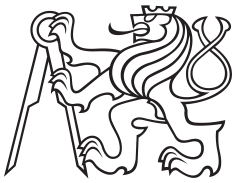
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická

Integrace systému Hodnocení doktorandů do systému HUB.FEL

Jiří Štěpán

Vedoucí práce: Ing. Lukáš Zoubek
Obor: Softwarové inženýrství a technologie
Květen 2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Štěpán** Jméno: **Jiří** Osobní číslo: **483549**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Integrace systému Hodnocení doktorandů do systému HUB.FEL

Název bakalářské práce anglicky:

Integration Doctoral study system into system HUB.FEL

Pokyny pro vypracování:

Vytvořte integrační vrstvu mezi již existujícím procesem Hodnocení doktorandů v IBM BPM a nově vyvíjeným samostatným frontendem:
o Integrační vrstva bude poskytovat aplikační rozhraní (API) pro získání příslušných dat ze systému IBM BPM. Toto rozhraní se bude skládat ze dvou modulů.
o První modul bude poskytovat aplikační data, která se budou dále v HUB.FEL organizovat do dashboardů.
o Druhý modul bude řešit komunikaci s frameworkem IBM BPM pro poskytování metadat o úkolech, které byli uživatelům v rámci procesu zadány. Současně s poskytováním metadat, bude integrační vrstva řešit problémy spojené se správným přesměrováním do formuláře pro splnění úkolu.
- Sestavte sadu testovacích scénářů pokrývající code-coverage aplikace a proveďte testování. Všechny nalezené chyby zaevidujte a opravte.
- Aplikaci nasaďte na server a spusťte do testovacího provozu.

Seznam doporučené literatury:

1. VALACICH, Joseph a Christoph SCHNEIDER. Information SystemsToday – Managing in the Digital World. 4th edition. Upper Saddle River, New Jersey: Prentice-Hall, 2010. ISBN 9780136078401.
2. THONG, James Y. L, CHEE-SING, YAP a K. S. RAMAN, STOWELL, Frank A., Daune WEST a James G HOWELL, ed. User Satisfaction as a Measure of Information System. Boston, MA: Springer US, 1993. ISBN 978-1-4615-2862-3.
3. GULLEDGE, Thomas. What is integration? Industrial Management and Data Systems. 2006, ISSN 0263-5577.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Lukáš Zoubek, katedra softwarového inženýrství FIT

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

Datum zadání bakalářské práce: **11.02.2021** Termín odevzdání bakalářské práce: **21.05.2021**

Platnost zadání bakalářské práce: **30.09.2022**

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

III. PŘEVZETÍ ZADÁNÍ

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

. _____
Datum převzetí zadání

_____
Podpis studenta

Poděkování

Děkuji svému vedoucímu Ing. Lukáši Zoubkovi za vstřícnost a velmi cenné rady, které mi dával na průběžných konzultacích po celou dobu psaní této práce.

Dále děkuji své rodině a blízkým, kteří mě i v nepříznivých podmínkách vždy podporovali.

Prohlášení

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

V Praze, 20.května 2021

Abstrakt

Práce se zabývá problematikou integrace informačního systému Hodnocení doktorandů do plánovaného nového systému HUB.FEL. Cílem práce je navrhnout toto propojení a implementovat ho.

Rešeršní část se skládá z obecného úvodu do problematiky informačních systémů a jejich důvodů k integraci. Dále je uvedeno několik vhodných metod pro integraci.

Další část práce je zaměřena na popis systémů a navrhované integrační vrstvy.

Poslední část práce se zabývá implementací a nasazení integrační vrstvy do testovacího provozu.

Klíčová slova: informační systém, integrace, REST rozhraní, mikroslužby, IBM BPM, úkol, Spring Boot

Vedoucí práce: Ing. Lukáš Zoubek
Centrum znalostního managementu, ČVUT FEL

Abstract

This thesis researches problems about integration of Doctoral study systems to a brand new system HUB.FEL. The main goal of the thesis is present the concept of solution that meets the conditions for integration.

The research part is consist of general introducing into this topic of system integration and its reasons to integrate. Next, a few examples of methods appropriate to integration are presented.

Furthermore, the work is devoted to the description of both systems and the proposed integration layer.

The rest of the work deals with the implementation of integration layer into the test operation.

Keywords: information system, integration, REST interface, microservices, IBM BPM, task, Spring Boot

Title translation: Integration Doctoral study system into HUB.FEL system

Obsah

Obrázky

Tabulky

Část I

Zadání, teoretická část

Kapitola 1
Úvod

V dnešním moderním světě se potkáváme s mnoha informačními systémy. Ať už se jedná o systémy v zaměstnání či na univerzitě, v obou případech přinášejí nemalé zjednodušení chodu organizace. Pro většinu společností představují klíčový stavební kámen, který je součástí jejich fungování.

Obvykle jedna organizace používá pro svůj chod i vícero informačních systémů. Poté ovšem může nastat situace, kdy společnost touží tyto dva oddělené systémy propojit do jednoho. V takovém případě se jedná o proces zvaný integrace.

1.1 Zadaní práce

Zadáním bakalářské práce je integrovat systém Hodnocení doktorandů do nově vyvíjeného systému HUB.FEL. Pokyny k zadání jsou následující:

1. Vytvořte integrační vrstvu mezi již existujícím procesem Hodnocení doktorandů v IBM BPM a nově vyvíjeným samostatným frontendem HUB.FEL:
   - Integrační vrstva bude poskytovat aplikační rozhraní (API) pro získání příslušných dat ze systému IBM BPM. Toto rozhraní se bude skládat ze dvou modulů.
   - První modul bude poskytovat aplikační data, která se budou dále v HUB.FEL organizovat do dashboardů.
   - Druhý modul bude řešit komunikaci s frameworkem IBM BPM pro poskytování metadat o úkolech, které byli uživatelům v rámci procesu zadány. Současně s poskytováním metadat, bude integrační vrstva řešit problémy spojené se správným přesměrováním do formuláře pro splnění úkolu.
2. Sestavte sadu testovacích scénářů pokrývající code-coverage aplikace a proveďte testování. Všechny nalezené chyby zaevidujte a opravte.
3. Aplikaci nasaďte na server a spusťte do testovacího provozu.

1.2 Motivace

Na elektrotechnické fakultě ČVUT se používá v současné době přibližně dvacet fakultních systémů. Každý systém byl vyvinut pro jiný účel. Motivací této práce je přinést uživatelům zjednodušení v podobě dostupnosti dat a poskytnutí jednotného rozhraní, kde uvidí všechny aktuální úkoly, které byly uživateli v rámci fakultních systémů přiřazeny.

Nedílnou součástí je také poznat a vyzkoušet si hlouběji týmovou práci v univerzitním prostředím. Podílet se na vývoji a prohloubit svoje technické znalosti a dovednosti v oblasti webových služeb a aplikací.

1.3 Cíle práce

Cílem této práce je projít jednotlivé části vývoje webové služby od analýzy, návrhu, implementaci až po spuštění finální aplikace na serveru.

Pro snadnější orientaci jsem pokyny předefinoval do těchto cílů. :

1. Provést analýzu obou zmíněných systémů a navrhnout vhodnou integrační vrstvu včetně architektury a vhodných technologií k použití (pokyn č.1).
2. Zajistit, aby integrační vrstva poskytovala systému HUB.FEL REST rozhraní, které poskytuje všechna data, pokrývající proces hodnocení doktorandů (pokyn č.1 - modul 1).
3. Zajistit, aby výsledné REST rozhraní splňovalo podmínky pro integraci do systému HUB.FEL. Jedná se například o možnosti filtrování a stránkování výsledků nebo o poskytování dat o uživateli a jeho právech (pokyn č.1 - modul 1).
4. Prozkoumat technologii IBM BPM a implementovat řešení, které v rámci procesu hodnocení doktorandů přijímá informace o vytvořeném či upraveném úkolu. Implementace zároveň řeší i problematiku přesměrování na jeho vykonání. Všechny tyto informace přeposílá do systému HUB.FEL (pokyn č.1 - modul 2).
5. Vytvořit sadu vhodných testovacích scénářů a implementované řešení otestovat. Všechny nalezené chyby následně opravit a zaevidovat (pokyn č.2).
6. Finální integrační řešení spustit do testovacího provozu a nasadit do infrastruktury systému HUB.FEL (pokyn č.3).

Kapitola 2
Informační systém

2.1 Co je informační systém?

Než si blíže přiblížíme integraci informačních systémů, zaměříme se na definici informačního systému jako takového. Z hlediska informačních technologií může být informační systém (IS) definován jako soubor vzájemně propojených komponent skládající se z hardwaru a příslušného softwaru, který lidé používají ke shromažďování, vytváření a distribuci dat, typicky v rámci podnikového procesu.[1]

Na informační systém se tedy můžeme dívat jako na podnikový nástroj, který usnadňuje řízení a chod společnosti. V dnešním světě ho drtivá většina společností používá jako nezbytný nástroj za účelem uchování a analýzu firemních údajů a informací.

2.2 Přínosy informačního systému

Důvodem, proč jsou IS natolik rozšířeným nástrojem, jsou jejich výhody a přínosy, které s sebou přinášejí. Mezi hlavní přínosy patří především zvýšení efektivity pracovníků. Díky centralizaci informací jsou data snadno dohledatelná a není problém je tedy kdykoliv získat a provést nad nimi další operace. Zároveň stoupá i kvalita získaných informací.

Informační systémy přinášejí značné ulehčení i z ekonomického hlediska. Pokud náklady na provoz nepřevyšují rozpočet podniku, stává se IS důležitou součástí, která slouží jako středobod společnosti.[2]

2.3 Faktory určující kvalitu IS

Existuje mnoho faktorů a měřítek pro sledování kvality informačního systému. Jeden z nich je indikátor uživatelské spokojenosti.[3] Skupina výzkumníků se ve své studii zaobírala otázkou, jaké faktory ovlivňují uživatelskou spokojenost nejvíce.[4] Definovala mnoho různých atributů rozdělených do sedmi dimenzí, ke kterým vypočítala procentuální ohodnocení, jak moc velký mají dopad na celkové uživatelské spokojenosti.

Pro tuto práci nejzajímavějšími výsledky jsou:

- Velký podíl na výsledném hodnocení je připisován kvalitě získané informace. Ta se skládá například z toho, jak moc je informace přesná ($\sim 67\%$), v jakém je formátu ($\sim 46\%$) a jak moc ji uživatel může dále personifikovat ($\sim 35\%$).
- Důležitým faktorem, který zvyšuje spokojenost je jednoduché ovládání informačního systému ($\sim 67\%$). S tím lehce souvisí i doba odezvy ($\sim 37\%$).
- Nemalý podíl je také přiřazován poskytování dat z jiného systému ($\sim 25\%$).

