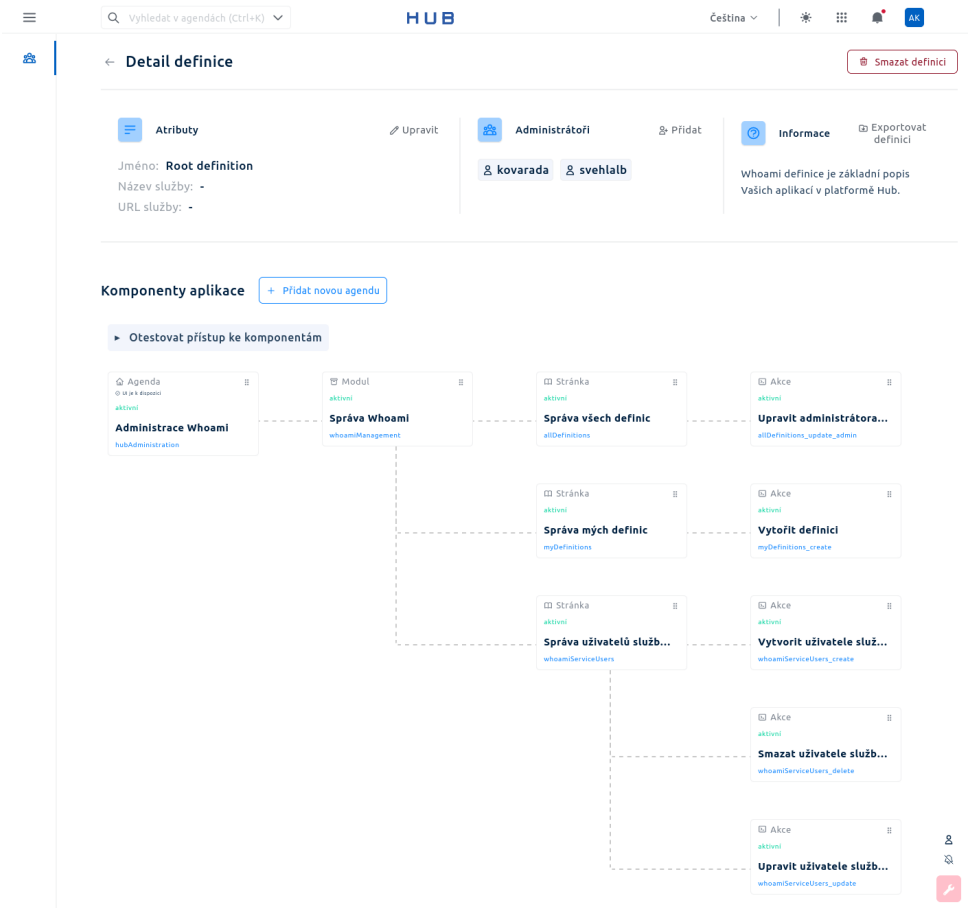


Obrázek 9.2: Screenshot stránky „Správa všech definic“

Odkaz či vytvoření definice přesměruje uživatele na stránku „Detail definice“, která obsahuje veškeré funkcionality k úpravě definic a Whoami

komponent. Tyto funkce jsou podrobněji popsány v rámci GraphQL dotazů výše a dále se proto zabýváme implementacemi specifickými pro UI.

Vytvořením databázového úložiště definic jsme se připravili o automatickou replikaci Whoami definic mezi prostředími. Jako náhrada byla vytvořena funkcionalita pro export/import definic. Definice lze exportovat prostřednictvím kopírování ve formátu JSON do schránky a vložení do formuláře pro import definice. Logika formuláře se musí pouze postarat o převedení vstupních dat na typ *WhoamiModuleComponentCopyInput*.



Obrázek 9.3: Stránka s detailem *root* definice

Pro zobrazení Whoami komponent byla zvolena stromová hierarchie. Vzhledem k jejich předvídatelné struktuře (jedná se o strom s výškou menší nebo rovnou 4) jsou UI komponenty rozmístěny jednoduchým způsobem pomocí mřížky a funkcionality CSS grid. Mřížku implementují funkce *Grid* a *Cell* podle následujících pravidel:

1. Mřížka má 4 sloupce (maximální výška stromu) a počet řádků se rovná počtu listů stromu.
2. Pro pozici elementu v mřížce (x, y), kde x odpovídá sloupci a y řádku

mřížky platí:

a. $x = x_p + 1$, kde x_p je sloupec rodiče a

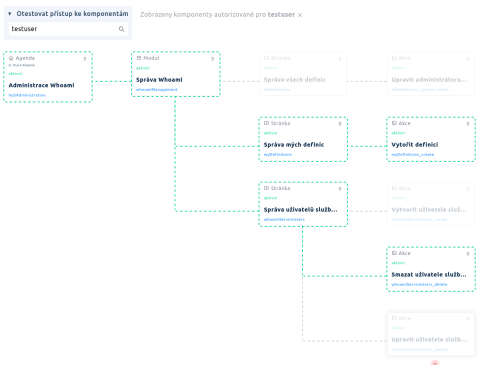
b. $y = y_p + l$, kde y_p je řádek rodiče a l je součet listů předcházejících sousedů elementu.

Buňky *Cell* renderuje rekurzivní funkce *ComponentTree*, která na základě vstupní pozice a obsahu vykreslí „svou“ komponentu a její potomky s pozicemi dle definovaných pravidel. Kořenem stromu je *ComponentTree*, kde je argumentem komponenta agendy a pozice $(1, 1)$.

Díky mřížce je dále poměrně jednoduché zobrazit linky znázorňující spojení předka s potomkem. Ty jsou totiž vykreslené jako levý a dolní rámeček průhledných HTML elementů, které se v mřížce rozprostírají od pozice předka (levý horní roh elementu) až k pozici potomka (pravý dolní roh), viz 9.4. Elementy také vkládá do mřížky funkce *ComponentTree*.

