FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Návrh a realizace spojovacího systému (workflow processor) do produktu „Operativní Manažerský Informační systém“ (OMIS).
Student:	Lukáš Jílek
Vedoucí:	Ing. Miroslav Prágl, MBA
Studijní program:	Informatika
Obor / specializace:	Softwarové inženýrství 2021
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2024/2025

Pokyny pro vypracování

Cílem projektu je návrh a prototypová implementace workflow processoru do stávajícího manažerského informačního systému "Operativní Manažerský Informační Systém" (OMIS).
Workflow processor bude konečný stavový automat určený pro zpracování firemních procesů.

Zadání:
- Proveďte průzkum stávající architektury rámcového prostředí produktu „Operativní Manažerský Informační systém“, možností řešení a dostupných alternativ, zdůvodněte potřebu vlastního řešení.
Navrhněte vhodné ukazatele pro hodnocení výsledného projektu.
- Pilotní workflow processor implementujte jako mikroslužbu (nebo komponentu) vložitelnou do komponentového rámcového prostředí do produktu „Operativní Manažerský Informační systém“. Využijte komponenty, které byly již dříve implementovány.
- Pro snazší konfigurovatelnost alokujte jednotlivé moduly komponenty tak, aby řídící algoritmy a grafické rozhraní byly realizovány v izolovaných jmenných prostorech.
Mezi řídícím algoritmem a obrazovkami zajistěte komunikaci.
- Otestujte a vyhodnoťte výsledek.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 2. října 2023 v Praze.

Bakalářská práce

NÁVRH A REALIZACE SPOJOVACÍHO SYSTÉMU (WORKFLOW PROCESSOR) DO PRODUKTU „OPERATIVNÍ MANAŽERSKÝ INFORMAČNÍ SYSTÉM“ (OMIS)

Lukáš Jílek

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Miroslav Prágl, MBA
16. května 2024

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Chtěl bych poděkovat především své rodině za podporu nejen při tvorbě této práce, ale během celého studia.
Dále bych rád poděkoval vedoucímu své bakalářské práce, Ing. Miroslavovi Práglovi, za konzultace a další pomoc při tvorbě této práce.
Zároveň bych chtěl poděkovat in memoriam inspirátorovi své práce, Ing. Janu Šlechtovi, za počáteční inspiraci k výběru tématu, volné ruce ve volbě technologií a počáteční konzultace některých problematik.

Prohlášení

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

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona

V Praze dne 16. května 2024

Abstrakt

Tato bakalářská práce se zabývá procesem návrhu, tvorby a nasazení spojovacího systému nad pracovními postupy do informačního systému pro manažery.

Systém OMIS je určen především pro malé a střední podniky. Spravuje pracovní procesy, plánování schůzek a reporting provedené práce. Aplikace podporuje projektovou práci, jež umožňuje efektivní nasazení pracovníků, plánování práce, zdrojů a zpětná vazba dává podniku důležitou konkurenční výhodu.

Zaměřil jsem se na kladené požadavky, průzkum ostatních řešení, způsob návrhu, některé speciální partie vývoje se speciálním zaměřením na programovací jazyk Java a framework Spring a zkušenosti s nasazením v reálném prostření.

Klíčová slova správce pracovního procesu, správa skupin, úkoly, plánování

Abstract

This bachelor's thesis deals with the process of designing, creating and deploying a connection system over work procedures into an information system for managers.

The OMIS system is primarily intended for small and medium-sized enterprises. Manages work processes, scheduling meetings and reporting work progress. The application supports project work, which enables efficient deployment of personnel, planning of work, resources, and feedback, giving the company an important competitive advantage.

I focused on the requirements, the research of other solutions, the design method, some special parts of development with a special focus on the Java programming language and the Spring framework, and experience with deployment in a real environment.

Keywords workflow manager, group management, tasks, planing

Seznam zkratek

API	Application Programming Interface
DFA	Deterministic Finite Automaton
EPA	Event-driven Process Chain
GUI	Graphical User Interface
IDE	integrated development environment (integrované vávojové prostředí)
JPA	Java Persistence Api
JVM	Java Virtual Machine
JSON	JavaScript Object Notation
JSON-LD	JSON for Linking Data
NFA	Nondeterministic Finite Automaton
OMIS	Operativní Manažerský Informační Systém
RDBMS	Relational Database Management System
REST	Representational State Transfer
WPSS	Workflow Protocols switching System

Úvod

V dnešní době řada organizací a institucí potřebuje formalizovat svoje pracovní postupy. Od primárního (zemědělství) přes sekundární (stavebnictví, průmyslová výroba, zásilkový obchod) po terciální sektor (školství, zdravotnictví), ale i řídící a vládní agendy (soudnictví, ústavní procesy). Organizace a instituce se snaží multiprojektově plánovat svou činnost a efektivně rozmisťovat zdroje. Plánovat strategicky a takticky, na operativní úrovni nasazovat konkrétní úlohy do hry. Tyto úlohy lze – částečně, nebo plně – automatizovat.

Automatozované úlohy a příslušné komunikační protokoly si vytváří každá organizace či instituce sama, tak aby vyhovovaly jejím potřebám. Většinou jsou protokoly specifikovány při vlastním nasazení spojovacího systému, avšak nesmí být problémem tyto postupy čas od času upravit.

Úlohy lze rozdělit do dvou podskupin: interní, které se plní v rámci jedné organizace a externí, kde jsou k plnění přizvány (spolupracují) jiné organizace nebo hostující osoby. Každá úloha má komunikační protokol pro všechny zúčastněné aktéry určené buď aktérem *in persona*[1] nebo jeho rolí v daném protokolu. V případě interních úloh se komunikační protokol označuje jako *intramise*, v případě externích úloh *extramise*.

Téma mne zaujalo kvůli jeho univerzálnosti a praktické rozšiřitelnosti do různých oblastí. Specifičností mé práce je aktivní řešení tohoto problému, které je dosud zpracováno pouze fragmentárně. Existující řešení se zaměřují pouze na určité aspekty pracovního procesu a jsou často integrovány s omezenou sadou dalších produktů. Nedostatek centrálního správního nástroje pro řízení všech součástí procesu je dalším významným faktorem. Z tohoto důvodu jsem se rozhodl vytvořit univerzální nástroj pro správu procesů, který bude sloužit jako základna pro integraci jednotného jednacího a komunikačního protokolu. Takový nástroj by měl přinést efektivitu a jednoduchost do řízení pracovních procesů a umožnit jejich optimalizaci a koordinaci napříč různými oblastmi a odděleními organizace.

Níže představím stručný úvod do jednotlivých částí:

Cíl práce – Zde stavovím cíle, které bych chtěl při řešení práce splnit.

Analýza – V první části budu zkoumat stávající infrastrukturu, identifikuji klíčové procesy, které k této problematice váží, analyzuji existující řešení. Ve druhé části

[1] osobně

rozeberu návrh systému včetně úprav systému, do kterého je systém vložen.

Implementace – Tato kapitola obsahuje detailní popis implementačních aspektů projektu, včetně technických detailů, architektury aplikace a použitých technologií.

Testování – V této kapitole se zabývám testováním vytvořeného systému.

Zhodnocení – V oslední kapitole práce se zaměřím na komplexní zhodnocení celého projektu, včetně vyhodnocení dosažených výsledků, přínosů práce a možných oblastí pro budoucí rozvoj či vylepšení.

Kapitola 1
Cíl práce

Ve své práci analyzuji, navrhnu, prototypově implementuji, otestuji a zhodnotím workflow processoru do stávajícího manažerského informačního systému „Operativní Manažerský Informační Systém“ (OMIS).

V rámci analýzy provedu rešerši domény automatizovaných pracovnívh procesů. Výstupem této analýzy bude seznam funkčních a nefunkčních požadavků, jež poté použiji při návrhu.

Na závěr práce základě testování zhodnotím své řešení a přínosy pro potencionální uživatele.

Kapitola 2
Analýza

Analýza je první fází pro vývoj řešení. V této kapitole se zabývám teorií procesního řízení, průzkumem stávající architektury, stanovím požadavky na výsledný produkt a prozkoumám existující řešení.