Z výsledků jejich práce můžeme vidět, že uživatelskou spokojenost ovlivňuje mnoho aspektů. Abychom vytvořili kvalitní informační systém, musíme tyto aspekty brát od počátku vývoje v potaz.

Kapitola 3
Systémová integrace

Z předchozí kapitoly již víme, že pokud má být IS kvalitní, musí být uživatelsky přívětivý a svým uživatelům musí poskytovat věrný zdroj informací. V určitých případech je ale potřeba pracovat i s informacemi z jiných systémů. K tomu je potřeba tzv. systémová integrace.

3.1 Definice

Pojem systémová integrace (SI) lze definovat mnoha způsoby. Autor Thomas Gulledge ve svém vědeckém článku popisuje integraci jako propojení více informačních systémů dohromady, takže informace a data procházejí skrz různé technologické prostředky.[5]

Jiný zdroj systémovou integraci orientuje blíže směrem k firemním business procesům v organizaci a uvádí ji jako spojení systémových částí do jednoho celku. Jako cíl integrace poté uvádí: *"Cílem je taková architektura informačního systému jakožto celku, která efektivně podporuje business procesy v organizaci."* [6]

3.2 Hlavní benefity integrace

Propojení informačních systémů nevede pouze ke zvýšení kvality. Důvodů k integraci je mnohem víc. Mezi hlavní pilíře se řadí zvýšení efektivity zaměstnanců.[7] Citovaná studie poukazuje na to, že se díky integraci eliminovaly bariéry mezi odděleními. Tím se zlepšila motivace pracovníků a následná efektivita.

Mezi další benefity patří snížení redundantních dat. Pokud jsou stejná data duplikována napříč dalšími systémy, je náročné udržovat informace aktuální. Po integraci není již třeba provádět vícenásobné aktualizace dat v každém IS zvlášť.[8]

3.3 Typy integrace

Existuje několik způsobů a metod vhodných pro integraci.[9] Pro potřeby této práce se zaměříme na dvě nejzákladnější metody integrace, na které můžeme narazit.

3.3.1 API

Nejvíce rozšířeným druhem integrace je tzv. API. Jedná se o způsob komunikace mezi dvěma informačními prvky, kdy na základě odeslaného požadavku dojde ke zpracovaní a zpětnému odeslání zprávy s výslednými daty.[11] Toto řešení se často uplatňuje mezi organizacemi, kdy API slouží jako rozhraní pro komunikaci s backendovými systémy.[10]

3.3.2 Webhooks

Narozdíl od API, integrace typu Webhooks je založena na událostech. V některé literatuře se často popisuje jako reverzní API nebo jako web callback.[12] Webhooks posílá data ostatním konzumentům v momentě, kdy nastane určitá událost.[13] Výhodou tohoto typu integrace je hlavně dostupnost dat v ostatních systémech v reálném čase.[12]

Část II

Popis systémů

Kapitola 4

Systém hodnocení doktorandů

V této kapitole si blíže přiblížíme systém Hodnocení doktorandů, jenž je vyvinut pomocí technologie IBM BPM. Součástí kapitoly bude i analýza architektury systému, kterou poté využijeme pro návrh integrační vrstvy.

4.1 Popis

Jak už z názvu vyplývá, hlavní činností systému je zastřešit celý proces hodnocení doktorandů na fakultě FEL. Tento systém byl vyvinut v rámci organizace CZM FEL ČVUT a je doposud stále používán. Podle systémové role nabízí uživatelům různé funkce a tabulky (dashboardy) k zobrazení.

Proces začíná vypsáním nového termínu pro zápis semestrálního hodnocení a končí závěrečným ohodnocením oborovou radou FEL. Během procesu se podle aktuální pozice generují jednotlivé úkoly, které musejí být pro pokračování procesu splněny.

Vyvinout takový systém od nuly by bylo příliš komplikované a zbytečně časově náročné. Proto se rozhodlo o použití technologie IBM BPM, která byla pro tyto účely vybudována.

4.2 Životní cyklus úkolu v procesu

V rámci procesu hodnocení doktorandů vzniká několik úkolů. Některé jsou přiřazeni pouze jednomu konkrétnímu uživateli. Tento typ úkolu přechází ze stavu *vytvořen* na *splněn* nebo na stav *po vypršení termínu*, pokud uživatel nestihne úkol vypracovat včas.

Některé úkoly jsou poté přiřazeni na celou skupinu pracovníků. V takových případech týmový úkol přináší ještě navíc stav *přisvojen* (angl. claimed). Jedná se o stav, kdy konkrétní uživatel ze skupiny si úkol nárokuje splnit. Ostatním z týmu se úkol odebere. Uživatel má možnost vrátit akci zpět. Tím se úkol přidá zpět všem uživatelům skupiny.

V rámci návrhu řešení je potřeba vycházet z těchto stavů a zajistit, aby integrační vrstva na tyto akce správně reagovala a provedla příslušné operace směrem k systému HUB.FEL.

4.3 Technologie IBM BPM

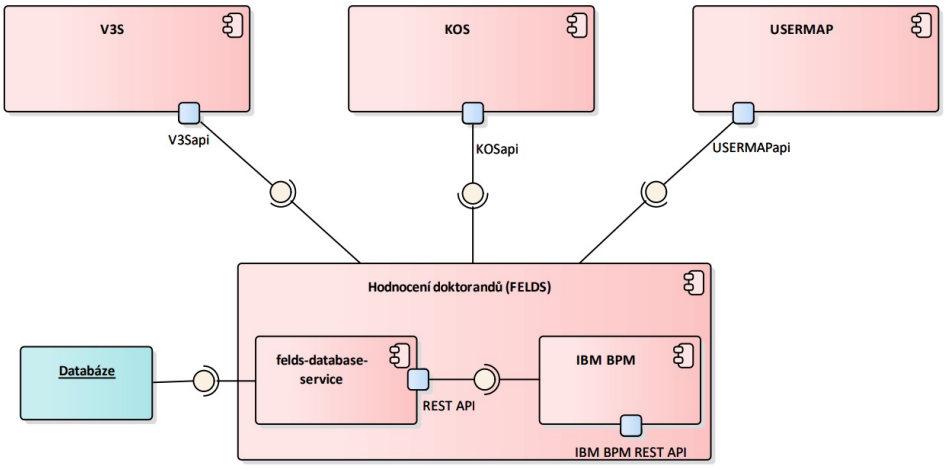
Technologie IBM BPM (Business process management) od společnosti IBM je druh softwaru, který poskytuje řadu nástrojů pro tvorbu komplexního systému v organizaci.[14] Budeme-li mluvit obecněji o BPM, zjistíme, že se jedná o styl řízení společnosti, v němž se celá organizace řídí firemními procesy. Procesy tedy hrají v BPM klíčovou roli.[15] Mnoho organizací volí BPM řešení hlavně díky možnosti neustálého monitoringu a rychlé reakce na příchozí změny.[15]

Cílem této technologie je automatizovat, digitalizovat a optimalizovat současné business procesy a zvýšit tím jejich efektivitu.[14]

4.4 Architektura systému

Při bližším zkoumáním architektury Hodnocení doktorandů (pracovně FELDS) zjistíme, že systém v sobě neobsahuje pouze modul IBM BPM. Jak dokládá obrázek č.4.1, vedle BPM modulu se zde nachází i komponenta nazvaná felds-database-service. Ta slouží jako servisní vrstva mezi databází a jádrem systému. Komponentě IBM BPM poskytuje REST API rozhraní pro načítání a zápis interních dat z databáze.

Ke svému správnému fungování potřebuje FELDS v rámci školní sítě komunikovat i s ostatními systémy. Konkrétně se jedná o systémy KOS, V3S a Usermap. Systémy mezi sebou komunikují pomocí API rozhraní, které má každý systém k dispozici.



Obrázek 4.1: Diagram komponent systému Hodnocení doktorandů

Modul IBM BPM mimo jiné ještě v sobě obsahuje svoje vlastní REST API rozhraní. To slouží k poskytování interních dat ohledně procesního řízení. Rozhraní může například poskytovat informace o konkrétní instanci, ve které se proces zrovna nachází.[16]

Kapitola 5
HUB.FEL

Nyní se podíváme na bližší specifikaci nového systému HUB.FEL, který se v době psaní této práce nachází ve fázi vývoje. Jeho nasazení do testovacího provozu je naplánováno na druhé pololetní roku 2021.

5.1 Vize

Myšlenka projektu spočívá v zjednodušení interakce uživatelů s fakultními systémy. A to pomocí sjednocení okolních systémů do jednotného celku tvořící ekosystém a následného vytvoření jednotného uživatelského rozhraní. Sjednocení bude probíhat na bázi agend, kdy bude systém rozdělen do logických částí, tak aby každá agenda odpovídala činnosti externího systému.

5.2 Cíle projektu

Jak již vyplývá z vize, hlavním cílem systému HUB.FEL je tedy zjednodušit uživatelskou interakci s fakultními systémy. Uživatelé mají k dispozici všechny data a přidělené úkoly z integrovaných systémů. Z HUB.FEL mají poté možnost tyto úkoly vykonat. A to buď přímo ze systému nebo přes odkaz, který je na externí systém přesměruje.

V době psaní této práce není zatím cílem integrovat všechny fakultní systémy. V první fázi vývoje se plánuje integrovat systém Témata disertačních prací a systém Hodnocení doktorandů.

Dalším cílem je snížit nutnost tvorby nových systémů. Při tvorbě nového systému proběhne nejdříve analýza, zda není možné plánované funkce rozšířit na nějaké již vytvořené agendě.

5.3 Současný stav

Na elektrotechnické fakultě se v současné době používá přibližně dvacet fakultních systémů. Každý systém byl vyvinut pro jiný účel. Jedná se například o systémy Moodle, Témata disertačních prací nebo výše zmíněný systém Hodnocení doktorandů. Každý systém používá různá zainteresovaná skupina uživatelů. Každý z těchto systémů poskytuje odlišné uživatelské rozhraní.