Obrázek 9.4: Zvýrazněný implementační detail UI linky stromu komponent

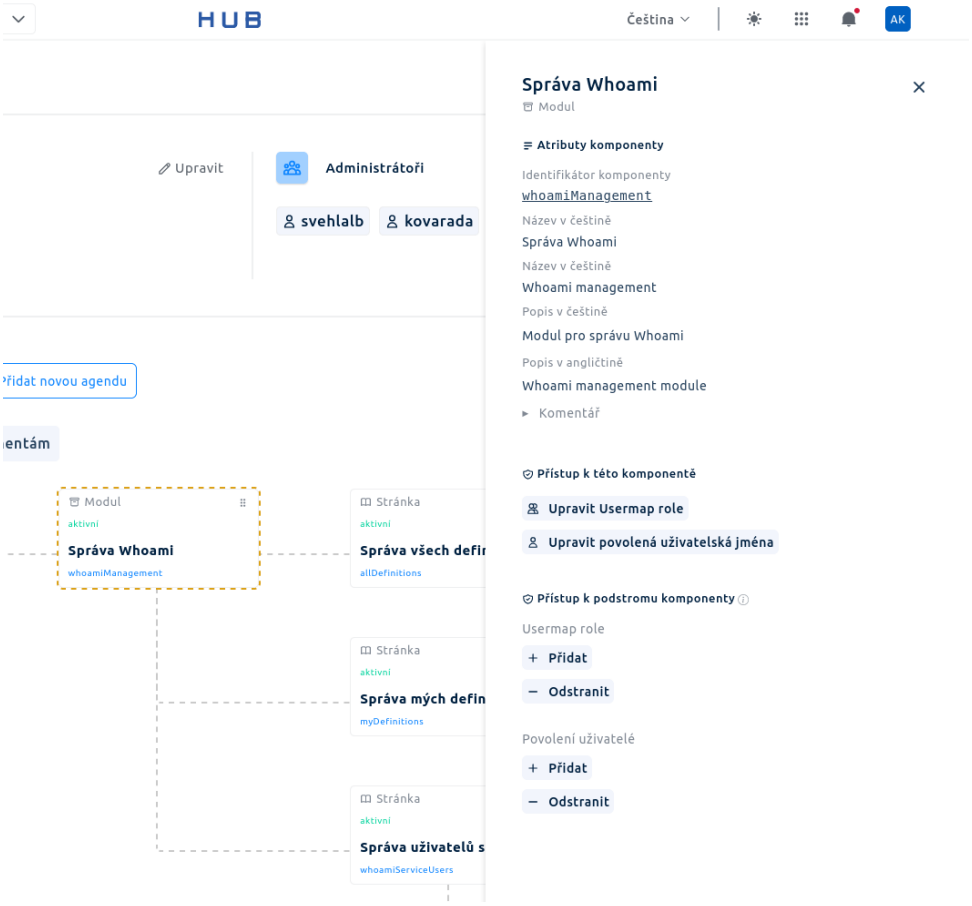
Pro funkcionalitu testování přístupu uživatelů ke komponentám se do funkce *ComponentTree* vkládá argument *userWhoamiFilter*, který obsahuje Whoami komponenty přístupné testovanému uživateli. Pokud je vstupní Whoami komponenta funkce obsažena v seznamu *userWhoamiFilter*, předá tuto informaci vykreslené spojovací lince a kartičce s detailem Whoami komponenty, které se zobrazí se zeleným ohraničením. V opačném případě se vykreslí s mírnou průhledností, viz 9.5.



Obrázek 9.5: Zobrazení autorizovaných komponent

Přes karty Whoami komponent lze komponenty mazat, přidávat potomky, upravovat jejich aktivitu či je přesouvat k jiným předkům pomocí tažení myší. Další atributy spolu s přístupy je možné upravit v detailu komponent (obr. 9.7)

Ucelený souhrn atributů definice a všech jejích komponent si lze zobrazit i jako dokument vygenerovaný v markdown formátu.

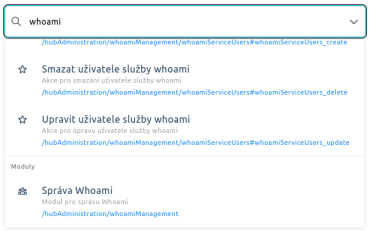


Obrázek 9.6: Detail Whoami komponenty

9.1.3 UI vyhledávání platformy

V současné verzi neexistuje backend pro funkcionalitu vyhledávání v rámci platformy a vyhledávací pole nabízelo pouze záznamy, které měl kód UI k dispozici. Díky rozšířenému formátu lze však uživateli poskytnout vyhledávání přes Whoami komponenty, které nyní obsahují jak název, tak i popis. Položky vyhledávání vyžadují relativní URL odkaz na daný obsah, komponenty se jednoduše rozšíří o atribut URL spojením jejich identifikátorů a identifikátorů předků.

Protože se komponenty typu akce nemusí mapovat na konkrétní UI elementy,



Obrázek 9.7: Vyhledávání s použitím obsahu Whoami komponent

je možné tyto komponenty použít k podrobnějšímu popisu stránky, které patří. Díky tomu získává integrující uživatel možnost libovolně rozšířit obsah vyhledávání dle potřeby.

9.1.4 Error handling

Nové GraphQL rozhraní pro chybové hlášky popsané v kapitole o nové Whoami službě je (a pravděpodobně bude) hojně používané, čímž vzniká v UI kódu opakovaný pattern. Pro usnadnění práce s chybovou odpovědí vznikla funkce *handleUserErrorResponse* s typovým předpisem

```
function handleUserErrorResponse(
  error: BaseError,
  handlers: {
    onUnauthorized?(error: UnauthorizedError): void;
    onUserInputError?(error: UserInputError): void;
    handleErrorField?(field: UserInputError['fields'][number]):
    void;
  },
): void
```

Argument *handlers* přiímá funkce, které se volají pro příslušný typ vstupního erroru. V případě erroru typu *UserInputError* se funkce *handleErrorField* zavolá pro každý objekt z pole atributu *fields* error objektu.