2.1 Procesní řízení

Procesní řízení je v současné době nejrozšířenějších manažerskách přístupů řízení organizace. Vytlačilo dosud hojně používané funkční řízení.

Principem funkčního řízení je rozdělení organizace mezi specializované týmy a práce na produktu je předávána sériově mezi nimi. Tento způsob řízení provází problémy převážně kompetenčního a komunikačního charakteru[1, str. 7-8]. Procesní řízení tyto problémy odstraňuje a navíc vylepšuje o schopnost dokumentace práce, uložení a sdílení know-how a pružnou reakci na změny v okolí [1, str. 7-8].

2.1.1 Proces

Základním kamenem procesního řízení jsou procesy. Formálně je proces definován jako soubor činností, které generují při daných vstupech a meziproduktech cílový výstup. Každému procesu je přiřazena zodpovědná osoba, nazývaná vlastník procesu. Proces by měl ideálně generovat jeden výstup. Procesy lze organizovat do hierarchie (procesní mapy), která zlepšuje přehlednost procesů[1, str. 14-15, 17]. Proces lze popsat různými metodami.

2.1.1.1 Event-driven Process Chain

První metoda nazvaná Event-driven Process Chain (EPC)[2, kap. 2.3], je založena na řetězení událostí a aktivit směřující naplnění cíle procesu. Při specifikování procesu pomocí EPC se využívá několika druhů elementů: aktivity, události a logické spojky. Aktivity určují co má být v rámci procesu vykonáno. Události popisují situace před a/nebo po vykonání aktivity a spojují jednotlivé aktivity. Událost může plnit i podmíněnou roli: při vstupu do události plní roli vstupní podmínky *(precondition)* bez jejíchž splnění nelze do aktivity přejít; v případě ukončení aktivity definuje výstupní podmínku *(postcondition)*,

bez které nelze aktivitu ukončit. Logické spojky se používají k rozdělování, spojování a popisu toku mezi aktivitami a událostmi. Často v procesu EPC plní i synchronizační roli. EPC metoda umožňuje pomocí vnořování a odkazování na ostatní procesy, modularizaci procesů a jejich znovupoužitelnost.

2.1.1.2 Finite State Machine

Druhá metoda nazvaná Finite State Machine (FSM)[1, str. 41] používá pro popis procesu formalismu nedeterministického konečného automatu (NFA). NFA je složen ze tří elementů: stavů, vstupu a přechodové funkce. Stavy tvoří konečnou exkluzivní a v souhrnu vyčerpávající množinu. V počátku své činnosti se NFA nachází v definovaném stavu a přechází do ostatních stavů podle předem definovaných pravidel. Tato pravidla jsou zakódována pomocí přechodové funkce; v kombinaci se vstupem se DFA rozhoduje do kterého dalšího stavu přejde.

2.2 Průzkum stávající architektury

Pro implementaci jsem dostal k dispozici technický návrh částí systému, které s budoucím spojovacím systémem interagují. Původní návrh OMISu vycházel z procesního modelu vyřizování obchodního případu.

Na počátku je nějaký produktový/produkční katalog (product/production catalogue). Ten si lze představit kupříkladu jako webovou aplikaci nějakého obchodu nebo rozcestníku veřejné správy. Uživatel si v produkčním katalogu vybere nějkou akci (nákup jednoho nebo více zboží, zpracování nějaké žádosti, např. výkazu pojištění). Tato akce se přetvoří v tzv. objednávku/požadavek, která se může v dalších krocích libovolně parametricky upravit. Po konečném schválení uživatelem a dokončení procesu se objednávka/požadavek odešle do marketing maileru. Z uživatele, který objednávku odeslal se stává zákazník/klient.

Marketing mailer slouží jako operátorův stůl, jeho úkolem je v případě potřeby se zákazníkem/klientem spojit a dojednat případné změny. Poté ke objednávka postoupena do group/time manageru.

Group/time manager objednávku rozkouskuje na jednotlivé fáze plnění (nalezení zboží – zabalení – odeslání, příprava podkladů – zpracování sestavy – odeslání výsledného dokumentu) které přetvoří na úkoly, přiřadí k nim skupiny lidí a naplánuje časové sloty[1]. Při plánování časových slotů se berou do úvahy časové možnosti vykonavatelů.

Následně je výstup group/time manageru (zadání jednotlivých úkolů, přiřazení skupin, lidí a naplánování časových slotů) zpracován project/task managerem. Project/task manager přiřadí úlohy do projektů a spustí proces realizace objednávky.

V současné době je výše uvedený návrh přepracováván. Ukázal se zejména nedostatek, kdy izolace jednotlivých fází neumožňuje znovupoužitelnost jednotlivých systémů. Proto se klient rozhodl pro přepracování celé architektury. Marketing mailer bude úplně zrušen a jeho odpovědnosti se přesunou mezi ostatní moduly. Group/time manager bude rozdělen do dvou nezávislých modulů. Time managger přidělovat časové sloty, ale až po zpracování požadavku group managerem. Zároveň bude bude upravovat sloty při

[1] časový úsek, kdy se má jednotlivá činnost/úkol vykonávat.

jakékoliv změně. Bude vtvořen nový modul workflow manager, který bude spravovat pracovní procesy. Zodpovědností nového modulu bude koordinace ostatních modulů.

2.3 Požadavky

Pro stanovení řešení jsem provedl sběr požadavků, které jsou kladeny na nový modul a analyzoval je. Požadavky jsem získal částečně z průzkumu stávající architektury a částečně z komunikace se zadavatelem. Použil jsem standardní kategorizaci na funkční a nefunkční požadavky. Poté jsem namodeloval případy užití.

2.3.1 Funkční požadavky

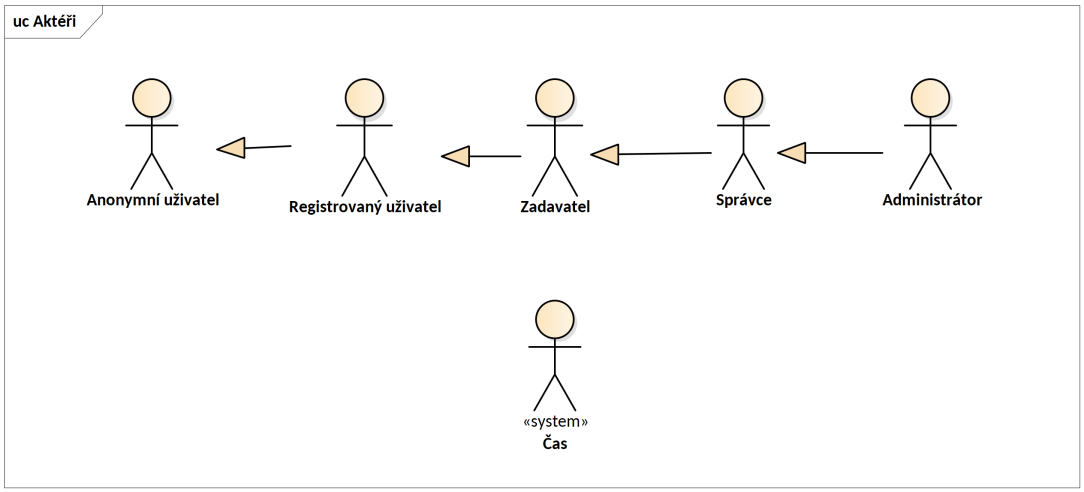
Pomocí funkčních požadavků stanovím, jaké funkce od nového modulu očekávám.

- **F1: Správa pracovních postupů** – Požadavek vyplývá ze zadání práce. Řešení bude vytvářet, upravovat a mazat pracovní postupy.
- **F2: Úkoly** – Výsledné řešení bude napojeno na modul pro úkoly. Každý úkol bude obsahovat napojení na nějaký pracovní proces a sledovat aktuální stav úkoilu vzhledem k danému pracovnímu procesu.
- **F3: Sedování aktuálního stavu úkolu** – Výsledné řešení umožní sledování aktuálního stavu úkolu.
- **F4: Přechod mezi aktivitami** – Řešení bude uživateli umožňovat přecházet (ručně nebo automatizovaně) mezi jednotlivými aktivitami a mezi jednotlivými pracovními postupy.