5.4 Úroveň integrace

Míra integrace se odráží podle toho, jak moc je externí systém benevolentní k integraci. Po zvážení různých přístupů k integraci systémů se definovaly tyto úrovně:

- **Integrace jako modul** - z externího systému je vytvořen modul, který je připojen k centrálnímu systému a vytváří tak jádro ekosystému.
- **Plná integrace** - pro přístup k datům a ovládacím prvkům se používá rozhraní API. Všechny formuláře a tabulky se vykreslují do jednotného vzhledu.
- **Částečná integrace** - systém má samostatný frontend. Zde potom záleží na možnostech systému a zvolené integraci:
  - **Obousměrná** - systém sdílí svoje ovládací prvky s HUB.FEL a naopak. Komunikace tedy může probíhat oběma směry (např. vyhledávání skrz vyhledávací modul centrálního systému).
  - **Jednosměrná** - komunikace je možná pouze směrem z centrálního systému (např. chytrý odkaz, který zobrazí konkrétní data)

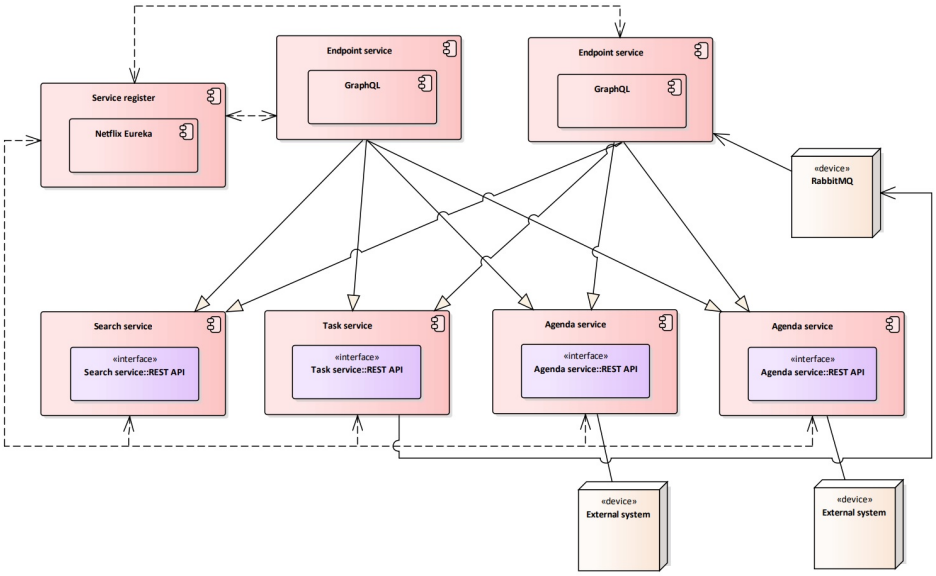
5.5 Architektura systému

Bylo rozhodnuto, že architektura HUB.FEL se bude držet modelu mikroslužeb (angl. microservices). Tento způsob architektury nám umožňuje snadno a efektivně integrovat různé systémy, které jsou postaveny na různých technologií. Zároveň také přináší značnou škálovatelnost a flexibilitu, co se týče budoucího rozšíření o nové funkce, či systémy. Další výhodou tohoto řešení je fakt, že jednotlivé mikroslužby mohou běžet na různých serverech. Nejsou tedy přímo vázány na jeden aplikační server. Jednotlivé moduly poté spolu komunikují skrz REST rozhraní.

High-level pohled níže ukazuje fragmentaci do jednotlivých mikroslužeb, které společně tvoří jádro systému. Jednotlivými komponenty jsou:

- **Agenda service** - představuje jednu konkrétní agendu, která komunikuje s externím systémem.
- **Task service** - má na starost správu úkolů a notifikací. Slouží jako centrální modul pro jejich příjem a práci s nimi. Má definované REST API rozhraní a při jeho zavolání si Task service uloží příchozí úkoly a odešle je do webové fronty RabitMQ.
- Jako hlavní komponentu považujeme **Service register**. Ta registruje všechny ostatní mikroslužby a vrací jejich instance.
- **Endpoint services** - uchovávají informace o tom, kde jsou jaká data dostupná, jakou konkrétní mikroslužbu mají zavolat.

- **Search service** - je odpovědná za vyhledávání jak v centrálním systému, tak i v jednotlivých agendách, které to umožňují.



Obrázek 5.1: High-level pohled na architekturu systému HUB.FEL

O podobě architektury bylo rozhodnuto ještě před zahájením této práce. Nebylo tedy možné vést debatu o jiném způsobu integrace než takovým, který vychází z modelu microservices architektury - tedy vytvořit novou Agenda service, která bude představovat systém Hodnocení doktorandů a bude připojena jako modul do jádra HUB.FEL.

Část III

Navrhované řešení

Kapitola 6

Integrační vrstva

V této kapitole si přiblížíme moje navrhované řešení, které řeší problematiku integrace systému Hodnocení doktorandů se systémem HUB.FEL. Toto navrhované řešení popisuje model jedné konkrétní Agenda service (viz. předchozí kapitola). Navrhované řešení jsem průběžně konzultoval s ostatními členy týmu na pravidelných schůzkách.

6.1 Požadavky na implementaci

Než začnu představovat návrh řešení, je potřeba zmínit požadavky, které mi byly společně se zadáním sděleny. Jedná se o tyto požadavky:

1. Mikroslužba obsahuje REST API, která uživatelům nabízí všechny tabulky (dashboardy) a ovládací prvky ze systému Hodnocení doktorandů.
2. Agenda service komunikuje s Task service a předává mu informace o úkolech a notifikací.
3. Včetně odesílání dat o úkolu, bude integrační vrstva na základě pravidel generovat odkaz, který uživatele přesměruje na vykonání úkolu.
4. Mikroslužba je stabilní vůči možným výpadkům klientské strany. Pokud klientská strana vypoví službu nebo její odpověd trvá příliš dlouho, vyhodí chybovou hlášku.
5. Uživatelům mikroslužba poskytuje možnost filtrování, řazení a stránkování výsledků.
6. Agenda service bude obsahovat nakonfigurovanou cache paměť, do které se budou ukládat často frekventovaná data.

6.2 Architektura integrační vrstvy

Ze seznamu požadavků zmíněné výše lze odvodit dvě primární funkcionality, které musí integrační vrstva splňovat. O plnění jednotlivých bodů se budu dále věnovat v dalších kapitolách.

První funkcionalitou je získání všech dat, která byla dosud dostupná v systému Hodnocení doktorandů. Jedná se hlavně o tabulky a ovládací prvky. Druhou funkcionalitou je získání všech vytvořených úkolů a odeslání jej do systému HUB.FEL.

Vycházíme-li ze zadání, integrační vrstva se musí skládat ze dvou modulů. Pro splnění cílů zadaní, jsem tento požadavek dodržel a rozdělil integrační vrstvu do dvou nezávislých modulů. První modul splňuje požadavky č.1, č.2, č.6 a č.7. Druhý modul následně splňuje požadavky ohledně úkolů a notifikací. Konkrétně tedy požadavky č.3 a č.4. Oba moduly poté splňují požadavek číslo 5. Tyto moduly pracují nezávisle na sobě a v následujících kapitolách se jimi budu zabývat v detailnější měřítku. Nyní bych je rád ve stručnosti představil.

6.3 Modul 1 - DS API

Prvním modul zastřešuje klíčovou funkcionalitu. A to poskytovat všechna potřebná data k tomu, aby v systému HUB.FEL byla možnost tvořit dashboardy, které byly doposud k vidění i v původním systému. Jedná se o 3 dashboardy, které musí být k dispozici.

Tento modul má v sobě vytvořené metody a funkce, které dovolují uživatelům filtrovat, řadit a stránkovat své výsledky. Současně pracuje s cache pamětí (v češtině lze přeložit jako mezipaměť), do které ukládá data pro zrychlení doby odezvy.

Detailnějšímu pohledu modulu 1 je věnována kapitola 7.

6.4 Modul 2 - Task manager

Druhý modul je orientován směrem k práci s úkoly a notifikacemi. Jeho primární činností je zajistit, aby všechny nově vzniklé úkoly byly odeslány do HUB.FEL s minimální odezvou. Modul úkol přijme, zpracuje a odešle dál směrem do Task service. Task manager zároveň bude informovat HUB.FEL o změně stavu úkolu, pokud k němu v původním systému došlo.

Bližší popis modulu 2 je popsán v kapitole 8.

6.5 FURPS analýza

Součástí návrhu byla provedena FURPS analýza pro definování funkčních požadavků a dalších faktorů určující výslednou kvalitu řešení.

1. Functionality - funkční požadavky (F)
   - poskytování interních dat k zobrazení dashboardů. Konkrétně se jedná o dashboardy:
     a. **Termíny semestrálního hodnocení** - přehledová tabulka, kde jsou vypsaný všechny důležité termíny
     b. **Studenti doktorského studia** - tabulka doktorských studentů, u kterých je hodnotitel vedený jako vedoucí práce
     c. **Semestrální hodnocení** - seznam všech semestrálních hodnocení studenta
   - přijímání a odesílání dat o stavu úkolu (vytvořen, splněn, odebrán apod.)
   - poskytování dat o uživateli a jeho právům, které mu bylo přiděleno v systému Hodnocení doktorandů
2. Usability - použitelnost (U)
   - výsledné tabulky (dashboardy) mají stejnou strukturu jako v původním systému Hodnocení doktorandů
3. Reliability - spolehlivost (R)
   - integrační vrstva poskytuje spolehlivá a aktuální data
   - řešení neobsahuje žádné kritické chyby, které by znemožňovaly používání systému
   - poskytuje data pouze ověřeným stranám
4. Performance - výkon (P)
   - doba odezvy integračního řešení nebude svojí implementací nikterak zpomalovat celkovou odezvu mezi systémy HUB.FEL a systémem Hodnocení doktorandů
   - integrační vrstva je schopna obsloužit i desítky paralelních požadavků
5. Supportability - podporovatelnost (S)
   - všechny komponenty integračního řešení a jeho dílčí jednotlivé části jsou zdokumentovány pro ostatní vývojáře (sekvenční diagram, diagram komponent apod.)
   - integrační vrstva je navržena k možnosti úprav nebo rozšířeních o nové funkce

6.6 Vhodné technologie

Nabízí se mnoho různých technologií a programovacích jazyků, které lze na tuto problematiku použít. Po diskuzi s ostatními členy týmu bylo rozhodnuto o použití následujících technologií:

- **Spring Boot** - jako nejvhodnější programovací jazyk jsem se rozhodl použít technologii Spring Boot založenou v jazyce Java. Tento framework je určen pro tvoření webových mikroslužeb a nabízí mnoho zajímavých vylepšení, které usnadní vývoj. Důležité je také poznamenat, že rozhraní FELDS API a jádro systému HUB.FEL je psáno taktéž v jazyce Spring Boot.
- **WebFlux** - verze Sgring Boot 2.0 přináší nově framework Spring WebFlux. Jedná se o rozšíření, které webové mikroslužbě dovoluje možnost asynchronního zpracování požadavku. Ve spojení s datovými proudy (např. Java 8 Stream) může tvořit kompletní reaktivní komponentu. Ve výsledku to znamená rychlejší zpracování a snadnější škálovatelnost. Spring WebFlux zároveň přináší nové datové typy. A to *Mono* a *Flux*.[17]
- **WebClient** - Jelikož data z původního systému budeme volat z již vytvořené API (viz. kapitola 4.4) potřebujeme knihovnu, která dokáže volat vzdálenou API a zpracovat výsledky do Mono, Flux. Jako nejlepší řešení jsem zvolil rozšíření WebClient, které nabízí mnoho metod a funkcí pro volání vzdálené API a následnou konverzi výsledku do Mono nebo Flux.[18]
- **HystrixCommand** - z požadavku číslo 5 (zajištění vyšší stability), bylo rozhodnuto o použití rozšíření HystrixCommand, které je schopno odchytit výpadek jak klientské strany tak i pokud dojde k neočekáváné chybě na integrační vrstvě.[19]
- **Hazelcast** - pro cachování výsledků jsme jako nejlepší řešení stanovil použití knihovny Hazelcast, která přináší velké možnosti konfigurace a následného přizpůsobení mezipaměti.[20]

Kapitola 7
Modul 1 - DS API

Tato kapitola se věnuje detailnímu popisu modulu 1 navrhované integrační vrstvy. Z předchozí kapitoly již víme, že primárním úkolem tohoto modulu je poskytovat všechna data ze systému Hodnocení doktorandů. Z FURPS analýzy (kapitola 6.5) tento návrh odpovídá funkčním požadavkům **1.a** a **1.d**.