Kapitola 10

Testování implementace

Jako metodu otestování nástroje pro správu Whoami definic jsme zvolili připojení integrovaných služeb. Jelikož během testování nebyla dokončena migrace platformy na GraphQL federaci, nebylo možné nasadit naše řešení na *development* prostředí. Whoami služba a frontend byly zprovozněny lokálně a k nasazeným integrovaným službám se připojovalo pomocí jejich veřejného (přes VPN) URL.

Pro každou službu poskytující `/whoami` endpoint byla v nástroji vytvořena nová definice, která odpovídá stávajícím komponentám. Pouze tento krok umožňuje poskytnutí stejné funkcionality systému jako doposud a tím v podstatě **zajišťuje migraci na nový zdroj Whoami definic** (původní Whoami služba je již v tomto případě nepotřebná). Pro integrované systémy byla zvolena následující strategie připojení:

Eprocesy Whoami definice obsahuje veškerá autorizační pravidla. Whoami služba se tedy nedotazuje na `/whoami` endpoint služeb Eprocesů a teoreticky je možné tuto implementaci ze služeb kompletně smazat.

Evaluace Autorizační pravidla Evaluací nelze podchytit pouze na základě rolí a testování proběhlo přidáním `/whoami/<username>` endpointu s novým formátem odpovědi.

Témata Systém témat doposud nemá konkrétně definovaná pravidla autorizace Whoami komponent. Vytvořili jsme tedy pouze základní definici komponent bez autorizačních pravidel a úpravy endpointu.

Dále byla přidána definice pro nástroje, které jsou určené vývojářům platformy a jsou umístěny v agendě *sandbox*. Jedná se především o nástroj GraphiQL [56], který je integrován pomocí knihovny do UI platformy a slouží ke snadnému testování GraphQL rozhraní Endpoint service. Tento nástroj dosud není v testovacím ani produkčním prostředí k dispozici (pro jeho funkční přidání do Whoami by bylo potřeba upravit několik služeb). Díky novému

granulárnímu udělování přístupu přes uživatelská jména ovšem není problém jej poskytnout pouze některým uživatelům bez nutnosti nasazení žádné nové verze služeb infrastruktury.

10.1 Základní integrace systémů

Pro vytvoření základních definic - neobsahují autorizační pravidla, pouze předpis komponent a adresy služeb - jsme použili záznamy z wiki platformy. Po vytvoření komponent a připojení služeb byly otestovány přístupy specifických uživatelů prostřednictvím impersonace.

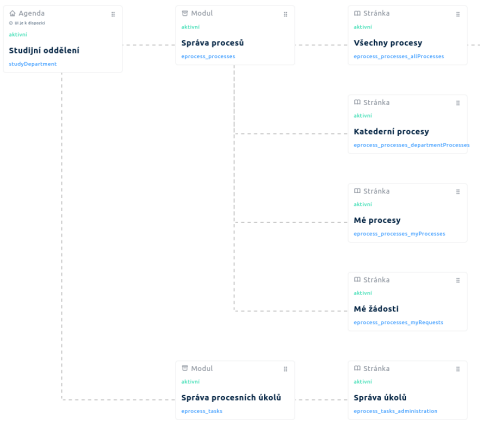
U Témat se nepodařilo připojit k Whoami endpointu kvůli chybě na straně služby Témat (autorizace v tu dobu nefungovala ani s původní Whoami službou) a výsledky neúspěšného připojení tohoto systému tedy opomíjíme. U systémů Evaluací a Eprocesů bylo prostřednictvím impersonace za uživatele s různými přístupy ověřeno, že autorizace zůstala korektně zachována. Díky tomu můžeme prohlásit, **že jsme úspěšně vytvořili nový, primární zdroj Whoami definic**, který nahrazuje původní službu Whoami. Kromě tohoto zdroje má uživatel integrující systém nyní k dispozici

- přidávání specifických přístupů podle uživatelského jména,
- přidávání přístupů podle rolí,
- de/aktivování komponent,
- změnu názvů komponent s jejich popisy,
- rozšíření vyhledávání pomocí „skrytých“ komponent

a to vše bez jediného zásahu do implementace. Definice ovšem v tomto stavu zatím neobsahují autorizační pravidla a systémy neposkytují nový `/whoami/<username>` endpoint, takže v tomto stavu nelze plnohodnotně testovat přístup uživatelů prostřednictvím editoru komponent.

10.2 Integrace Eprocesů

Pro vytvoření autorizačních pravidel komponent jsme vycházeli z analytického podkladu projektu, který obsahuje tabulku s přiřazením rolí pro přístup k datovým položkám agendy. Tento popis ovšem neobsahuje informace o Whoami modulech a stránkách, datové položky bylo proto nutné nejprve přiřadit k identifikátorům podle specifikace ve wiki platformy.



Obrázek 10.1: Definice Whoami komponent pro systém Eprocesy

Zde si opět můžeme všimnout, že informace o Whoami komponentách se nejen duplikují mezi analytickým zadáním, backendovou implementací, dokumentací platformy a frontendovou implementací, ale dokonce ani neexistuje ucelený popis, který by obsahoval veškeré informace pro implementaci Whoami systému - analýza obsahuje popis business rolí, wiki stránky obsahují identifikátory s popisy komponent a backendový kód obsahuje validační logiku.