2.3.2 Nefunkční požadavky

Pomocí nefunkčních požadavků určím další vlastnosti mého řešení. Pro nefunkční požadavky platí, že různorodost platforem podporovaných produktem OMIS se odvíjí od dynamicky se rozvíjejícího sektoru webových technologií. Bez datového spojení (zvláště, když chceme spolupracovat ve víceuživatelském módu) nelze produkt realizovat. Avšak nelze očekávat, že produkt bude podporovat pouze uzavřené formáty. Toto omezení stanovuji z důvodu zajištění budoucí kompatibility komponent OMISu. Pokud by toto omezení neplatilo, byla by byla výrazně ztížena možnost realizace škálování produktu a případné obohacení produktu o nové funkce.

- **N1: Webová aplikace** – Řešení bude realizováno jako webová aplikace, což zaručí snadný a přístupný způsob interakce s uživateli.
- **N2: Propojení s ostatními produkty** – Implementace tohoto řešení zahrnuje vytvoření rozhraní, které umožní propojení s ostatními aplikacemi v rámci systému.
- **N3: Intuitivní rozhraní** – Modul musí být navrhnut tak, aby mohl uživatel pohodlně přepínat mezi jednotlivými aktivitami.
- **N4: Rozšiřitelnost** – Architektura tohoto řešení je navržena tak, aby podporovala budoucí rozšíření a integraci s dalšími komponentami systému OMIS.



Obrázek 2.1 Přehled hierarchie aktérů

2.3.3 Případy užití

V další fázi potřebuji podrobně popsat funkční požadavky. Sestavím model případů užití složený z diagramů a doprovodného textu, který slouží k popisu případů užití.

2.3.3.1 Aktéři

Entity, které moje řešení užívají se nazývají aktéry. V modulu jsem identifikoval čtyři aktéry, kteří jsou hierarchicky uspořádáni (obr. 2.1). Vyjímku v této hierarchii jsem udělil pouze aktérovi *Čas*,

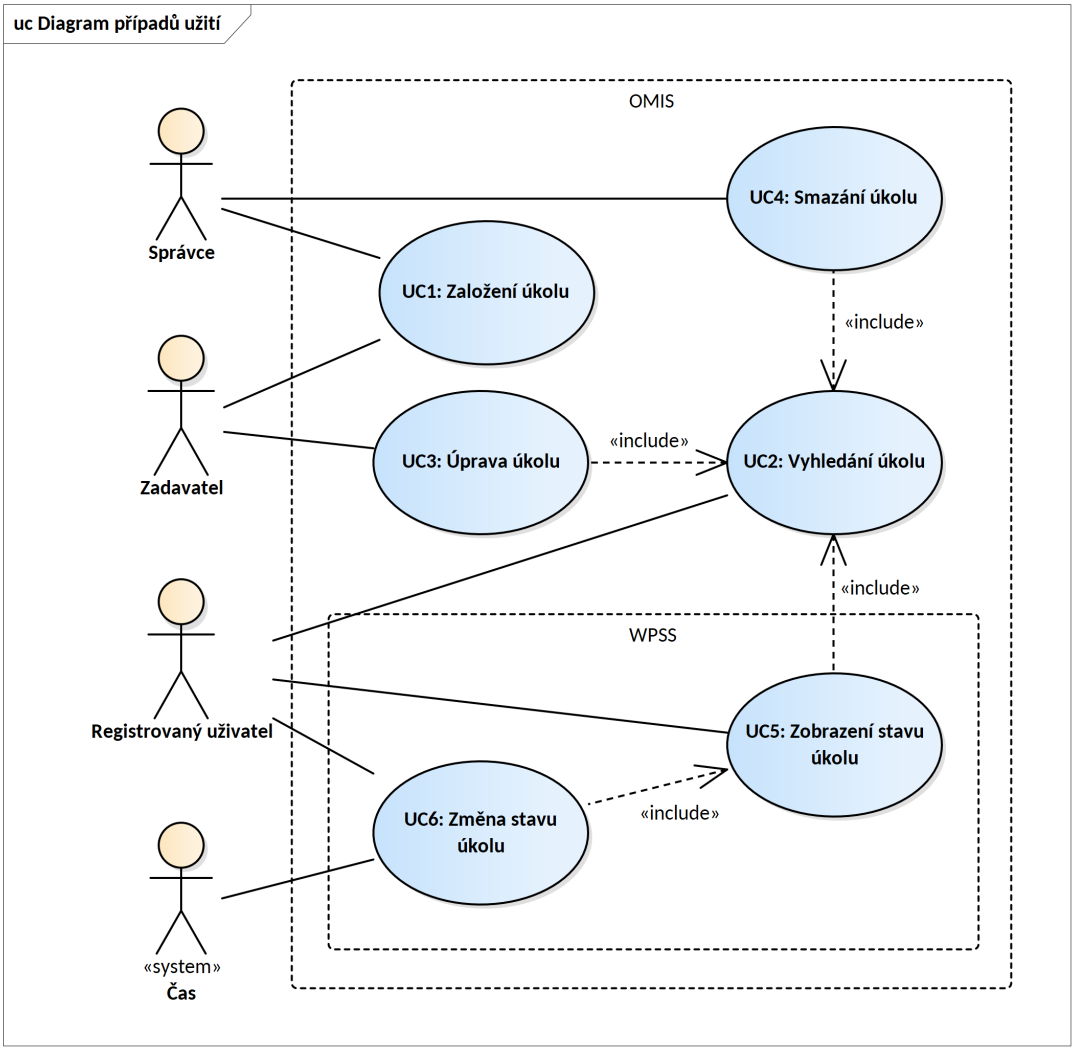
- **Anonymní uživatel** – Jakákoliv uživatel, která navštíví platformu. Tomuto uživateli umožním registraci uživatelského konta, přihlášení a odeslání zapomenutého hesla.
- **Registrovaný uživatel** – Registrovaný uživatel má v systému zavedeno konto a systém ho plně identifikoval dle platných politik systému. Oproti *neregistrovanému uživateli* má přístup do schválených projektů a úkolů a posouvat aktuální stav
- **Zadavatel** – *Registrovaný uživatel*, který může vytvořit nový úkol a upravovat ho.
- **Správce** – Správce má stejné možnosti, jako *registrovaný uživatel*, navíc může upravovat jednotlivé úkoly a projekty.
- **Administrátor** – Administrátor je nejvýše postavený aktér. Navíc může zakládat, měnit a rušit uživatelská konta.
- ***Čas*** – Aktér pro automaticky spouštěné úlohy na základě časové události. Je rovnocenný *správci*, ale stojí mimo hierarchii.

2.3.3.2 Případy užití

Případy užití jsem namodeloval pomocí UML diagramu na obrázku 2.2. Následují jednotlivé případy včetně základních scénářů. U případu **UC2: Vyhledání úkolu** jsou vypsány dva alternativní scénáře.

- **UC1: Založení úkolu**.
  - Správce se přihlásí do aplikace.
  - Přejde do sekce úkoly a klikne na tlačítko pro založení nového ukolu.
  - Vyplní vyžadovaná políčka formuláře.
  - Formulář potvrdí potvrzovacím tlačítkem.
- **UC2: Vyhledání úkolu**

  1. scénář:
     - Aktér vyhledá úkol prostřednictvím přímého přístupu přes URL.
     - Pokud není přihlášen, projde procesem přihlášení.
  2. scénář:
     - Uživatel se přihlásí do aplikace.
     - Přejde do sekce úkoly a klikne na tlačítko zobrazení podrobností u vybraného úkolu.
- **UC3: Úprava úkolu**
  - Správce se přihlásí do aplikace.
  - Přejde do sekce úkoly a klikne na editační tlačítko u vybraného úkolu.
  - Vyplní vyžadovaná políčka formuláře.
  - Formulář potvrdí potvrzovacím tlačítkem.
- **UC4: Smazání úkolu**
  - Správce se přihlásí do aplikace.
  - Přejde do sekce úkoly a klikne na tlačítko smazání u vybraného úkolu.
- **UC5: Zobrazení stavu úkolu** – Aktér po vyhledání zobrazí stav úkolu vzhledem k jeho pracovnímu procesu.
  - Uživatel se přihlásí do aplikace.
  - Přejde do sekce úkoly a klikne na tlačítko zobrazení podrobností u vybraného úkolu.
- **UC6: Změna stavu úkolu**
  - Uživatel se přihlásí do aplikace.
  - Přejde do sekce úkoly a klikne na rozbalovací seznam vedle informace o stavu u vybraného úkolu.
  - Vybere požadovanou akci.