7.1 Požadované vlastnosti

Při společné diskuzi došlo ke společnému závěru, že nejvhodnější způsob integrace bude vytvořit REST API. Rozhraní má mít podobu standartního vrstevnatého modelu skládající se z hierarchie Controller -> Service -> DTO -> Client. Detailnějšímu popisu vrstev jsou věnovány další sekce v této kapitole.

Dále bylo rozhodnuto o dodržení společné jmenné konvence koncových bodů, aby se udržela jednotvárnost napříč ostatními integrovanými systémy. Jedná se například o poskytování řazení a filtrování výsledků. Dalším požadovaným bodem bylo vytvoření dokumentace pro možné budoucí úpravy nebo rozšíření.

7.2 Analýza FELDS API

Z kapitoly 5 již víme, že pro přenos dat mezi jádrem systému Hodnocení doktorandů a databází se používá rozhraní FELDS API. Jedná se o klasické REST rozhraní poskytující interní data ze systému Hodnocení doktorandů.

Z poskytnuté dokumentace tohoto rozhraní jsem musel nejdříve vybrat důležité koncové body, podle kterých jsem ve výsledku schopen vytvořit všechny tabulky a dashboardy.

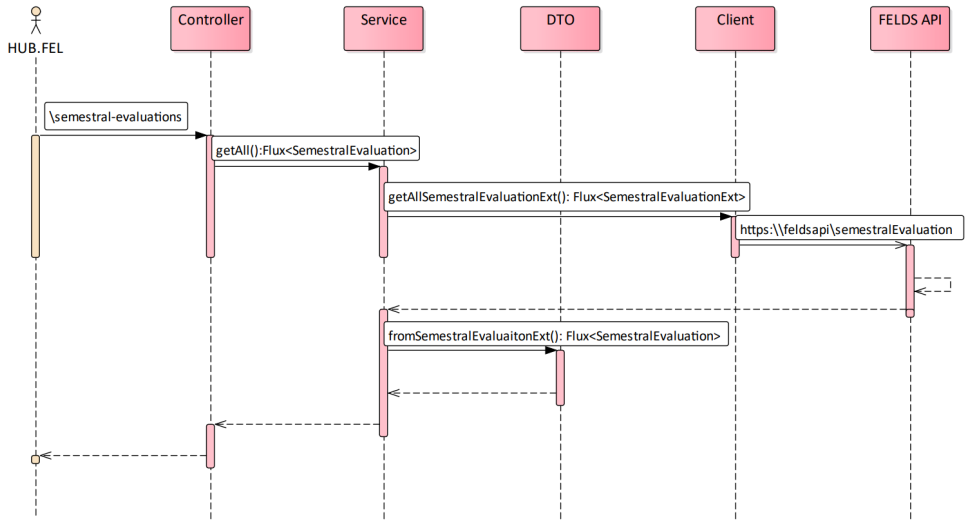
Zároveň jsem se snažil najít vhodný způsob optimalizace. Například pokud FELDS API používá pro získaní termínů semestrálního hodnocení dva koncové body (jeden na seznam termínů a druhý na přeložení kódu příslušného semestru na název), v mém navrhovaném řešení stačí zavolat pouze jeden koncový bod.

7.3 Navrhovaná architektura

Jelikož bude ve výsledku integrační vrstva vedena jako integrační modul systému HUB.FEL je potřeba (stejně jako u ostatních microservices) vytvořit rozhraní na zpracování požadavků. Skrz toto rozhraní budou poté ostatní moduly komunikovat s externím systémem Hodnocení doktorandů - například pro zobrazení dashboardu o doktorandském hodnocení.

7.3.1 Rozložení do vrstev

Jak dokládá sekvenční diagram na obrázku č.7.1, při návrhu jsem se držel standardního postupu při tvorbě vrstevnatého modelu REST API. Jelikož jako zdroj dat je další rozhraní, zvolil jsem nejspodnější vrstvu pojmenovat jako client, jehož úkolem je komunikace s FELDS API.



Obrázek 7.1: Sekvenční diagram vrstevnatého modelu DS API

7.3.2 Controller

První vrstvou je REST Controller, který přijme na koncovém bodu požadavek o data a volá příslušnou metodu vrstvy service. Každý controller pracuje s různou množinou instancí services. Celkeme se jedná o šest vytvořených controllerů, které pokrývají všechna potřebná data k tomu, aby mohly generovat dashboardy na straně HUB.FEL.

7.3.3 Servisní vrstva

Jako další v pořadí vrstevnatého modelu je vrstva service. Ta má na starost mnoho důležitých funkcionalit. První je volání metody clienta na získaní potřebných objektů. Jako další je navázaní na mapování do nových objektů

a následným uložením do cache paměti. Pokud jsou data nalezena v cache paměti jsou data vrácena z ní.

7.3.4 Mapování objektu

Důležitou součástí servisní vrstvy je mapovaní získaných objektů od clienta na nové objekty, které jsem v rámci analýzy upravil tak, aby se limitovaly zbytečné atributy a lépe odpovídaly potřebám HUB.FEL. Po převzetí dat z clienta pomocí metody *map()* se namapují staré objekty na nové. Každý objekt má k sobě vytvořenou svoji Mapper třídu, která má definované metody na přetypování ze starého objektu na nový.

Pokud v rámci mapování je potřeba dalších dat, pomocí metody *zipWith()* zavolám příslušnou metodu clienta a výsledky poté vložím do mapperu.

Jako příklad jsem zvolil ukázku mapování kateder. FELDS API nemá vytvořený koncový bod, který by poskytoval zároveň český a anglický překlad. V mém řešení se tedy nejdříve zavolá client metoda na získaní anglických názvů. Poté se definuje nový požadavek na české názvy a metodou *zipWith()* se výsledky spojí do dvojic. Dle přidělené metody *DepartmentMapper::fromDepartmentPair* se dvojice přetypuje na společnou entitu Department.

```
1 Flux<Department>departmentFlux = departmentClient.getPairs("en")
2     .zipWith(departmentClient.getPairs("cs"),DepartmentMapper::
    fromDepartmentPair);
```

Listing 7.1: Mapování do společné entity Department metodou *zipWith()*

7.3.5 Dotazování klientské strany

Nejspodnější vrstvou je vrstva client. Ta jako jediná navazuje spojení s FELDS API a podle volané metody volá její koncové body.

WebClient

Z předchozí kapitoly víme, že jsem se pro komunikaci s FELDS API rozhodl použít knihovnu WebClient. Po přijetí výsledku z rozhraní převede svoji odpověď na typ Flux nebo Mono. Flux, pokud se jedná o list objektů. Mono, pokud se jedná jen o jeden objekt.

```
1 Flux<StudentOverviewExt> studentOverviewExtFlux =
    webClientBuilder.build()
2     .post()
3     .uri("/v2/semestralEvaluation/studentOverviews")
4     .contentType(MediaType.APPLICATION_JSON)
5     .body(BodyInserters.fromValue(studentOverviewFilter))
6     .retrieve()
7     .bodyToFlux(StudentOverviewExt.class);
```

Listing 7.2: Ukázka vytvoření nového požadavku pomocí WebClient rozhraní

Z ukázky zobrazené výše můžete vidět část z kódu, kde se volá koncový bod o zjištění přehledů studentů. Pomocí metody *build()* se nejdříve vytvoří nová instance. Poté metodou *uri()* je definovaný cílový bod a metodou *body()* je vloženo tělo do požadavku. Nakonec je výsledek převeden na objekt typu *StudentOverviewExt* a celý výsledek poté do typu Flux.

Hystrix fallback

Důležitou roli zajišťující stabilitu je hystrix fallback. Ten má na starost sledovat celý průběh získávání dat od clienta až po předání controlleru. Zároveň je také schopna vyhodit chybovou hlášku, pokud je FELDS API nedostupné nebo její odpověď trvá příliš dlouho.

Kapitola 8

Modul 2 - Task manager

Nyní si blíže přiblížíme druhou část navrhovaného řešení. A to modul sloužící pro správu úkolů a notifikací vzniklých v rámci procesu hodnocení doktorandů. Tato kapitola se zabývá analýzou aktuálního stavu a následným představením navrhovaného řešení. Z FURPS analýzy (kapitola 6.5) tento návrh odpovídá funkčním požadavkům **1.b**, **1.c** a **1.e**.

8.1 Požadované vlastnosti

Stejně jako u předchozího modulu, i zde vzniklo několik požadavků, které výsledné řešení musí splňovat. První požadovanou vlastností je co nejkratší doba prodlevy. Optimálně, pokud to integrovaný systém dovolí, je schopnost zachytit vzniklou událost a v tomtéž okamžiku informovat o události integrační vrstvu. Ta informaci zpracuje a dál přepošle do systému HUB.FEL.

Dalším požadavkem je vytvoření unikátního identifikátoru. Každý úkol a notifikace musí mít definovaný svůj vlastní identifikátor, pod kterým bude uložen do systému. V rámci schůzek bylo rozhodnuto, že každý integrovaný systém si identifikátory bude tvořit podle předpisu **SYSTEM_ID_OBJECT_ID**. SYSTEM_ID představuje id systému, odkud úkol nebo notifikace pochází. OBJECT_ID poté označuje interní identifikátor objektu.

Následně bude tento vzorec zašifrován pomocí hashovací funkce pro udržení konstantní délky. Touto metodou se zaručí, že dva odlišné systémy nebudou generovat stejné identifikátory.