Až po kompilaci těchto zdrojů vzniklo přidělení Usermap rolí k identifikátorům. Zde se ukázalo, **že vyjádření některých přístupů autorizace pomocí výčtu Usermap rolí je velice pracné, a náchylné na chyby**. Například pro identifikátor *eprocess_processes_departmentProcesses* má dle analýzy přístup „Referent“, „Vedoucí katedry, zástupce“ a „Proděkan“. Jako referenta může osobu označovat několik možných rolí (dle její agendy), neexistuje však žádná role obecně označující tuto funkci. Podobně je tomu tak u ostatních jmenovaných případů, role pro vedoucí katedry musí obsahovat informaci, k jaké katedře se funkce váže. V tomto případě by uživatel musel zadat vedoucí role pro *všechny* katedry zvlášť.

Tento nedostatek jsme se rozhodli vyřešit upravením autorizační logiky rolí tak, že seznam rolí může obsahovat řetězce ve formátu regulárních výrazů. Po aplikaci úprav se předpis Usermap rolí pro komponentu *eprocess_processes_departmentProcesses* zkrátil na

- `B-d{5}-VEDOUCI`,
- `B-d{5}-PRODEKAN(.*)`,
- `(.*)-REFERENT-(.*)`,
- `B-d{5}-VEDOUCI-ZASTUPCE`.

Až po zavedení této změny bylo možné vytvořit autorizační schéma pro všechny komponenty Eprocesů. Následně proběhlo úspěšné ověření správnosti autorizace (opět prostřednictvím impersonace) bez připojení na službu systému, čímž jsme tým Eprocesů *teoreticky* zbavili potřeby implementace endpointu `/whoami`. Zdůrazňujeme, že pouze teoreticky. Z konzultace řešení s vedoucím projektu totiž vyplynulo, že **zvažují nadále udržovat autorizační logiku v rámci své služby**. Je tak z toho důvodu, že jejich rozhraní autorizují stejným způsobem jako Whoami odpovědi (např. metoda pro zjištění zdali je osoba student se použije jak pro ověření stránky „Mé žádosti“, tak pro endpoint, ze kterého se obsah stránky získává). Používání autorizace ve Whoami službě by proto přineslo duplikaci logiky, což by mohlo být náročné na údržbu a zároveň náchylné na nekonzistence.

10.3 Integrace Evaluací

Autorizační logika služeb systému Evaluace je spjatá s jejich interní databází a nelze ji vyjádřit prostřednictvím Usermap rolí. Whoami služba již díky základní definici komponent Evaluací poskytuje korektní data, ale není možné testovat přístupy uživatelů v editoru komponent. Pro otestování integrace jsme se proto rozhodli rozšířit stávající kontroler endpointu `/whoami` o endpoint odpovídající novému formátu:

```
/whoami/{username}:
  get:
    parameters:
      - name: username
        in: path
        required: true
        schema:
          type: string
    tags:
      - whoami
    summary: Returns user's whoami permissions in new whoami format
    operationId: getWhoamiByUser
    responses:
      "200":
        description: User permissions
        content:
          application/json:
            schema:
              type: array
              items:
                type: string
```

Pro rozšíření endpointu ve službě Evaluation Period bylo potřeba upravit OpenApi specifikaci služby, ze které byl následně vygenerován předpis metody poskytující novou logiku. Metoda byla pro účely testování implementována jednoduchým převedením existující odpovědi na seznam identifikátorů:

```java
@Override
public ResponseEntity<List<String>> getWhoamiByUser(String
    username) {
    var oldResponse = whoAmIService.createWhoAmIForUser(username
    );
    var whoamiIds = oldResponse.getModules().stream().map(
    EmployeeEvaluationModule::getId).collect(Collectors.toList()
    );
    return ResponseEntity.ok(whoamiIds);
}
```

Upravená verze služby Evaluation Period byla nasazena na development prostředí, načež proběhla úspěšná kontrola autorizování agend spolu se správným zobrazováním přístupů osob v editoru komponent.

10.4 Vyhodnocení testování

Definice komponent integrovaných systémů se podařilo vložit do nové Whoami služby prostřednictvím vzniklé administrativní agendy v UI platformy. Po připojení systémů poskytovala služba správné odpovědi, úspěšně jsme tedy vytvořili nástroj pro správu definic a nahradili jím stávající Whoami service a definice v User Preference service.

Nedostatkem našeho řešení bylo nastavení autorizačních pravidel komponent prostřednictvím jednotlivých Usermap rolí. Nastavení nebylo použitelné pro systém Eprocesů, kterým jsme chtěli tuto funkcionalitu ověřit. Kvůli tomuto nedostatku jsme rozšířili validační logiku Whoami služby o možnost vkládání rolí ve formátu regulárních výrazů. Po implementaci tohoto rozšíření bylo možné vytvořit kompletní autorizační schéma na úrovni Whoami služby.

Autorizace na úrovni Whoami služby ovšem přináší duplikaci již existující logiky uvnitř služeb Eprocesů. Tento fakt odhalil jistou přezíravost v návrhu našeho řešení, kde jsme možnost vzniku takovéto duplikace nevzali v potaz. **Je tedy možné, že se autorizace na úrovni nové Whoami služby ve výsledku použije jen ve výjimečných případech, jako jsou zmíněné vývojářské nástroje platformy.** Možnou variantou je také tento přístup docela obrátit a použít novou Whoami službu jako autorizační autoritu (služba poskytuje endpoint pro ověření přístupu uživatele k dané komponentě), na základě které by si integrované systémy ověřovaly přístupy k datům. Tento přístup je ovšem čistě hypotetický a má zjevné komplikace, jako je nadbytek síťových požadavků či zahlcení Whoami služby.