Obrázek 2.2 Diagram případů užití

2.3.4 Kritéria hodnocení výsledného řešení

Pro hodnocení výsledného řešení jsem vybral 2 kritéria: možnost průchodu jednotlivými případy užití a míru splnění funkčních a nefunkčních požadavků.

2.4 Analýza existujících řešení

Pro splnění požadavků jsem zvažoval 2 možné alternativy: externí napojení na již existující řešení nebo interní vývoj. Pro první alternativu jsem zjistil, jaká hotová řešení je možno použít, jejich přednoti a nedostatky. Provedl jsem průzkum a vybral tři reprrezentativní vzorky. Pro druhou alternativu navrhnu vlastní řešení. Jelikož u této varianty předpokládám, že budou jednotlivá kritéria splněna, porovnám varianty z první alternativy a poté tu nejvhodnější porovnám s druhou alternativou.

2.4.1 JIRA

JIRA je webový nástroj zaměřený na evidenci chyb v procesu vývoje sofwarových produktů. Je vyvíjen společností Atlassian, Inc. od roku 2002. Nástroj je primárně provozován jako cloudová služba s možností provozu na vlastní infrastruktuře.

Vnitřně lze JIRu integrovat pomocí zásuvných modulů[**Jira**]. Pro vnější integraci s ostatními produkty podporuje REST[3] rozhraní.[4].

Pro organizaci úkolů v projektu má JIRA vybudován hierarchický systém, který je možné měnit. Nejčastěji se používá hierarchie *epic – story – task – subtask*. *Epic* reprezentuje nějaký velký díl práce, časově rozprostřený mezi jednotlivé sprinty. *Story* reprezentuje nějaký konkrétní výstup pro zákazníka. *Task* a *subtask* reprezentují úkoly, které je třeba pro tento výstup splnit.[5]

JIRA disponuje spojovacím systémem pro úkoly. Pro model pracovního postupu používá model FSM. Stavy mohou být přiřazeny to *stavových kategorií*. Přechody jsou řešeny pouze čistým propojením dvou stavů. Nevýhodou tohoto čistého propojení je nutnost znát daný pracovní proces, jeho přechody a podmínky předem.[6] To obvykle vede ke vzniku různých postranních návodů, které komplikují jak samotné provádění, tak následný audit.

- ***Klady***
  - Pokročilá správa pracovních postupů a úkolů
- ***Zápory***
  - Práce s přechodem mezi stavy vyžaduje speciální manuál

2.4.2 Azure Boards

Azure Boards je součástí produktové řady DevOps od společnosti Microsoft. Tento nástroj je primárně navržen pro vývojáře v rámci integrované vývojové prostředí Visual Studio.

Úkoly v Azure Boards mají podobný organizační systém jako JIRA, avšak nemá možnost dynamické konfigurace pracovního postupu během průběhu projektu. Organizace úkolů a pracovních postupů jednotlivých typů úkolů je nastavena administrátorem při vytváření projektu pomocí procesního modelu. Tento model obsahuje pouze jednotlivé stavy, přechody mezi nimi nejsou definovány. Při práci s tímto nástrojem je proto nutné vytvořit manuál popisující pracovní postup[7].

- ***Klady***
  - Propojení na ostatní produkty Microsoftu
- ***Zápory***
  - Práce s přechodem mezi stavy vyžaduje speciální manuál

2.4.3 MantisBT

MantisBT byl v roce 2016 jedním z nejoblíbenějších nástrojů pro správu chyb. Tento nástroj s otevřeným zdrojovým kódem je vyvíjen od roku 2000 a je licencován pod svobodnou licencí. Implementace je provedena v jazyce PHP a data jsou ukládána v různých relačních databázových systémech, včetně PostgreSQL, MariaDB/MySQL a MS SQL[8, kap. 1].

Úkoly jsou organizovány do projektů bez určení jejich typu nebo hierarchické struktury. Stavy úkolů jsou řízeny pevně stanoveným pracovním postupem, který umožňuje konfiguraci pouze přechodů mezi stavy [8, kap. 4]

- ***Klady***
  - Open source
- ***Zápory***
  - Pracovní postup je obtížně konfigurovatelný

Výsledky průzkumu jsem shrnul do tabulky 2.1. Tabulka zobrazuje jednotlivá řešení a splnění kritérií. Splněné kritérium jsem v tabulce označil symbolem „A“, v případě nesplnění symbolem „N“. Výsledkem tohoto průzkumu je, že pro spojovací systém existují různá integrovaná řešení. Ani jedno nesplňuje nejdůležitější požadavek – intuitivní rozhraní. U mnou zkoumaných řešení řechod mezi aktivitami probíhá pomocí změny této aktivity. Uživatel musí pracovní proces znát z jiného zdroje. Dospěl jsem k závěru, že žádné řešení z první varianty plně nevyhovuje a tedy přistupuju na variantu druhou – vytvoření vlastního řešení.

2.5 Návrh vlastního řešení

V mém návrhu kombinuji obě metody popisu pracovního procesu, jak EPC, tak FSM. Obě metody jsou do určité míry kompatibilní. Za základ používám metodu FSM, ze které přebírám DKA. Na stavy DKA mapuji aktivity a na počátky přechodů mapuji události

Tabulka 2.1 Srovnání řešení první varianty

Název řešení	F1: Správa pracovních postupů	F2: Úkoly	F3: Sedování aktuálního stavu úkolu	F4: Přechod mezi aktivitami	N1: Webová aplikace	N2: Propojení s ostatními produkty	N3: Intuitivní rozhraní	N4: Rozšiřitelnost	UC1: Založení úkolu	UC2: Vyhledání úkolu	UC3: Úprava úkolu	UC4: Smazání úkolu	UC5: Zobrazení stavu úkolu	UC6: Změna stavu úkolu
JIRA	A	A	A	A	A	A	N	A	A	A	A	A	A	A
Azure Boards	A	A	A	A	A	A	N	A	A	A	A	A	A	A
MantisBT	N	A	A	A	A	N	N	N	A	A	A	A	A	A

(ve formě výstupní podmínky). Logické spojky nemohu mapovat tak jak jsou, použiji spíše dodatečné stavy, které jsou vhodně doplněny vstupními a výstupními podmínkami. Kombinace DKA a FSM je výhodná vzhledem k rozšiřování o další atributy.

2.5.1 Doménový model

Pro vývoj vlastního řešení jsem vytvořil doménový model, který je na obrázku 2.3. Entity jsem z části převzal z části ze starého systému, zbytek z komunikace se zadavatelem.

Entity jsem rozdělil do dvou skupin.