8.2 Task service

Systém HUB.FEL má pro úkoly a notifikace dedikovanou vlastní mikroslužbu, pojmenovanou jako Task service (viz kapitola 5.5). Ta pro integrované systémy představuje vstupní bod, se kterým musí komunikovat. Jedná se o menší REST rozhraní. Kromě přidání nového úkolu a notifikace disponuje rozhraní i koncovými body na označení úkolu a notifikace jako splněné. Dále nabízí pro obě entity operaci delete k jejich odebrání.

Pro zajištění jednotného rozhraní HUB.FEL bylo rozhodnuto, že Task service bude mít definovaný obecný model úkolu a notifikace, kterými se

budou ostatní systémy řídit. Každý systém před odesláním dat bude muset provést mapování na kompatibilní podobu a následně až poté úkol nebo notifikaci odeslat. Na straně Task service poté dochází ještě k validaci, zda schéma splňuje všechny požadované atributy. Odpovědí rozhraní je příslušný http stavový kód.

8.3 Standartizace zpráv

Jak bylo zmíněno výše, všechny vzniklé úkoly a notifikace musejí být před odesláním nejdříve namapovány na novou entitu. V rámci analýzy vznikly obecné modely úkolu a notifikace, které obsahují všechny důležité atributy. Společná entita úkolu vypadá následovně:

```
{
 "aid": "5f4dcc3b5aa765d61d8327deb882cf99",
 "assignees": [{
               "username": "string",
               "read":"boolean"
               }],
 "origin": "A name or id of an origin system.",
 "content": {
   "title": "Fill form",
   "description": "",
   "deadline": "2020-09-17T19:49:27",
   "solve": "valid URL solve link",
   "solveType": "EXTERNAL_FORM ",
   "created": "2020-09-17T19:49:27",
   "finished": "2020-09-07T07:02:01",
   "notification": false,
   "creator": "Some unique user id. Blank if not created by
    user.",
   "assets": [{
       "title": "Fig 2",
       "link": "https://www.kometa.fel.cvut.cz",
       "type": "LINK"
     }]
 }
}
```

Listing 8.1: Struktura společné entity úkolu

Atribut *aid* představuje unikátní identifikátor, který byl vytvořen z předpisu pro generování. Důvody a způsob generování popisuji v kapitole 8.1.

Důležitou hodnotu obsahuje parametr *solveType*. Ten definuje, zda je formulář ke splnění úkolu generován přímo v systému HUB.FEL nebo kvůli své složitosti je potřeba uživatele přesměrovat do původního systému. Pokud atribut obsahuje hodnotu *EXTERNAL_FORM* dává tím externí systéme najevo, že splnění úkolu bude probíhat na jeho straně. Součástí takového objektu se poté očekává i validní odkaz na jeho splnění v atributu *solve*.

Kvůli komplexnosti systému Hodnocení doktorandů a platformy IBM BPM bylo stanoveno, že všechny úkoly budou přesměrovány do externího systému, odkud je uživatel může vypracovat.

8.4 Navrhované řešení

Nyní popíšu navrhované řešení, které řeší problematiku integrace úkolů. Stejný postup je navržen i pro práci s notifikacemi.

8.4.1 Zachycení události

Framework IBM BPM, které tvoří jádro systému Hodnocení doktorandů, disponuje rozšířením Dynamic Event Framework. Jedná se o nástroj, který přináší schopnost zachytit veškeré události, které v rámci procesu vznikají. Data z události pak můžou být dále zpracována a odeslána na cílovou destinaci. Jedná se například o události splnění nebo vytvoření nového úkolu.

Pro naše účely modulu 2 je toto rozšíření plně dostatečné. Po zachycení dané události dojde ke sběru dat, která jsou následně odeslána k integrační vrstvě. Ta má pro tyto účely definovaný koncový bod, kde očekává data ve smluvené podobě. Reálná ukázka domluveného schématu úkolu vypadá následovně:

```
{
  "assignees": ["loffldav","stepaji9"],
 "created": "2021/03/02 01:11:34.561 CET",
  "deadline": "2021/03/22 22:59:00.0 CET",
  "origin": "FEL_DSE",
  "taskId": "6001",
  "title": "Review doctoral study",
  "type": "CREATED",
  "url": "https://bpmpc.feld.cvut.cz/shibredir/teamworks/process
    .lsw?zWorkflowState=1&zTaskId=59471"
}
```

Listing 8.2: Struktura domluveného schématu BPM úkolu

Tento model v sobě obsahuje všechna potřebná data, která pro vytvoření obecného modelu potřebujeme. Atribut *url* v sobě skrývá odkaz na vykonání daného úkolu.

8.4.2 Mapovaní na společnou entitu

Po přijetí dat proběhne mapování na společnou entitu HUB.FEL úkolu. Podle přijatého atributu *type* integrační vrstva rozhodne, jaké metody dále zavolat. Atribut *type* vychází z životního cyklu úkolu (viz kapitola 4.2) a může nabývat těchto hodnot:

- CREATED - byl vytvořen úkol
- COMPLETED - úkol byl splněn
- CLAIMED - uživatel si připnul úkol (platí pouze pro skupinové úkoly)
- REASSIGNED - uživatel zrušil připnutí úkolu
- EXPIRED - čas pro splnění úkolu vypršel

V případě možnosti CLAIMED je nejdříve odeslán požadavek na Task service na odebrání týmové úkolu. Po přijetí zprávy o smazaní je vytvořen nový individuální úkol pro uživatele, který je v atributu *assignees*. V případě typu REASSIGNED je postup obrácený. Nejdříve je odebrán individuální úkol a poté je vytvořen úkol týmový.

Následně je provedeno mapování na společnou entitu. Jelikož bude konverze potřeba i u jiných integrovaných systémů, rozhodl jsem se navrhnout část modulu tak, aby byla použitelná jako knihovna pro ostatní integrační řešení. Knihovna kromě nových entit nabízí i připravené metody pro komunikaci s Task service.

8.4.3 Přesměrování na vykonání úkolu

Poté co úkol prošel upravujícím procesem a byl odeslán směrem do Task service, je zobrazen příslušnému uživateli na hlavním dashboardu. Uživatel společně s informacemi o úkolu má k dispozici i odkaz, který ho přesměruje do původního systému, kde daný úkol může splnit.

Vize systému HUB.FEL je taková, že přihlašování bude probíhat skrz stejnou SSO bránu, jaká se používá v rámci celé katedry FEL. Uživatel tedy při přesměrování nemusí již znovu proces přihlášení absolvovat. Systém si zkontroluje, zda má uživatel aktivní relaci a pokud ano, je automaticky přesměrován na splnění úkolu.

8.4.4 Označení úkolu jako splněný

Jakmile uživatel odešle formulář a systém úkol vyhodnotí jako splněný, vytvoří se událost o splnění úkolu. Dynamic event framework tuto událost odchytí a odešle jej směrem na integrační vrstvu. Atribut type v tomto případě nabývá hodnoty COMPLETED. Integrační vrstva z přijatých dat vytvoří požadavek na označení úkolu jako splněný.

Část IV

Implementace, testování a nasazení

Kapitola 9
Implementace řešení

Při implementaci řešení jsem postupoval podle návrhu, který byl představen v předchozích kapitolách. Není tedy třeba ho v této kapitole znovu připomínat.

Místo toho jsem se rozhodl věnovat tuto kapitolu krátkým ukázkám implementace integračního řešení. Jednotlivé ukázky vycházejí z požadovaných vlastností na implementaci (kapitola 6.1).

Integrační vrstva byla implementovaná v jazyce Spring boot. Důvody použití tohoto jazyka popisuji v kapitole 6.6.

9.1 Mezipamět

Požadavek č. 7 zní: "*Agenda service má nakonfigurovanou cache paměť pro často frekventovaná data*". Z analýzy systému Hodnocení doktorandů vyplývá, že v určitých částech procesu hodnocení může docházet k rychlým aktualizacím dat. To má za následek jejich následnou neaktuálnost, pokud jsou data uložena na integrační vrstvě v cache paměti. Není tedy možné data uchovávat příliš dlouho.

9.1.1 Způsob ukládání

V rámci společných debat s ostatními bylo rozhodnuto, že data budou v paměti uchovávána po krátkou dobu (přibližně 5 minut), poté se cache paměť automaticky vyprázdní.

Po získání odpovědi z FELDS API, jsou data namapována podle upraveného schématu a jsou vložena do cache paměti. Při následném vzniku stejného požadavku se integrační vrstva podívá nejdříve do cache paměti a pokud požadovaná data nalezne, vrátí je. Data jsou do cache paměti ukládány podle unikátního identifikátoru (např. id semestrálního hodnocení)

9.1.2 Konfigurace

Pro implementaci cache paměti jsem se rozhodl použít knihovnu Hazelcast. Toto rozšíření jsem zmiňoval v kapitole 6.6, kde jsem popisoval její přednost bohaté možnosti konfigurace.

Pro znázornění konfigurace slouží ukázka kódu níže:

```
<map name="evaluation-deadlines">
    <time-to-live-seconds>300</time-to-live-seconds>
    <in-memory-format>BINARY</in-memory-format>
    <merge-policy>
        com.hazelcast.map.merge.LatestAccessMergePolicy
    </merge-policy>
    <cache-deserialized-values>
        INDEX-ONLY
    </cache-deserialized-values>
</map>
```

Listing 9.1: Ukázka konfigurace mezipaměti

Každá entita určená k ukládání do mezipaměti má vytvořenou svoji cache pamět pomocí tagu *<map>*. Ta se může různě nastavovat pomocí dalších tagů a atributů.

V tomto případě se jedná konkrétně o konfiguraci objektu termínů hodnocení a skládá se z těchto atributů:

- *<time-to-live-seconds>* - počet sekund udávající, jak dlouho mají být data v cache paměti uložena [21]
- *<in-memory-format>* - forma ukládání dat do paměti[22]
- *<merge-policy>* - metoda slučování cachovaných dat[23]
- *<cache-deserialized-values>* - způsob ukládání deserializovaných hodnot[24]

Na stejném principu mají svoji cache paměť definované i ostatní entity jako například seznam studentských oborů, přehled studentských hodnocení nebo seznam všech kateder.

V praxi můžeme pozorovat značné zrychlení celého procesu získávání dat. Většinou se jedná o zrychlení v jednotkách stovek milisekund, přibližně o 400 - 700ms podle velikosti dat. V případě, že má cache paměť data k dispozici, se poté doba odezvy pohybuje kolem 5 - 80ms.

9.2 Filtrování výsledků

Mezi dalšími požadavky na implementaci patří i možnost filtrování výsledků. Rozhraní FELDS API tuto vlastnost neposkytuje, bylo tedy potřeba vyřešit tuto funkcionalitu na straně integrační vrstvy.