Dalším nedostatkem řešení bylo zpětně odhaleno mapování Whoami endpointů služeb podle jejich jmen zadaných v definicích. Služba totiž při získávání Whoami odpovědí služeb jednoduše skládá adresu služby s cestou `/whoami`. Některé služby ovšem používají složitější cestu, např. `/proxy/whoami` v případě Eprocesů. Pokud uživatel zadá absolutní URL, může přidat libovolný

prefix. V případě zadání jména služby však nemá cestu jak ovlivnit, protože URL služby se pak získává z discovery služby. Pro tento účel byl do Whoami definice přidán atribut *whoamiPath*, který lze nastavit v případě, že je cesta k Whoami endpointu jiná nežli standardní `/whoami`.

Kapitola 11

Vyhodnocení výsledků

V následující kapitole uvádíme zhodnocení výsledků prostřednictvím subjektivního hodnocení projektovým manažerem platformy, porovnáním výsledných hodnot metrik a úvahou nad dopadem řešení na integrační proces.

11.1 Validace projektovým manažerem platformy

Závěrem testování proběhla prezentace funkcionalit a integrovaných definic hlavnímu manažerovi projektu Hub, který výsledky zhodnotil jako přínosné a že se doslova „hrozně těší“ na jejich nasazení.

Jako nedostatkem nástroje bylo identifikováno nastavování přístupu k testovacímu prostředí, které neumožňuje hromadné přidávání uživatelů, např. podle rolí. Na základě tohoto podnětu byla do *Root* definice přidána nová agenda s identifikátorem *testAccess*, která určuje přístup k testovacímu prostředí. Resolver pro Whoami odpověď uživatele byl upraven tak, že pokud uživatel nemá přístup k této agendě, jeho Whoami odpověď je v testovacím prostředí prázdná. Tímto způsobem je administrátorům *Root* definice umožněno spravovat přístup podle uživatelského jména i rolí.

Díky tomuto zásahu již v nové Whoami službě prakticky není důvod uchovávat entity uživatele a veškeré záznamy spolu s GraphQL mutacemi pro správu uživatelů byly ze služby odstraněny. Z UI administrační agendy byly odebrány příslušné formuláře. Mutace *deleteWhoamServiceUser* byla nahrazena *removeAdminFromAllDefinitions*, která odebere příslušné uživatelské jméno administrátora ze všech jeho definic.

11.2 Výsledné hodnoty metrik

Díky nástroji na správu Whoami definic již není nutné ověření analytikem platformy, zdali má definice systému veškerá potřebná data a neobsahuje duplicitní identifikátory. Frontend vývojář nemusí kontrolovat strukturu komponent a backend vývojář není potřeba k validaci výstupních dat služeb. Hodnota metriky *Počet rolí k ověření správnosti Whoami definice* se tímto sníží na její optimální hodnotu nula.

Teoretická hodnota metriky *Počet rolí k implementaci Whoami definice* po aplikaci řešení je dvě, protože je teoreticky možné nastavit autorizační pravidla v rámci definice (role analytika) a frontend vývojář poté pouze doplní komponenty pro příslušné identifikátory. Vzhledem k tomu, že stávající integrované systémy tento přístup nebudou aplikovat a budou nadále udržovat implementaci Whoami endpointu (role backendového vývojáře), stanovíme výslednou hodnotu řešení na tři nutné role.

Výslednou hodnotu metriky *Duplikace Whoami definic* stanovujeme na jedna, protože jediné místo, kde se musí uložit struktura Whoami komponent systému je nová služba Whoami. Nová administrační agenda slouží zároveň i jako dokumentace (primární zdroj) a implementace (frontendová i backendová) vyžaduje pouze správnou hodnotu identifikátorů.

Název metriky	Výchozí hodnota	Optimální hodnota	Teoretická hodnota řešení	Výsledná hodnota
Počet rolí k ověření správnosti Whoami definice	3	0	0	0
Počet rolí k implementaci Whoami definice	4	1	2	3
Duplikace Whoami definic	4	1	1	1

Tabulka 11.1: Výsledný přehled hodnot metrik

11.3 Dopad na integrační proces

Definice Whoami formátu integrované aplikace je pouze částí procesu integrace a nemůžeme tedy prohlásit, že bychom na základě zlepšení hodnot daných metrik vyřešili všechny problémy procesu. Whoami definice jsou ovšem jedním ze základních artefaktů integrace systému do platformy Hub, který byl dosud vnímán spíše jako balast, zpravidla dodáván až v pozdějším stadiu integrací, pokud vůbec. Na základě změny metrik věříme, že implementované úpravy zjednoduší jejich správu a zdůrazní jejich roli v celém procesu.

Whoami formát je specifický produkt platformy a jako takový vyžaduje jistý „onboarding“ nových uživatelů v podobě vysvětlování formátu a jeho

fungování, natož používání nástroje na jeho správu. Z pohledu frontendového vývojáře platformy je ovšem jednoduché vyvodit strukturu Whoami komponent již podle hrubých představ uživatele a vytvořit tak základní definici, kterou bude uživatel moct následně libovolně rozvíjet. Definice by tímto byla dostupná již od rané fáze vývoje.

Pro kompletní integraci plně integrovaného systému jsou po uživateli vyžadovány následující analytické výstupy:

1. GraphQL schémata služby/služeb systému,
2. Podklady pro UX/UI tým k vytvoření návrhů (wireframy, konzultace apod.),
3. Whoami definice dané aplikace.

Již po vytvoření GraphQL schématu a jeho validaci frontendovým týmem je možné vyvíjet backendovou část systému, případně jeho adaptér v rámci infrastruktury. Ideální je služby vyvíjet pomocí knihovny LibCommon, která poskytuje veškeré nutné konfigurace (vývojáři mohou implementovat služby bez použití knihovny či v úplně jiném jazyce, v tom případě však musí implementovat její funkcionality sami). Pro nasazení integrované služby je nutné přidat její záznam do hlavního docker compose souboru s odkazem na korektně označený image. Pro usnadnění dodržování standardů release managementu je možné použít nově vytvořenou CI šablonu. Frontend se zpravidla vyvíjí až na základě UI návrhů, ačkoli v případě jednodušších funkcionalit je možné vycházet již z existujících komponent. V momentě, kdy

1. backendové služby systému implementují GraphQL schéma,
2. backendové služby implementují Whoami endpoint (v případě, že systém nepoužívá autorizaci na úrovni Whoami služby),
3. frontend obsahuje korektně namapované Whoami komponenty a
4. implementace UI odpovídá představám uživatele,

je možné po ověření funkčnosti integrovaného systému v testovacím prostředí nasadit produkční release (v koordinaci s ohledem na zásady release managementu).