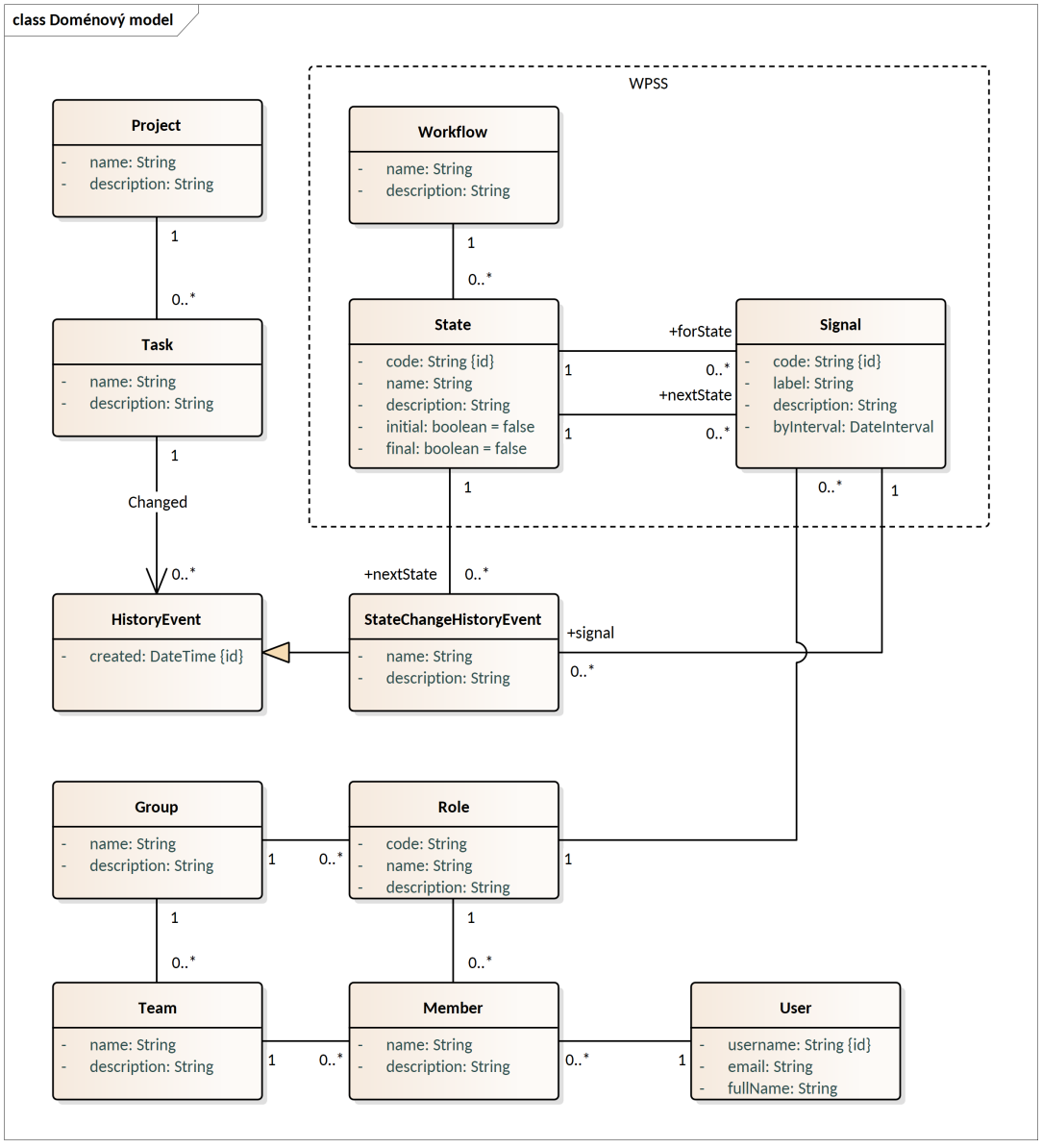
2.5.1.1 Entity použité workflow managerem

První skupinu tvoří entity, které přímo používá modul workflow manageru,

- **Workflow** ***(Pracovní postup)*** – Reprezentuje pracovní proces s definovaným názvem a volitelným popisem.
- **State** ***(Stav)*** – Představuje konkrétní fázi, kterou pracovní proces prochází. Každý stav je identifikován svým kódem a má svůj název a volitelný popis. Může být označen jako počáteční nebo koncový stav.
- **Signal** ***(Signál)*** – Jednoznačně propojuje dva stavy v pracovním procesu. Signál umožňuje přechod mezi stavy.

2.5.1.2 Entity napojené na workflow manager

Druhou skupinu tvoří entity, které jsou na modul workflow manager napojeny.



Obrázek 2.3 Datový model WPSS s napojením na OMIS

- **Project** – Projekt reprezentuje soubor úkolů s cílem dosáhnout společného výsledku nebo vytvoření nějakého produktu.
- **Task *(Úkol)*** – Reprezentuje jeden úkol na kterém pracuje v jeden okamžik jeden člověk. Úkol má název a popis.
- **StateChangeHistoryEvent** – Reprezentuje změnu stavu jednoho úkolu. Zaznamenává přechod mezi stavy, jakým signálem byl tento přechod vyvolán a osobu která přechod iniciovala.
- **User** – Entita reprezentující uživatelské konto.
- **Group** – Entita reprezentující skupinu. Skupina je v kontextu projektu OMIS chápána jako soubor rolí, které spolupracují. V případě IT podniku existují skupiny jako *„Vedení“*, *„Výroba frontendu v Jazyce Java“* nebo *„Testování“*.
- **Role** – Role ve skupině definuje rámec odpovědnosti. Například skupina *„Vedení“* obsahuje role *„Majitel“*, *„Ředitel“* a *„Vedoucí ekonomického úseku“*.
- **Team** – Konkrétní skupina složená z členů. Každý člen týmu má určitou roli definovanou ve skupině.
- **Member** – Reprezentuje člena týmu. Člen týmu je uživatel, který má v týmu svou specifickou roli.

Kapitola 3
Implementace

V této kapitole popíšu implementaci včetně použitých technologií a rozebeu některé jevy se kterými jsem se při implementaci setkal

3.1 Postup implementace

Samotnou implementaci jsem rozdělil do dvou hlavních částí. První část se zaměřuje na samotný spojovací systém, který zahrnuje správu skupin a týmů. Druhou část tvoří technologický demonstrátor obsahující nezbytné prvky projektu OMIS, které se spojovacím systémem spolupracují. Zde je hlavním cílem správa uživatelů a jejich oprávnění, spolu sřízením úkolů. V rámci projektu jsem implementoval API i uživatelské rozhraní (GUI).

3.2 Použité technologie

Pro výrobu prototypu spojovacího systému pro pracovní postupy jsem zvolil prostředí JVM a Spring Boot. Pro spouštění jsem použil platformu Docker.

3.2.1 Docker

Docker je open-source platforma navržená pro vývoj, doručování a spouštění aplikací. Skládá se ze dvou částí: démona (proces běžící na pozadí, který spravuje kontejnery) a klienta (uživatelský komunikátor s démonem).

Platforma je založena na dvou důležitých konceptech: izolaci a dokumentaci prostředí. Pro izolaci prostředí se používá technika kontejnerizace. Kontejner je oddělený proces běžící na hostujícím systému. Jediným omezením pro spuštění kontejneru je architektura procesoru pod kterou je spuštěn hostující systém.

Pro spuštění kontejneru je nezbytný tzv. obraz (*image*). Tento obraz obsahuje všechny potřebné komponenty pro běh aplikace, včetně běhového prostředí, konfiguračních souborů a dat. Docker obraz se skládá zněkolika znovupoužitelných vrstev, které jsou vytvářeny postupně během procesu sestavení. Tento proces sestavení je řízen pomocí speciálního konfiguračního souboru nazývaného *Dockerfile* (3.1). Po sestavení je možné obraz nahrát (*push*) do registru, odkud si ho mohou ostatní uživatelé stáhnout (*pull*) nebo využít jako základ pro další úpravy.

Výpis kódu 3.1 Ukázkový kód Dockerfile pro spuštění Java aplikace zabalené do JAR archivu

```
version: '3.8'


services:
  app:
    image: openjdk:23-slim-bookworm
    restart: 'always'
    ports:
      - "8080:8080"
    working_dir: /path/to/app
    entrypoint: java -Xms128m -Xmx512m -jar -Dserver.port=8080 target/app-0.0.1-SNAPSHOT.jar
    environment:
      SPRING_DATASOURCE_URL: "jdbc:postgresql://database:5432/app"
      SPRING_MAIL_HOST: "sendmail"
    volumes:
      - .:/path/to/app:rw
```

V rámci mé práce jsem využil Docker jako prostředek dokumentace prostředí, ve kterém byl spuštěn prototyp spojovacího systému. Pomocí Dockeru jsem zajistil konzistentní a snadno reprodukovatelné prostředí pro vývoj a testování aplikace.