9.2.1 Řazení

Řazení výsledků je provedeno pomocí hashovací mapy, kde jako klíč slouží identifikátor řazení (bySemesterCode) a odpovídající hodnota klíče je poté generický typ *Comparator<>*, která ve svoji přetížené metodě *compare(object1,object2)* definuje, podle jakých atributů se mají objekty mezi sebou porovnat.

Ukázka níže popisuje konkrétní definici hashovací mapy pro řazení objektů semestrálních hodnocení.

```
private Map<String,Comparator<SemestralEvaluation>>
    comparatorMap = new HashMap<>();
private Comparator<SemestralEvaluation> bySemesterCode = new
    Comparator<SemestralEvaluation>() {
    @Override
    public int compare(SemestralEvaluation s1,
    SemestralEvaluation s2) {
        return s1.getSemester().getCode().compareTo(s2.
    getSemester().getCode());
    }
};
```

Listing 9.2: Ukázka řazení semestrálních hodnocení podle kódu semestru

Po přijetí dat z FELDS API se podle klíče zjistí odpovídající komparátor a pomocí metody *sort(comparator)* se data seřadí.

Vykonavatel požadavku má rovněž k dispozici zvolit si směr řazení, čili sestupně nebo vzestupně. Pokud u požadavku nalezne integrační vrstva směr řazení "DESC"je na komparátor zavolána metoda *reversed();*, která způsob řazení otočí.

9.2.2 Filtrování

Podobný způsob jako u řazení se používá i u filtrování. Podle typu objektu jsou vymezeny atributy, podle který lze filtrovat.

Ukázka níže zobrazuje konkrétní příklad definice filtru pro semestrální hodnocení.

```
List<Predicate<SemestralEvaluation>> filterBy = new ArrayList
    <>();

Predicate<SemestralEvaluation> filterStudentLogin = s ->
    studentLogin.contains(s.getLogin());
Predicate<SemestralEvaluation> filterDepartmentCode = s ->
    departmentCode.contains(s.getDepartmentCode());
```

Listing 9.3: Deklarace filtrů semestrálních hodnocení

Z ukázky si lze povšimnout, že pro realizaci filtru jsem použil rozhraní typu Predikát, ve kterém je pomocí lambda výrazu definována podmínka pro splnění.

Po přijetí požadavku se integrační vrstva podívá do parametru požadavku, podle jakých filtrů se budou výsledky filtrovat. Odpovídající filtr je poté vložen do listu *filterBy*.

Následnou aplikaci filtrů zobrazuje ukázka níže:

```
semestralEvaluations.filter(filterBy.stream().reduce(x->true,
    Predicate::and));
```

Listing 9.4: Aplikování filtru na seznam výsledků

Zavoláním metody *filter()* na list prvků má za následek, že dochází postupně k parametrizace podle vstupních hodnot požadavku. Funkce *filterBy.stream().reduce(x->true, Predicate::and)* poté způsobuje, že výstupem může být jen výsledek, který vyhovuje pouze všem filtrům.

9.2.3 Stránkování

Po řazení a filtrování přichází poslední úprava výstupu. Tím je stránkování. K provedení jsou potřeba dva parametry - limit a odsazení (anglicky offset). Pomocí těchto dvou čísel jsme schopni přesně vymezit horní a dolní mez pro výstup výsledků.

Celou problematiku stránkování stěžuje fakt, že integrační vrstva používá reaktivní datové typy Mono a Flux. Není tedy možné použít metodu *sublist()* jako u klasického ArrayListu.

Ukázka níže zobrazuje způsob stránkování výsledků pomocí metody *distinct()*, ve které místo odlišnosti posuzujeme, zda je výsledek v limitu nebo ne.

```
flux.distinct(fluxItem -> 0,
    () -> new FluxCounter(offset, limit),
    (counter, key) -> counter.next(),
    new Consumer<FluxCounter>() {
        @Override
        public void accept(FluxCounter fluxCounter) {}
    }
);
```

Listing 9.5: Ukázka stránkování výsledků pomocí metody distinct()

Přejaté hodnoty limit a offset nastaví pomyslnou dolní a horní mez, ve kterých se výsledky musejí nacházet. Metoda *counter.next()* vrací true/false podle toho, zda se výsledek nachází ještě v limitu. Přetížená metoda *accept()* společně s atributem *fluxItem* zde nemají žádný smysl, jsou zde pouze z důvodu předpisu funkce.

9.3 Bezpečnost

Bezpečnost celého implementovaného řešení je v režii systému HUB.FEL pod kterou byla mikroslužba nasazena. Ta pro zabezpečení má v plánu používat ověřování přes OAuth2 server, který v době psaní této práce není spuštěn.

Rozhraní FELDS API, které používám jako zdroj dat Modulu 1, má nastavenou autentifikaci pomocí basic authentification. Při každém volání se pomocí dvojice jméno-heslo musí integrační vrstva nejdříve autentizovat. Stejnou metodiku používá i IBM BPM REST rozhraní. Oboje přihlašovací údaje se nacházejí v konfigurační souboru application.properties. Odtud jsou při startu mikroslužby vyjmuty a pomocí anotace @Value vloženy do proměnné. Následně jsou přihlašovací údaje při každém požadavku vloženy do hlavičky požadavku.

Pro zabezpečení Modulu 2 proti útokům jsem zvolil standartní postup pomocí basic-authentification. Cílem je omezit volání koncového bodu jen pro oprávněné subjekty. Integrační vrstva má vytvořeného uživatele, kterému jsem náhodně vygeneroval posloupnost znaků jako heslo. Jméno s heslem je nastavitelné v konfiguračním souboru application.properties. Systém Hodnocení doktorandů se tedy při každém požadavku na modul 2 musí nejdříve autentizovat pomocí uživatelského jména a hesla. Konkrétně se jedná o koncové body

/tasks a */notifications*. Pokud jsou údaje správné, může být požadavek na koncový bod odeslán. Není možné odesílat úkoly a notifikace bez správných přihlašovacích údajů.

9.4 WhoAmI požadavek

Jednou z klíčových vlastností integrační služby je poskytování práv o uživateli. Po úspěšném přihlášení do systému HUB.FEL je vyslán požadavek WhoAmI napříč všemi integrovanými systémy. Cílem požadavku je zjistit, ke kterým datům a dashboardům má uživatel v externím systému přístup.

Pro tyto účely má rozhraní IBM BPM vyhrazený koncový bod, který podle vloženého uživatelského jména vrací seznam všech dashboardů, které si uživatel může zobrazit. Zároveň vrací i plno dalších dat, jako je například název, id dashboardu nebo název skupiny, do které dashboard spadá.

Na pokyn od architekta systému Hodnocení doktorandů jsem do modelu doplnil ještě validní URL odkaz, který uživatele přesměruje na konkrétní dashboard. Toto řešení je zcela dočasné a slouží pro první fázi spuštění systému HUB.FEL.

Stejně jako u mapování úkolů a notifikací na jednotný model, i zde platí udržet co nejvíce jednotné schéma napříč všemi externími systémy. Před odesláním dat je tedy potřeba přeměnit data do kompatibilní podoby pro HUB.FEL. Schéma vypadá následovně.

```
dashboardPermission{
    "read": true
    "title": "Dashboard Display Title"
    "readLink": "https://validDashboardURL"
}
```

Listing 9.6: Struktura dashboardPermission objektu

Pokud má uživatel právo na zobrazení dashboardu, je mu vytvořen výše zobrazený objekt. Z předchozích kapitol již víme, že systém Hodnocení doktorandů používá 3 dashboardy - Semestrální hodnocení, Termíny semestrálních hodnocení, Studenti doktorského studia.

V případě, že má uživatel přístup do všech dashboardů je mu vytvořen objekt *semestralEvaluationPermission*, *studentsofDoctoralStudyPermission* a *semestralEvaluationsDeadlinesPermission*.

9.5 Dokumentace rozhraní

V rámci systému HUB.FEL vznikl soubor všech dokumentací mikroslužeb, které jsou aktuálně zapojeny do architektury systému.

Pro ulehčení práce ostatním vývojářům, kteří budou moje implementované řešení dále používat, jsem vytvořil dokumentaci mého integračního řešení. Použil jsem na to nástroj Swagger, který byl pro tyto účely vytvořen a je široce rozšířen. Konkrétně jsem použil variantu pro Spring Boot, která sama namapuje všechny koncové body a automaticky dokumentaci vygeneruje.

Součástí dokumentace jsou i všechna schémata objektů, které mikroslužba používá.

Výslednou dokumentaci lze poté nalézt na zvolené URL adrese (viz kapitola 11 Nasazení). Dokumentaci je možné dále upravovat pomocí definovaných anotací. Následující příklad popisuje ukázku definici možných odpovědí koncového bodu */study-branches*.

```
@ApiResponses(value = {
    @ApiResponse(responseCode = "200", description = "OK"),
    @ApiResponse(responseCode = "400", description = "Bad input
    parameter", content = @Content)
})
public Flux<StudyBranch> getAll(){ // Function content }
```

Listing 9.7: Použití anotace @ApiResponses pro vygenerování dokumentace rozhraní

Tímto způsobem je zaručena konzistence dat, kdy dokumentace vždy popisuje aktuální stav rozhraní. Při budoucích úpravách není potřeba udržovat dokumentaci s reálným stavem mikroslužby.

Kapitola 10
Testování

Po úspěšné implementaci přichází na řadu integrační řešení otestovat. Tato kapitola popisuje postup vytvoření testovacích scénářů, způsobu testování a jejich následné vyhodnocení.

10.1 Způsob testování

Pro správné otestování implementované řešení je důležité vybrat korektní metodiku testů. Jelikož se jedná o integrační vrstvu mezi dvěma systémy, uživatelské testy nejsou relevantní. Nabízí se použití jednotkových testů, kde postupně otestujeme metody a funkce a budeme následně porovnávat s očekáváným výsledkem.[25]

Jako další způsob testování se nabízejí integrační testy. Tento způsob testů se používá při testování interakce mezi jednotlivými částmi aplikace.[27] V mém případě integračních testy mají za úkol otestovat komunikaci s rozhraním FELDS API a otestovat korektní přijetí dat. Framework Spring Boot nabízí pro implementaci těchto testů několik anotací, které usnadní jejich tvorbu.

Současně se zde nabízí použít metodiku End-to-End testů. Úkolem takového testu je projít aplikaci od začátku do konce a simulovat tak reálné budoucí použití aplikace v praxi.[26]

V případě testování integrační vrstvy, rozhodl jsem se nejprve pomocí jednotkových testů otestovat všechny důležité metody. Pro snadnější tvorbu a vyhodnocení testů jsem použil framework JUnit, který je ve frameworku Spring Boot velmi rozšířený.[25]

Co se týče End-to-end testů, zde jsem jako nejvhodnější nástroj pro otestování zvolil klienta Postman. Ten nabízí možnost otestovat koncový bod REST rozhraní. Definují se vstupní parametry a očekávaný výstup. Pokud očekávaný výstup odpovídá reálnému výstupu je test vyhodnocen jako úspěšný. Velkým přínosem je i možnost měření doby odezvy, kdy podle výsledného času je test vyhodnocen jako úspěšný nebo neúspěšný.