11.4 Další kroky

Jak již bylo zmíněno, nasazení nové Whoami služby na testovací prostředí aktuálně závisí pouze na dokončení nové GraphQL federace platformy. Jakmile

budou tyto backendové služby v provozu, stačí ve frontendovém repozitáři provézt merge větve, na které probíhal vývoj UI nástroje, do větve hlavní. Během revize kódu v nástroji SonarQube jsme ovšem nalezli chybějící podporu pro přesměrování impersonačních požadavků na `/whoami` endpointy integrovaných služeb. Abychom mohli nástroj nasadit i do produkce, bude nutné tuto funkcionalitu doplnit. Dalším důležitým doplněním je přidání nástroje pro správu databázových migrací, např. Flyway [57].

Pro zajištění kvality kódu je dále naplánována revize nové Whoami service seniorním Java vývojářem CZM (kód frontendové části neplánujeme revidovat, autor je vedoucí frontendový vývojář platformy). Implementace služby dozajista obsahuje prostor pro různá vylepšení, během implementace služby jsme se soustředili především na její funkčnost na úkor optimalizací některých detailů (např. složitějších databázových dotazů). V kódu jsou tato místa označená komentářem „TODO“ a podle nástroje SonarQube je jich v projektu pět.

Co se týče nerealizovaných řešení z šesté kapitoly, v současné době již probíhá analýza podpory content managementu, vycházející z návrhu v sekci 6.4. Whoami identifikátory komponent jsou vhodným propojením CMS záznamů s částmi UI aplikace a řešení je proto navrhováno již s ohledem na utilizaci nové Whoami služby a administrační agendy.

11.4.1 Plánované integrace

Kromě probíhajících integrací Témat a Eprocesů se chystá připojení nového systému pro správu doktorandského studia pod záštitou CZM, který by již měl probíhat podle výše zmíněného procesu a používat nejnovější funkce platformy.

Dále probíhají konzultace se členy Střediska výpočetní techniky a informatiky, kteří projevili zájem o možnost integrace některých svých systémů do platformy Hub.

Kapitola 12

Závěr

Cíli této práce bylo analyzovat integrační platformu Hub, zmapovat slabé stránky a navrhnout způsoby vylepšení, vedoucí k zvýšení kvality procesu integrace systémů. Těchto cílů jsme dosáhli následujícím způsobem.

Prostřednictvím analýzy jsme zachytili aktuální stav platformy Hub, identifikovali problémy procesu integrace systémů a na jejich základě definovali kritéria kvality spolu s jejich metrikami. Navrhli jsme optimalizaci pouze těch metrik, které nebyly ovlivněny paralelně probíhajícím úpravám platformy. Společně se stakeholdery projektu jsme seřadili kritéria dle priority a nejvyšší váhu získaly metriky udávající použitelnost autorizačního schématu Whoami. Tyto metriky jsme optimalizovali prostřednictvím vytvoření nástroje na správu Whoami definic a nového Whoami formátu.

Nástroj se skládá z nové platformní služby a administračního modulu v rámci UI platformy. Implementovaná služba slouží jako primární zdroj autorizačních schémat, čímž redukuje duplikaci schémat výchozího stavu platformy a zároveň zjednodušuje formát dat vyžadovaný od integrovaných systémů. Díky nástroji může integrující uživatel libovolně upravovat a testovat svá autorizační schémata bez nutnosti nasazení nových verzí platformních služeb. Nový formát obsahuje rozšířená metadata, která dále umožňují integrujícím uživatelům částečně spravovat obsah svých aplikací a zjednodušují frontendovou implementaci.

Implementace řešení přinesla kromě zmíněného nástroje i vedlejší produkty jako rozšířené vyhledávání obsahu integrovaných aplikací či GitLab CI šablonu pro automatizaci git workflow, kompatibilní se zásadami release managementu platformy. Dále byl definován nový formát GraphQL odpovědí pro indikaci chybných stavů s příslušnými frontendovými utilitami. Nová služba je první platformní službou s vlastním GraphQL rozhraním a je proto závislá na nové GraphQL federaci. Federace bohužel nebyla v době vypracování této práce plně funkční a službu se proto nepodařilo nasadit na testovací prostředí.

Pro testování funkcionality byla v nástroji vytvořena autorizační schémata

integrovaných systémů a připojeny služby s implementovaným Whoami rozhraním. Připojení služeb proběhlo úspěšně. Definování autorizačních pravidel na úrovni schématu objevilo nedostatky v používání Usermap rolí, které byly vzápětí opraveny a veškeré funkcionality nástroje byly úspěšně ověřeny. Po konzultaci s vedoucím projektu Hub byla dodatečně upravena funkcionalita ke správě přístupů na testovací prostředí.

Hodnoty všech vybraných metrik se podařilo zlepšit, ačkoli ne u všech bylo dosaženo očekávaných hodnot řešení. Výsledný stav metrik obsahuje tabulka 11.1. V rámci práce jsme se ovšem kvůli dynamickému prostředí platformy zaměřili pouze na metriky podmnožiny systému a nepodařilo se nám tak vytvořit ucelenou sadu kritérií, která by mohla sloužit ke kontinuálnímu ověřování kvality celého systému.