3.2.2 Java Virtual Machine

Java Virtual Machine (JVM) je virtuální stroj sloužící ke spouštění počítačových programů. Zdrojový kód programu se napřed zkompiluje do tzv. *bytecode* (sada instrukcí určené pro JVM). V tomto stavu se program distribuje. Před spuštěním programu v JVM prostředí se poté *bytecode* překládá do strojového kódu procesoru systému, na které program běží. *Bytecode* zajišťuje přenositelnost a nezávislost programů na cílovém systému.

Nejčastěji užívaným kompilátorem do *bytecode* je kompilátor z programovacího jazyka Java. Java je programovací jazyk se statickou kontrolou kódu. Java je důsledně objektově orientovaný jazyk, syntakticky vychází z C++. Je zaměřena na maximální bezpečnost (nemá nebezpečnou práci s ukazateli, nedělá nebezpečné implicitní přetypování). Podle TIOBE Indexu je čtvrtým nejpoužívanějším jazykem[9]. Pro Javu existuje velké množství knihoven a frameworků a pro programátory velké množství materiálů, návodů a postupů pro vývoj.

Pro JVM lze kompilovat i z jiných jazyků (například *Kotlin* vytvořený českou vývojářskou firmou JetBrains).

3.2.3 Spring

Spring Framework je aplikační rámec pro vývoj enterprise aplikací v programovacím jazyce Java. Vznikl v rove 2003 jako reakce na složitost a těžkopádnost ranných specifikací J2EE. Snaží se používat ověřená řešení, odkládat návrhová rozhodnutí na co nejpozdější dobu a na základě znovupoužitelných bloků sestavit výslednou aplikaci. Spring Framework je modulárním frameworkem, umožňující použití pouze potřebné části ekosystému k řešení daného problému.

Jádro frameworku je postaveno ná návrhovém vzoru *Inversion of Control* (IoC). Podstatou návrhového vzoru je předání zodpovědnodti vytváření a provázání objektů z aplikace na framework. To je zajištěno tzv. IoC kontejnerem, pomocným objektem, který za pomoci konfigurce posktuje službu[1]. Ostatním objektům tedy stačí mít refe-

[1] vytváří, doplní závislosti a inicializuje na požádání zkonfigurovaný objekt

renci na příslušnou službu, kterou IoC kontejner poskytuje. Další moduly zajišťují propojení s databázovým systémem, poskytování různých služeb (REST API, JSON-LD,...), zabezpečení a jiné potřebné funkce.[10]

Spring Boot je rozšíření Spring frameworku, které pomocí předpřipravené konfigurace prostředí zjednodušuje a zrychluje produkci enterprise aplikací. Dodává se do aplikací ve formě *startérů*[2].

3.2.4 REST

REST je architektonický vzor pro vytváření API pro prezentaci a manipulaci dat ze serveru. Je založen na několika návrhových charakteristikách:

- Klient a server jsou odděleni
- Mezi jednotlivé klientskými dotazy neexistuje z pohledu serveru žádná závislost. Veškeré řízení komunikace je plně závislé na klientovi.
- Klient a server komunikují předvídatelným způsobem.
- Přítomnost prostředníků nemění chování serveru.

REST pužívá HTTP protokol pro kladení požadavků a získávání odpovědí. [11] [12]

3.3 Inicializace projektu

Při vytvoření nového projektu jsem použil *Spring Initializr* (`https://start.spring.io`). Spring Initializr je webová aplikace určená pro generování Spring Boot projektů. Po nakonfigurování parametrů projektu je možné si stáhnout výsledný projekt ve formátu ZIP archivu.

V projektu jsem využil následující závislosti:

- **Spring Boot Starter Web** – Startér pro tvorbu webových a REST aplikací. Umožňuje rychlé a snadné nastavení základních funkcí webové aplikace, jako je například zpracování HTTP požadavků a odpovědí.
- **Spring Boot Starter Security** – Knihovna poskytující prostředí pro ověřování uživatelů, řízení přístupu ke zdrojům a ochranu webové aplikace proti známým útokům, jako jsou XSS nebo CSRF. Umožňuje integraci s dalšími knihovnami pro rozšíření bezpečnostních funkcí.
- **Spring Boot Starter Data JPA** – Projekt Spring Data JPA usnadňuje implementaci přístupu k persistentním datovým zdrojům pomocí *Java Persistent API (JPA)*. Poskytuje nástroje pro jednoduché mapování objektů do databáze a zpět, což výrazně zjednodušuje práci s daty.
- **Spring Boot Starter Mail** – Podpora pro odesílání elektronické pošty. Umožňuje snadnou konfiguraci a odesílání e-mailů přímo z aplikace, což je užitečné pro zasílání notifikací a dalších zpráv uživatelům.

[2] kolekce modulů reprezentována společným názvem

- **Spring Boot Starter Validation** – Poskytuje nástroje pro snadnou a přesnou validaci dat. Pomocí anotací lze jednoduše definovat validační pravidla pro vstupní data, což zvyšuje spolehlivost a bezpečnost aplikace.
- **Spring Boot Starter Test** – Framework pro psaní automatizovaných testů. Umožňuje psaní a spouštění testů pro různé části aplikace, což je klíčové pro zajištění kvality a spolehlivosti kódu.
- **Spring Boot Starter Thymeleaf** – Poskytuje šablonovací systém Thymeleaf pro generování výstupů. V současné době podporuje generování textového výstupu a jeho nadstaveb (XML, HTML, CSS,...). Umožňuje dynamicky vytvářet a upravovat HTML stránky na základě dat z aplikace.
- **Lombok** – Lombok je knihovna, která pomocí anotací automatizuje generování tzv. *boilerplate kódu*[3]. Typicky se v projektu generují *gettery* a*settery*[4]. Použitím Lomboku lze výrazně zjednodušit a zpřehlednit zdrojový kód.

3.4 Nastavení aplikace

Po stažení projektu vyvstala potřeba dodat závislostem některá důležitá nastavení. To je možné udělt třemi způsoby:

- **Konfigurační třídy** – Jedná se třídy, které jsou označeny pomocí třídní anotace `@Configuration`. Tyto třídy slouží *IoC kontejneru* pro nastavování jednotlivých objektů, které pak spravuje.
- **Properties soubory** – Soubory, ve kterých lze nastavit některé parametry (např. port na kterém bude spuštěná aplikace poslouchat požadavky) V projektu je automaticky vygenerován `src/main/resources/application.properties`, parametry se dávají přednostně sem. Pokud vývojář potřebuje nastavit odlišné parametry u automatizovaných testů, je možné příslušné parametry předefinovat v souboru `src/test/resources/application.properties`. Ten není automaticky generován.
- **Proměnné prostředí** – Některé komponenty springu hledají nastavení i v proměnném prostředí při spuštění programu. Pokud je proměnná komponentou detekována, má při nastavování komponenty přednost před Properties soubory.

3.5 Odesílání elektronické pošty

Odesílání elektronické pošty jsem implementoval za použití knihovny Java Mail. Obsah zpráv vytvářím pomocí šablonovacího systému Thymeleaf. Šablony jsou umístěně ve složce `src/main/resources/templates/mails`.

Obsluhu odesílání zpráv jsem soustředil do speciální třídy `MailerService`. Ukázkový kód této třídy je ve výpisu kódu 3.2. Pro každou šablonu jsem použil vlastní `send*(...)` metodu do které předávám v parametrech datové objekty.