10.2 Testovací scénáře

První fází testování bylo otestovat pomocí JUnit testů všechny důležité metody a procedury z Modulu 1 a z Modulu 2. Tyto testy jsou součástí zdrojového kódu a při budoucích úpravách je lze spustit znovu.

Hlavním cílem jednotkových testů bylo pokrýt všechny funkce, které mají na starost správné mapování na nový typ objektu. V každém testu nejdříve definuji očekávaný vstup (odpověď z FELDS API) a následně provedu mapovací funkci. Pomocí assert metod poté porovnávám, zda se atributy namapovaly správně.

Integračními testy testuji koncové body FELDS API rozhraní. Jejich úkolem je primárně ověřit, že volaný koncový bod je dostupný a vrací správný typ objektu. Zároveň pomocí integračních testů testuji koncový bod IBM BPM REST API, který používám při zjišťování whoAmI požadavku.

Pro End-to-End testování jsem vytvořil sbírku testovacích scénářů. Scénáře pokrývají všechny koncové body, které integrační řešení nabízí. Každému testu jsem přidal kombinaci různých vstupních parametrů (např. filtrování nebo řazení) a očekávaný výstup.

Na základě dokumentace rozhraní testy vyhodnocovaly, zda odpověď integrační vrstvy odpovídá schématu definované v dokumentaci. Dále pak, zda odpověď vrací korektní stavový kód. Všechny end-to-end testy jsou přiloženy v elektronické příloze této práce.

Ukázka níže zobrazuje konkrétní testovací scénář pro otestování koncového bodu vracející přehledy doktorských studentů. Hlavním úkolem testu bylo ověřit, zda vyplněný filtr vrací korektní hodnoty. V tomto případě filtr obsahoval informaci, že chceme studenty pouze z jednoho studijního oboru:

```
/*Test status code 200 */
pm.test("Status code is 200", () => {
  pm.expect(pm.response.code).to.eql(200);
});

/*Test correct filtering method*/
pm.test("Correct studyBranchCode", () => {
    const studentOverviews = pm.response.json();
    for (var i = 0;i<studentOverviews.length;i++) {
        var studentOverview = studentOverviews[i];
        pm.expect(studentOverview.branchProgrammeCode == "1701
    V011");
    }
});
```

Listing 10.1: Ukázka testovacího scénáře pro end-to-end testy

End-to-End testování Modulu 2 bylo provedeno na stejný způsob jako u Modulu 1. I zde vzniklo několik testovacích scénářů simulující budoucí reálné použití integrační vrstvy v praxi (tedy přidání úkolu do fronty v systému HUB.FEL). Po odeslání se kontroloval přijatý status kód včetně těla odpovědi, zda odpovídá dokumentaci rozhraní Task-service. V rámci testování vzniklo i několik testů, kdy vědomě na rozhraní Task Service odesílám nevalidní data a očekávám na výstupu chybovou hlášku.

Ukázku JUnit testu mapující data na objekt HubTask popisuje níže zobrazený kus kódu. Nasimulovaná vstupní data (BPMTask) jsou skutečnou podobou reálných dat přijatých ze systému Hodnocení doktorandů.

```
@Test
public void taskBuilderTest() {
    BPMTask bpmTask = mockCreatedBPMTask();
    HubTask hubTask = taskService.createNewHubtask(bpmTask);

    assertThat(hubTask).isNotNull();
    assertThat(hubTask.getContent()).isNotNull();
    /*test correct mapping*/
    assertEquals(bpmTask.getAssignees().size(), hubTask.
    getAssignees().size());
    assertEquals(bpmTask.getAssignees().get(0), hubTask.
    getAssignees().get(0).getUsername());

    assertEquals(bpmTask.getOrigin(), hubTask.getOrigin());
    /* other assertEquals() methods */
}
```

Listing 10.2: Ukázka JUnit testu

10.3 Vyhodnocení testovacích scénářů

Celkově bylo vytvořeno 35 end-to-end testovacích scénářů a 28 jednotkových testů. End-to-end testy v sobě obsahují přibližně 90 test assertů. Integračních testů bylo poté vytvořeno přesně 10.

Jednotkové testy odhalily drobné chyby v implementaci jako například mapování proměnných s null hodnotou. Dále byla častou chybou špatná konvence proměnných. Jedna z větších chyb byla ukryta u filtrování a řazení, kdy integrační vrstva vracela chybové hlášky při řazení s null hodnotami.

Cílem testování bylo ověřit všechny funkční a nefunkční požadavky, zda splňují faktory definované ve FURPS analýze v kapitole 6.5. Pokud test odhalil chybu, byla následně v kódu chyba opravena. Každý JUnit test měl za úkol ověřit konkrétní funkcionalitu integrační vrstvy. End-to-End testy poté představovaly reálné procházení integrační vrstvou za ostrého provozu. Úkolem integračních testů bylo ověři správné přijetí a komunikaci s rozhraním FELDS API.

Kapitola 11

Nasazení

Po úspěšném otestování zbývá celé integrační řešení nasadit a spustit do testovacího provozu. Tato kapitola pojednává o nutných krocích vedoucí k úspěšnému zapojení mikroslužby do systému HUB.FEL.

11.1 Zapojení do HUB.FEL

Z předchozích kapitol víme, že pro architekturu systému HUB.FEL byla zvolena architektura mikroslužeb. Pro jejich snadnější zapojení do systémové infrastruktury byla vytvořena knihovna *LibCommon*. Tato knihovna nastaví konfiguraci a připojení k service register. Ta má na starost registrovat a vracet instance konkrétní mikroslužby. Detailnímu popisu jednotlivých mikroslužeb systému HUB.FEL se věnuji v kapitole 5.

Použití knihovny je velmi jednoduché. Do hlavní třídy obsahující main metodu stačí přidat dědičnost z abstraktní třídy *AbstractFelHubApplication*. Primární činností abstraktní třídy je zaregistrovat modulu do registru mikroslužeb, která je implementovaná technologií Netflix eureka. K hlavní třídě je ještě potřeba přidat anotaci @EnableDiscoveryClient. Tímto krokem se bude mikroslužba registrovat jako nová Agenda service (viz. kapitola 5).

```
1 @EnableDiscoveryClient
2 @SpringBootApplication
3 @EnableCircuitBreaker
4 @EnableCaching
5 public class DsmicroserviceApplication extends
      AbstractFelHubApplication {
6     public static void main(String[] args) {
7         SpringApplication.run(DsmicroserviceApplication.class,
      args);
8     }
9 }
```

Listing 11.1: Použití knihovny LibCommon

Aby se předešlo fatálním chybám, bylo potřeba před implementací knihovny zrevidovat stejné číslo verze u dependency závislostí. Jedná se například o použití stejné verze frameworku Spring Boot apod.

11.2 Přípravy před nasazením

Po úspěšném přidání LibCommon knihovny bylo na řadě mikroslužbu nahrát do fakultního GitLab repozitáře. Centrum znalostního managementu má pro systém HUB.FEL a jeho části vytvořenou skupinu repozitářů. Tam lze vytvořit různé větvě pro různé verze mikroslužby (např. produkční/testovací větev).

Před nahráním na fakultní repozitář jsem provedl ještě revizi a údržbu zdrojového kódu. Cílem bylo odebrat všechny přebytečné kusy kódu. Například nepoužívané importy tříd nebo metody, které jsem v průběhu vývoje zakomentoval z důvodu použití jiných metod.

11.3 Spuštění do provozu

Z repozitáře lze integrační řešení nasadit na server. Odtud je přístupné v rámci síťové infrastruktury CZM, stejně jako systém HUB.FEL. Integrační řešení běží konkrétně na adrese https://dsefel.hub.czm.fel.cvut.cz. Aktuální dokumentace je poté přístupná z adresy https://dsefel.hub.czm.fel.cvut.cz/swagger-ui.html.

Před nasazení, při tvorbě aplikačního souboru, proběhly veškeré JUnit a integrační testy. Po nasazení jsem jako poslední krok spustil všechny end-to-end testy z kapitoly 10. Všechny testy proběhly v pořádku.

Tímto bylo integrační řešení mezi systémem Hodnocení doktorandů a systémem HUB.FEL uvedeno do provozu.

Kapitola 12

Závěr

Závěrem bych rád shrnul všechny důležité body, kterými se tato práce zabývala. Začátek práce byl věnován teoretické části, a to definici informačního systému a jeho přínosům v podnicích a v organizacích. Dále jsem se zaměřil na systémovou integraci, jaké jsou její druhy a klíčové benefity, které s sebou integrace přináší. Z citovaného průzkumu na téma Faktory určující kvalitu IS vyšlo najevo, že velký vliv na výslednou kvalitu je přisuzován snadnému ovládání a přesnosti získané informace. Přibližně 25% dopad na výslednou kvalitu je poté připisován systémové integraci. Ze získaných výsledků tohoto průzkumu jsem dospěl k závěru, že systémová integrace dává smysl.

V další části práce jsem se věnoval detailnímu rozboru systému Hodnocení doktorandů a systému HUB.FEL. U analýzy systému Hodnocení doktorandů jsem blíže prozkoumal platformu IBM BPM. Výsledkem bylo zjištění, že technologie obsahuje REST rozhraní pojmenované jako REST IBM BPM. To slouží pro poskytování dat o systémových datech jako například práva uživatele a seznam jeho dostupných dashboardů. Z analýzy architektury bylo zjištěno, že systém používá rozhraní FELDS API pro čtení a zápis dat do databáze.

Analýza systému HUB.FEL byla pojata ve stejném měřítku. Kromě rozboru architektury jsem popsal vizi a cíle projektu. Z vize vyplývá, že hlavním cílem je zjednodušit interakci uživatelů s fakultními systémy, kterých se v současné době používá zhruba kolem 20. Tomu odpovídá i architektura systému, která je strukturována do podoby mikroslužeb. Důvodem je snadná flexibilita a škálovatelnost pro další rozšíření.

V další části práce jsem použil výsledky z analýzy jako podklad pro návrh integrační vrstvy. Na základě požadavků zadání jsem architekturu rozdělil do dvou oddělených modulů. První modul představuje REST rozhraní a druhý modul slouží jako správce úkolů a notifikací. Pro zjištění funkčních a nefunkčních požadavků integrační vrstvy jsem vytvořil FURPS analýzu, kterou jsem následně využil jako validátor pokrytí v části testování. Součástí byl i rozbor technologií, které jsou vhodné pro použití v implementaci. Rozhodl jsem se použít framework Spring boot společně ještě s dalšími knihovny jako například Hystrix fallback nebo Hazelcast. Splněním těchto bodů jsem splnil cíl práce č.1.