Schéma autorizací Whoami je páteřní funkcionalitou platformy, která dosud nebyla integrujícím uživatelům příliš srozumitelná a pro její aplikaci či jednoduché úpravy bylo zpravidla potřeba součinnosti několika rolí. Věříme, že nový nástroj zjednoduší použitelnost platformy a zároveň povýší autorizační formát na její plnohodnotný *feature* (nejen povinnost integrace).

Literatura

[1] LIPOWSKI Adam. Aplikace pro podporu hodnocení zaměstnanců na Čvut fel. Master's thesis, 2022.

[2] APOLLO GRAPH INC. Introduction to apollo federation. dostupné z `https://www.apollographql.com/docs/federation/`. navštíveno 1.5.2023.

[3] SOFTWARE QUALITY METRICS METHODOLOGY WORKING GROUP. Ieee standard for a software quality metrics methodology. *IEEE Std 1061-1992*, pages 1–96, 1993.

[4] ALTEXSOFT. System integration: Types, approaches, and implementation steps. dostupné z `https://www.altexsoft.com/blog/system-integration/`. navštíveno 11.4.2023.

[5] FOLIO3 DYNAMICS. Integration systems: What is system integration? types of system integration - advantages of each method - a complete guide. dostupné z `https://dynamics.folio3.com/blog/system-integration/`. navštíveno 11.4.2023.

[6] BIGCOMMERCE. Systems integration: Types and methods + how to connect systems. dostupné z `https://www.bigcommerce.com/articles/business-management/systems-integration/`. navštíveno 11.4.2023.

[7] CZM. Podnět a cíl projektu. Interní dokumentace CZM, 2021.

[8] CZM. Vize - jednotný systém. Interní dokumentace CZM, 2020.

[9] CZM. Jednotný frontend: Dokument a4. Interní dokumentace CZM, 2020.

[10] CZM. Projektový plán. Interní dokumentace CZM, 2022.

[11] CZM. Home stránka wiki projektu. Interní dokumentace CZM, 2022.

[12] MOZILLA. Spa (single-page application). dostupné z https://developer.mozilla.org/en-US/docs/Glossary/SPA. navštíveno 23.11.2022.

[13] DOBRICA Liliana and OVASKA Eila. A survey on software architecture analysis methods. *Software Engineering, IEEE Transactions on*, 28:638–653, 2002.

[14] LEWIS James and FOWLER Martin. Microservices. dostupné z https://martinfowler.com/articles/microservices.html. navštíveno 23.11.2022.

[15] VMWARE. Messaging that just works — rabbitmq. dostupné z https://www.rabbitmq.com/. navštíveno 23.11.2022.

[16] INC. NETFLIX. Netflix/eureka: Aws service registry for resilient mid-tier load balancing and failover. dostupné z https://github.com/Netflix/eureka. navštíveno 8.1.2023.

[17] SMARTBEAR SOFTWARE. Openapi specification - version 3.0.3 | swagger. dostupné z https://swagger.io/specification/. navštíveno 18.12.2022.

[18] VMWARE. Spring boot. dostupné z https://spring.io/projects/spring-boot. navštíveno 23.11.2022.

[19] GITLAB. Package registry. dostupné z https://docs.gitlab.com/ee/user/packages/package_registry/index.html. navštíveno 23.11.2022.

[20] THE APACHE SOFTWARE FOUNDATION. Maven – introduction. dostupné z https://maven.apache.org/what-is-maven.html. navštíveno 23.11.2022.

[21] T. LODDERSTEDT, J. RICHTER, and B. CAMPBELL. Oauth 2.0 rich authorization requests. dostupné z https://www.ietf.org/archive/id/draft-ietf-oauth-rar-16.html. navštíveno 23.11.2022.

[22] FIELDING Roy T. and RESCHKE Julian. Hypertext transfer protocol (http/1.1): Authentication. dostupné z https://datatracker.ietf.org/doc/html/rfc7235. navštíveno 23.11.2022.

[23] KEYCLOAK TEAM. Keycloak. dostupné z https://www.keycloak.org/. navštíveno 10.5.2023.

[24] GRAPHQL CONTRIBUTORS. Graphql. dostupné z https://spec.graphql.org/October2021/. navštíveno 23.11.2022.

[25] MOZILLA. Websocket. dostupné z https://developer.mozilla.org/en-US/docs/Web/API/WebSocket. navštíveno 23.11.2022.

[26] SCHAFER Dan and KUENZEL Laney. Subscriptions in graphql and relay. dostupné z https://graphql.org/blog/subscriptions-in-graphql-and-relay/. navštíveno 24.11.2022.

[27] GITLAB. Gitlab ci/cd. dostupné z https://docs.gitlab.com/ee/ci/. navštíveno 23.11.2022.

[28] DOCKER INC. docker images | docker documentation. dostupné z https://docs.docker.com/engine/reference/commandline/images/. navštíveno 24.11.2022.

[29] DOCKER INC. Swarm mode overview | docker documentation. dostupné z https://docs.docker.com/engine/swarm/. navštíveno 24.11.2022.

[30] DOCKER INC. Compose specification | docker documentation. dostupné z https://docs.docker.com/compose/compose-file/. navštíveno 24.11.2022.

[31] GITLAB. The .gitlab-ci.yml file. dostupné z https://docs.gitlab.com/ee/ci/yaml/gitlab_ci_yaml.html. navštíveno 25.11.2022.

[32] PROMETHEUS AUTHORS. Prometheus - monitoring system time series database. dostupné z https://prometheus.io/. navštíveno 30.4.2023.

[33] VMWARE. Spring cloud sleuth. dostupné z https://spring.io/projects/spring-cloud-sleuth. navštíveno 9.1.2023.

[34] GRAFANA LABS. Grafana: The open observability platform | grafana labs. dostupné z https://grafana.com/. navštíveno 9.1.2023.