[3] Kód řešící obecný problém, který je nutné psát opakovaně
[4] *Getter/setter* - metoda sloužící k získání/uložení soukromé členské proměnné v objektu

Výpis kódu 3.2 Ukázkový kod maileru

```
@Service
public class MailerService {

  @Autowired
  JavaMailSender mailSender;

  @Autowired
  private ResourceLoader resourceLoader;


  public MimeMessage sendSomething(SomethingData somethingData, RecipientData recipientData)
  throws MessagingException, MailException {
    MimeMessage message = mailSender.createMimeMessage();
    MimeMessageHelper helper = createMimeMessageHelper(message);

    try {
      helper.addTo(recipientData.getEmail(), recipientData.getUsername());

      Map<String, Object> model = new HashMap<>();
      model.put("something", somethingData);

      Context context = new Context();
      context.setVariables(model);
      Document body = Jsoup.parse(MailConfiguration.mailTemplateEngine()
          .process("path/to/something.html", context)
      );
      helper.setSubject(body.title());
      helper.setText(body.toString(), true);

      helper.addAttachment(
          "welcome.pdf",
          resourceLoader.getResource("classpath:static/mails/something.pdf")
      );

      mailSender.send(message);

      return message;
    } catch (UnsupportedEncodingException e) {
      throw new RuntimeException(e);
    }
  }


  private static MimeMessageHelper createMimeMessageHelper(MimeMessage message)
    throws MessagingException {
    return new MimeMessageHelper(message, true, Encoding.DEFAULT_CHARSET.displayName());
  }
}
```

3.6 Zabezpečení

Pro zabezpečení aplikace jsem použil Spring Security. Konfiguraci jsem zajistil pomocí konfigurační třídy `SecurityConfiguration`. Z důvodu implementace API a GUI rozhraní, jsem konfiguraci zabezpečení rozdělil na dvě části. Pro potřebu obou částí jsem vytvořil správu uživatelů a jejich přihlášení. API používá pro autentizaci prootokol *HTTP basic authentication*, GUI používá webový přihlašovací formulář.

3.7 Business vrstva

Na této vrstvé se nachází doménové entity. Při implementaci jsem vycházel z doménového modelu (obrázek 2.3) a v případě potřeby rozšířil o další atributy (unikátní ID záznamu, propojení s ostatními entitami,...)

Ke každé entitě jsem vytvořil ještě několik obslužných tříd:

- **Formulářová třída** – Datový objekt určený k uložení nového záznamu. Pokud je ID entity automaticky generováno databází nebo ORM frameworkem, v formulářové třídě se nevyskytuje.
- **ID třída** – Reprezentuje identifikátor entity. Je určena anotací `@IdClass` na entitě.
- **Datová třída** – Používá se při předávání dat do prezentační vrstvy.

3.8 Datová vrstva

Datová vrstva získává data z nějakého datového skladu (databáze). K reprezentaci jednotlivých konceptů (Uživatel, Úkol, Stav,...) jsem použil entity.

Komunikaci s persistentní vrstvou (databází) jsem zajistil pomocí tzv. repozitáře. Repozitář je třída implementující rozhraní `JpaRepository<E, ID>`, kde E je typ entity k uložení a ID je typ třídy použité jako primární klíč (v projektu striktně používám ID třídu).

Rozhraní `JpaRepository<E, ID>` poskytuje základní služby pro manipulaci s entitami v databázi (uložení, získání dle ID entity, mazání). Pro získání entity (kolekce entit) na základě složitějších ktitérií jsem většinou použil metodu odvození dotazu z názvu metody[5], ve složitějších případech jsem dotaz definoval přímo v JPQL pomocí anotace `@Query`. Rozdíl názorně ukazuji ve výpisu kódu 3.3. Metoda `List<ByPassword> findByUser(User user)` je zapsána metodou odvozovací, metoda `List<ByPassword> user(@Param("user") User user)` používá anotaci `@Query`, uvnitř anotace je definován dotaz v JPQL.

Ve složitějších případech jsem použil Java Persistence API specifikace, konkrétně Criteria API. Criteria API je rozhraní, které umožňuje vytvářet dotazy nad persistentní vrstvou pomocí objektově orientovaného přístupu. V projektu jsem Criteria API použil pro vyhledávání úkolů podle složitějších kritérií (například vyhledávání dle signálů vedoucích z aktuálního stavu, na které může uživatel ve svých přidělených roli reagovat).

[5] dokumentace na `https://docs.spring.io/spring-data/jpa/reference/jpa/query-methods.html`

Výpis kódu 3.3 Ukázkový kod repozitáře

```
@Repository
public interface ByPasswordJpaRepository extends JpaRepository<ByPassword, ByPasswordId> {

  List<ByPassword> findByUser(User user);

  @Query("select␣u␣from␣ByPassword␣u␣where␣u.user␣=␣:user")
    List<ByPassword> user(@Param("user") User user);
}
```

3.9 Převod entit na datové objekty

Pro předání dat do prezentační vrstvy API rozhraní jsem použil převodu entity do datového objektu pomocí nástroje MapStruct[13]. MapStruct je nástroj pro generování kódu v jazyce Java. Nástroj je optimalizován ke generování kódu pro převod objektů mezi různými vrstvami aplikace. Poskytuje konfigurační nástroje pro přizpůsobení automaticky generovaného kodu podle potřeb aplikace.

Kapitola 4
Testování

Tuto část práce věnuji testování aplikace. Použiji automatické i manuální testování.

4.1 Automatické testování

Podstatou automatického testování je použití specializovaného software k řízení, provádění a ověřování výsledků z testování s předem připravovanými předpokládanými výsledky. Výhodou automatického testování je zajištění vysoké kvality kódu během procesu vývoje. To je zajištěno pomocí předem připraveného testovacího scénáře, který lze opakovaně provést.

Pro ověření funkčnosti aplikace a zajištění kvality kódu jsem použil metodu integračního testování zdrojového kódu. Integrační testy testují malé části kódu (funkce, třídy, metody,...) ve spolupráci s ostatními částmi aplikace. Na rozdíl od jednotkových testů, integrační testování může ověřit kromě zdrojového kódu i funkčnost okolní infrastruktury. V projektu jsem pro integrační testování použil dva externí systémy: RDBMS[1] na ukládání dat a testovací framework GreenMail[14] pro testování odesílání zpráv elektronické pošty.

Otestované části kodu:

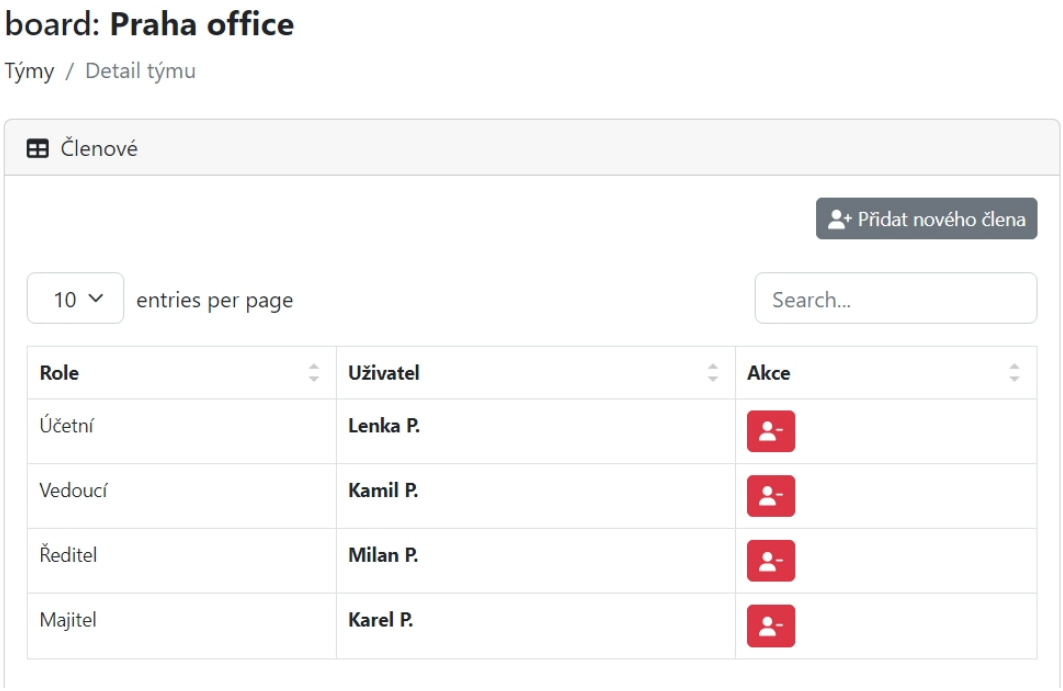
- základní operace (vytvoření, uložení, získání a smazání) nad persistentním úložištěm u většiny entit
- REST API volání
- Proces přepnutí stavu

Testy jsem pokryl většinu zdrojového kódu.