V detailním pohledu jsem poté představil návrh pro modul 1. To představuje

typické REST rozhraní rozložené do vrstevnatého modelu, kde jako zdroj dat používá rozhraní FELDS API pro interní data a pro systémová data rozhraní REST IBM BPM. Rozhraní je navíc rozšířeno o funkce filtrování a stránkování výsledků. Tím byly splněny cíle práce č.2 a č.3.

Modul číslo 2 poté představuje správce úkolů a notifikací. Po zachycení události v systému Hodnocení doktorandů je proveden sběr dat a data jsou následně odeslána směrem k integrační vrstvě. Ta data přijme a provede požadované operace jako je například generování unikátního identifikátoru. Podle typu přijaté události provede integrační vrstva příslušné akce směrem k mikroslužbě Task service. Tím byl splněn cíl práce č.4.

Důležitou činností integrační vrstvy je mapování na jednotné schéma úkolu a notifikace. Jelikož tuto konverzi budou nuceni řešit i ostatní systémy plánované k integraci, rozhodl jsem se pro tyto účely vytvořit knihovnu. Tu jsem dále poskytnul CZM.

Integrační vrstvu jsem podle návrhu implementoval do funkčního řešení. Následně jsem vytvořil testovací scénáře pro end-to-end testy a integrační řešení otestoval. Kromě end-to-end testů jsem vytvořil ještě jednotkové a integrační testy, které testují metody a funkce integrační vrstvy. Všechny nalezené chyby jsem opravil. Tím byly splněny cíle práce č.5.

Do otestovaného řešení jsem následně implementoval knihovnu pro registraci k ostatním mikroslužbám systému HUB.FEL. Tím jsem zajistil, že mikroslužba je připojena jako součást systému HUB.FEL. Finální verzi mikroslužby jsem nahrál na fakultní repozitář a nasadil do provozu. Mikroslužba je dostupná na adrese https://dsefel.hub.czm.fel.cvut.cz (potřeba být připojen k CZM VPN). Tím jsem splnil poslední cíl práce č.6.

Splněním těchto cílů jsem splnil všechny pokyny, které mi byly v rámci bakalářské práci zadány. Výsledek práce bude dále používán jako součást systému HUB.FEL, který je plánován ke spuštění v druhé polovině roku 2021.

Literatura

[1] VALACICH, Joseph a Christoph SCHNEIDER. *Information Systems Today – Managing in the Digital World.* 4th edition. Upper Saddle River, New Jersey: Prentice-Hall, 2010. ISBN 9780136078401.

[2] NÁPLAVA, Pavel. *Úvod do předmětu a problematiky Informačních systémů* [přednáška]. Praha: České vysoké učení technické v Praze, 19.2.2020.

[3] THONG, James Y. L, CHEE-SING, YAP a K. S. RAMAN, STOWELL, Frank A., Daune WEST a James G HOWELL, ed. User Satisfaction as a Measure of Information System Effectiveness [online]. Boston, MA: Springer US, 1993 [cit. 2020-11-29]. ISBN 978-1-4615-2862-3. Dostupné z: doi:10.1007/978-1-4615-2862-3_86

[4] KALANKESH, Leila, Zahra NASIRY, Rebecca FEIN a Shahla DAMANABI. *Factors Influencing User Satisfaction with Information System.* Galen Medical Journal [online]. 2020, , 1686 [cit. 2020-11-29]. Dostupné z: doi:10.31661/gmj.v9i0.1686

[5] GULLEDGE, Thomas. *What is integration?* Industrial Management and Data Systems [online]. 2006, **106**(1), 5-20 [cit. 2020-11-30]. ISSN 0263-5577. Dostupné z: doi:10.1108/02635570610640979

[6] *Systémová integrace (System Integration).* ManagementMania.com [online]. Wilmington (DE), 2015, 8.2.2015 [cit. 2020-11-30]. Dostupné z: https://managementmania.com/cs/systemova-integrace

[7] BERNARDO, Merce, Alexandra SIMON, Juan José TARÍ a José F. MOLINA-AZORÍN. Benefits of management systems integration: a literature review. *Journal of Cleaner Production* [online]. 2015, **94**, 260-267 [cit. 2020-12-01]. ISSN 09596526. Dostupné z: doi:10.1016/j.jclepro.2015.01.075

[8] CHAPMAN, Christopher S. a Lili-Anne KIHN. Information system integration, enabling control and performance. *Accounting, Organizations and Society* [online]. 2009, **34**(2), 151-169 [cit. 2020-12-01]. ISSN 03613682. Dostupné z: doi:10.1016/j.aos.2008.07.003

[9] HENDERSON, Chloe. 4 Types of System Integration- Pros vs. Cons of Each Method. *Any Conector* [online]. 24 July 2020 [cit.

2020-12-04]. Dostupné z: https://anyconnector.com/system-integration/types-of-system-integration.html

[10] WHAT IS API INTEGRATION? *HCL Technologies* [online]. [cit. 2020-12-04]. Dostupné z: https://www.hcltech.com/technology-qa/what-is-api-integration

[11] What is API: Definition, Types, Specifications, Documentation. *AltexSoft* [online]. 2019, 18 Jun, 2019 [cit. 2020-12-04]. Dostupné z: https://www.altexsoft.com/blog/engineering/what-is-api-definition-types-specifications-documentation/

[12] What's a Webhook? *SendGrid* [online]. 2020, June 24, 201 [cit. 2020-12-04]. Dostupné z: https://sendgrid.com/blog/whats-webhook/

[13] GUAY, Matthew. What are webhooks?: A simple guide to connecting web apps with webhooks. *Zapier Inc.* [online]. 2020, September 12, 2020 [cit. 2020-12-04]. Dostupné z: https://zapier.com/blog/what-are-webhooks/

[14] What is business process management? *IBM* [online]. [cit. 2020-12-04]. Dostupné z: https://www.ibm.com/cloud/automation-software/business-process-management

[15] NÁPLAVA, Pavel.*Informační systémy a procesní řízení* [přednáška]. Praha: České vysoké učení technické v Praze, 23.9.2020.

[16] IBM BPM REST APIs for development. *IBM: Knowledge Center* [online]. [cit. 2020-12-04]. Dostupné z: https://www.ibm.com/support/knowledgecenter/SSFPJS_8.5.7/com.ibm.wbpm.main.doc/topics/cdev_restapis.html

[17] Web on Reactive Stack. *Spring Framework Documentation* [online]. c2002-2020 [cit. 2020-12-14]. Dostupné z: https://docs.spring.io/spring-framework/docs/current/reference/html/web-reactive.html

[18] Spring 5 WebClient. *Baeldung* [online]. [cit. 2020-12-14]. Dostupné z: https://www.baeldung.com/spring-5-webclient

[19] Introduction to Hystrix. *Baeldung* [online]. [cit. 2020-12-14]. Dostupné z: https://www.baeldung.com/introduction-to-hystrix

[20] Guide to Hazelcast with Java. *Baeldung* [online]. [cit. 2020-12-14]. Dostupné z: https://www.baeldung.com/java-hazelcast

[21] Configuring Map Eviction. *Hazelcast Documentation* [online]. [cit. 2021-02-25]. Dostupné z: https://docs.hazelcast.org/docs/3.6/manual/html-single/index.html#configuring-map-eviction

[22] VEENTJER, Peter. In-Memory Format. *Hazelcast* [online]. September 21, 2013 [cit. 2021-02-25]. Dostupné z: https://hazelcast.org/blog/in-memory-format/

[23] Split-Brain Recovery. *Hazelcast Documentation* [online]. [cit. 2021-02-25]. Dostupné z: https://docs.hazelcast.com/imdg/latest/network-partitioning/split-brain-recovery.html#merge-policies

[24] Caching Deserialized Values. *Hazelcast Documentation* [online]. [cit. 2021-02-25]. Dostupné z: https://docs.hazelcast.com/imdg/latest/performance/caching-deserialized-values.html

[25] BUREŠ, Miroslav. Jednotkové testování: JUnit, TestNG a základy efektivního návrhu [přednáška]. Praha: České vysoké učení technické v Praze, 28.3.2019.

[26] FRAJTÁK, Karel. End-to-End Testing [přednáška]. Praha: České vysoké učení technické v Praze, 18.4.2019.

[27] HOMBERGS, Tom. Integration Tests with Spring Boot and @SpringBootTest. *Reflectoring* [online]. [cit. 2021-4-27]. Dostupné z: https://reflectoring.io/spring-boot-test/

Přílohy

Příloha A

Seznam zdrojů

- Obrázek 4.1 – Vytvořeno autorem 12. 2. 2021.
- Obrázek 5.1 – Vytvořeno autorem 5. 3. 2021.
- Obrázek 7.1 – Vytvořeno autorem 15. 3. 2021.

Příloha B

Slovníček pojmů

- IS - Informační systém
- SI - Systémová integrace
- CZM - Centrum znalostního managementu
- IBM - International Business Machines Corporation
- IBM BPM - IBM Business Process Manager
- API - Aplikační rozhraní
- REST API - Restové aplikační rozhraní
- HTTP - Hypertext Transfer Protocol
- FELDS - FEL database-service
- DTO - Data transfer object
- SSO - Single sign-on
- GitLab repozitář - verzovací systém

Příloha C

Seznam elektronických příloh

- Příloha 1 - Zdrojový kód

  Zdrojový kód je rozdělen do několika modulů pro snadnější orientaci

  - **app** - spouštěcí modul
  - **dsclient** - obsahuje client třídy, ve kterých jsou definovené requesty na FELDS API
  - **dto** - nové entity, na které se mapují entity z dsclient
  - **cache** - třídy potřebné ke cachování výsledků
  - **service** - servisni třídy, které volají metody clienta a mapují na nové entity + ukladání do cache paměti
  - **rest** - obsahuje REST controllery, které vytvářejí koncové body (viz swagger dokumentace)

  Integrační a jednotkové scénáře se nacházejí na následujících cestě

  - **Integrační testy** - app/src/test/java/cz/cvut/fel/czm/dsapi/client/FEL_DSE_Test.java
  - **Jednotkové testy** - service/src/test/java/cz/cvut/fel/czm/dsapi/-mapper/

- Příloha 2 - End-to-end testovací scénáře

  Příloha 2 obsahuje seznam všech end-to-end testovacích scénářů, které byly vytvořeny. Soubor lze importovat do programu Postman (potřeba být připojen k CZM VPN).