[35] ELASTICSEARCH B.V. Kibana: Explore, visualize, discover data | elastic. dostupné z https://www.elastic.co/kibana/. navštíveno 9.1.2023.

[36] FACEBOOK. React docs beta. dostupné z https://beta.reactjs.org/. navštíveno 24.11.2022.

[37] CZM. Tagging standards. Interní dokumentace CZM, 2022.

[38] GIT. Git - tagging. dostupné z https://git-scm.com/book/en/v2/Git-Basics-Tagging. navštíveno 5.12.2022.

[39] PRESTON-WERNER Tom. Semantic versioning - 2.0.0. dostupné z https://semver.org/spec/v2.0.0.html. navštíveno 5.12.2022.

[40] CAMUNDA. The universal process orchestrator | camunda. dostupné z https://camunda.com/. navštíveno 18.12.2022.

[41] FAGERHOLM Fabian and MÜNCH Jürgen. Developer experience: Concept and definition. 2012.

[42] KUKREJA Nupul. Measuring software maintainability. dostupné z https://quandarypeak.com/2015/02/measuring-software-maintainability/. navštíveno 22.12.2022.

[43] GITLAB. Commits api. dostupné z https://docs.gitlab.com/ee/api/commits.html#create-a-commit-with-multiple-files-and-actions. navštíveno 6.1.2023.

[44] INC. MONGODB. Mongodb: The developer data platform | mongodb. dostupné z https://www.mongodb.com/. navštíveno 6.1.2023.

[45] VMWARE. 7. declarative rest client: Feign. dostupné z https://cloud.spring.io/spring-cloud-netflix/multi/multi_spring-cloud-feign.html. navštíveno 13.4.2023.

[46] VMWARE. Spring security :: Spring security. dostupné z https://docs.spring.io/spring-security/reference/index.html. navštíveno 13.4.2023.

[47] NETFLIX. Home - dgs framework. dostupné z https://netflix.github.io/dgs/. navštíveno 1.4.2023.

[48] THE POSTGRESQL GLOBAL DEVELOPMENT GROUP. Postgresql: The world's most advanced open source database. dostupné z https://www.postgresql.org/. navštíveno 16.4.2023.

[49] INC. POSTMAN. What is postman? postman api platform. dostupné z https://www.postman.com/product/what-is-postman/. navštíveno 17.4.2023.

[50] INC. POSTMAN. Setting up mock servers | postman learning center. dostupné z https://learning.postman.com/docs/designing-and-developing-your-api/mocking-data/setting-up-mock/. navštíveno 17.4.2023.

[51] OPENJS FOUNDATION. Node.js. dostupné z https://nodejs.org/en. navštíveno 17.4.2023.

[52] INC. POSTMAN. Running collections on the command line with newman | postman learning center. dostupné z https://learning.postman.com/docs/collections/using-newman-cli/command-line-integration-with-newman/. navštíveno 19.4.2023.

[53] GITLAB. Using postgresql | gitlab. dostupné z https://docs.gitlab.com/ee/ci/services/postgres.html. navštíveno 19.4.2023.

[54] CZM. Release management. Interní dokumentace CZM, 2022.

[55] REMIX. React router: Declarative routing for react.js. dostupné z https://v5.reactrouter.com/web/guides/quick-start. navštíveno 24.4.2023.

[56] GRAPHQL. graphql/graphiql: Graphiql the graphql lsp reference ecosystem for building browser ide tools. dostupné z `https://github.com/graphql/graphiql`. navštíveno 8.5.2023.

[57] RED GATE SOFTWARE LTD. Homepage - flyway. dostupné z `https://flywaydb.org/`. navštíveno 19.5.2023.

Přílohy

Příloha A

Použité zkratky

API Application programming interface

CI/CD Continuous integration/continuous delivery

CRUD Create, read, update, delete

DTO Data transfer object

HTTP Hyper text transfer protocol

REST Representational state transfer

SoC Separation of Concerns

SPA Single page application

SSO Single sign on

UI User interface

UX User experience

Příloha B

Terminologie

Code coverage. Procentuální hodnota kódu pokrytého testy.

Cohesion. Míra do které jsou části softwarového modulu účelově spřízněny.

Coupling. Míra závislosti mezi softwarovými komponentami.

Cyklomatická složitost. Metrika udávající složitost programu na základě lineárně nezávislých logických cest ve zdrojovém kódu.

Edge case. Událost či kombinace vstupů, která vzniká s malou četností při používání software nástroje.

Feature coverage. Procentuální hodnota funkcionalit, pro které existují testy.

Framework. Softwarová služba poskytující řídící logiku aplikace, do které uživateli stačí implementovat funkční prvky specifické pro jeho případy užití.

Impersonace. Použití aplikace uživatelem přihlášeným za jinou osobu.

Separation of Concerns. Návrhový princip popisující rozdělení zodpovědností částí softwaru.

Single responsibility principle. Přístup k návrhu architektury, který přikazuje každé části (funkce,třída, modul) nejvýše jednu zodpovědnost/-účel.

Stakehholder. Osoba nebo skupina osob, která ovlivňuje či může být ovlivněna vstupy a výstupy projektu.

Use case. V překladu případ užití, neboli akce, kterou lze vykonat prostřednictvím systému. Lze ji také chápat jako požadavek na funkcionalitu systému.

Příloha C

Zdrojové kódy

Veškeré zdrojové kódy implementovaného řešení jsou k dispozici na fakultním GitLabu. Přikládáme odkazy na konkrétní stavy repozitářů v době odevzdání práce. Pro získání přístupu lze kontaktovat autora.

Nová Whoami service: `https://gitlab.fel.cvut.cz/czm/hub/backend/whoami-service-new/-/tree/9ded2024`

Hub frontend: `https://gitlab.fel.cvut.cz/czm/hub/frontend/frontend-base/-/tree/da5bba8f`