4.2 Uživatelské testování

Prototyp jsem otestoval u zadavatele s využitím šesti testerů, rozdělených do dvou týmů. První tým představoval vedení podniku (obr. 4.1), zatímco druhý tým tvořili vývojáři

[1] Relational Database Management System



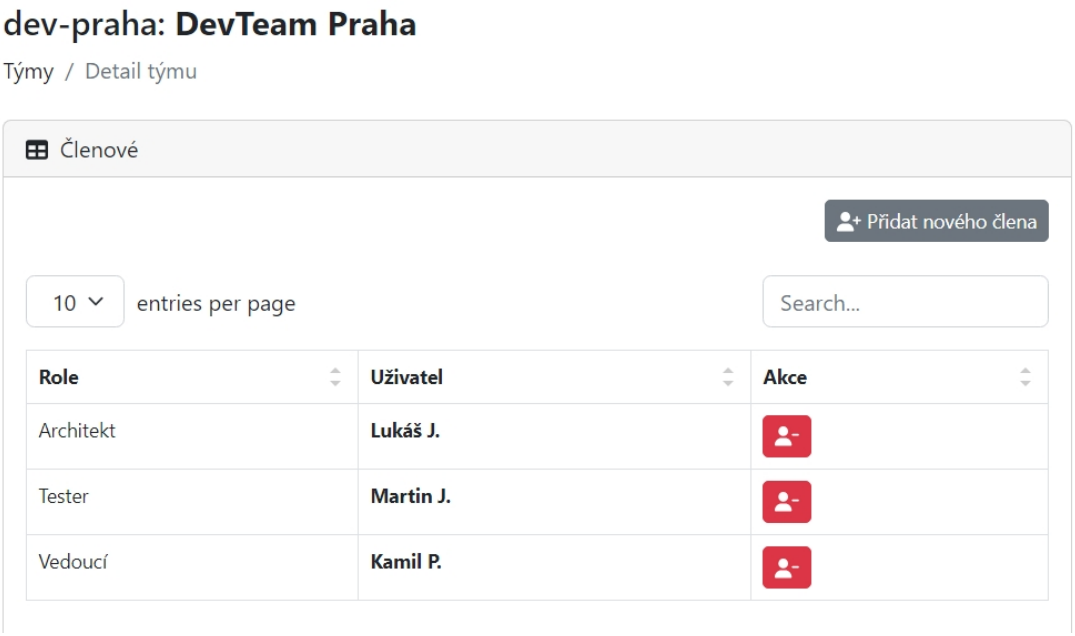
Obrázek 4.1 Obsazení vedení firmy

Java aplikací v Praze (obr. 4.2). Vedení podniku se zaměřovalo na vyřizování objednávek a smluv, zatímco vývojářský tým se soustředil na plnění úkolů.

Se zadavatelem jsme se dohodli, že produkt pilotně otestujeme ve dvou procesech: proces vyřizování objednávky a proces plnění vývojových úkolů.

Vyřízení objednávky může být zahájeno přijetím objednávky nebo podepsáním smlouvy, jež firma obdrží poštou. Poté, co tato hmotná dokumentace dorazí, je předána vedoucímu, který ji dále předkládá řediteli. Ředitel se detailně seznámí se smlouvou a pokud s jejími podmínkami souhlasí, dává svůj souhlas. V opačném případě, pokud nesouhlasí nebo pokud smlouva čeká na schválení déle než pět dní od přijetí vedoucím, je archivována pro případné budoucí potřeby. Pokud ředitel souhlasí, majitel firmy smlouvu podepisuje, čímž dojde k formálnímu uzavření kontraktu. V případě, že majitel není spokojený se smlouvou, vrátí ji zpět vedoucímu k přepracování podmínek.

Proces plnění vývojových požadavků probíhí následovně: Nejprve jsou definovány a specifikovány vývojové požadavky na základě analytické fáze a požadavků zákazníka. Tyto požadavky jsou následně vedoucím týmu transformovány do úkolů, které jsou přiřazeny jednotlivým vývojářům v týmu. Plnění vývojových požadavků probíhá iterativně. Jednotlivé úkoly jsou vývojáři vybírány z backlogu a implementovány. Po dokončení úkolů jsou provedeny interní kontroly kvality (code review, unit testy, integrační testy), následně jsou provedeny případné úpravy a opravy chyb. Po úspěšném absolvování jsou úkoly uzavírány a předány k dalšímu zpracování.

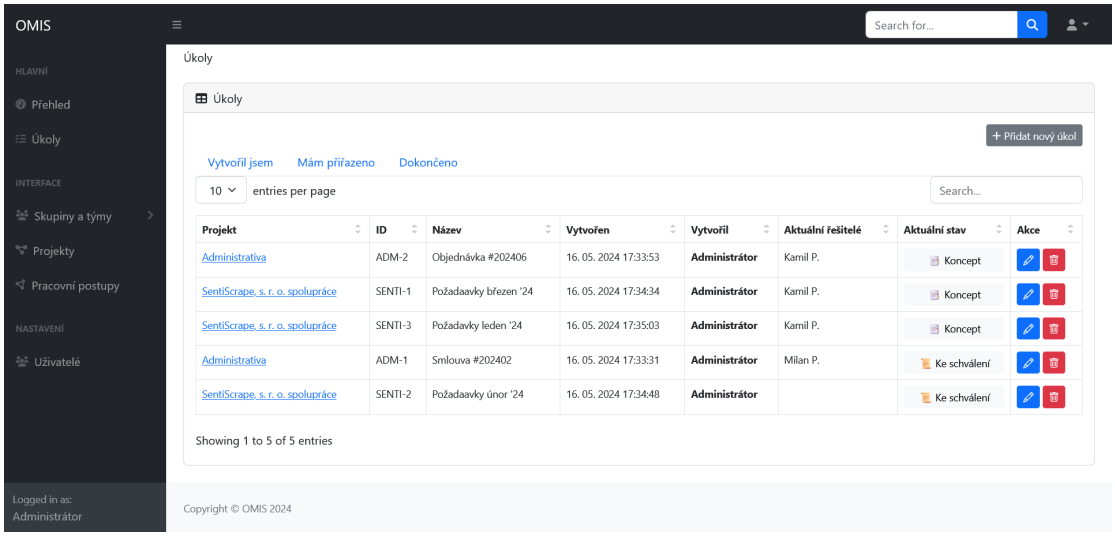


Obrázek 4.2 Obsazení pražského vývovového týmu

4.2.1 Průběh testování

V uživatelském testování se aktivně zapojilo všech šest dostupných testerů. Testování probíhalo na různých počítačích, aby byla zajištěna široká variabilita prostředí. Při navrhování testovacích scénářů jsem vycházel z případů užití, které byly identifikovány během analytické fáze projektu. Níže představím jednotlivé scénáře:

- **Založení úkolu**.
  - Správce se přihlásí do aplikace.
  - Přejde do sekce úkoly a klikne na tlačítko pro založení nového ukolu.
  - Vyplní vyžadovaná políčka formuláře.
  - Formulář potvrdí potvrzovacím tlačítkem.
- **Úprava úkolu**
  - Správce se přihlásí do aplikace.
  - Přejde do sekce úkoly a klikne na editační tlačítko u vybraného úkolu.
  - Vyplní vyžadovaná políčka formuláře.
  - Formulář potvrdí potvrzovacím tlačítkem.
  - Přejde do sekce úkoly a klikne na tlačítko smazání u vybraného úkolu.
- **Změna stavu úkolu**
  - Uživatel se přihlásí do aplikace.



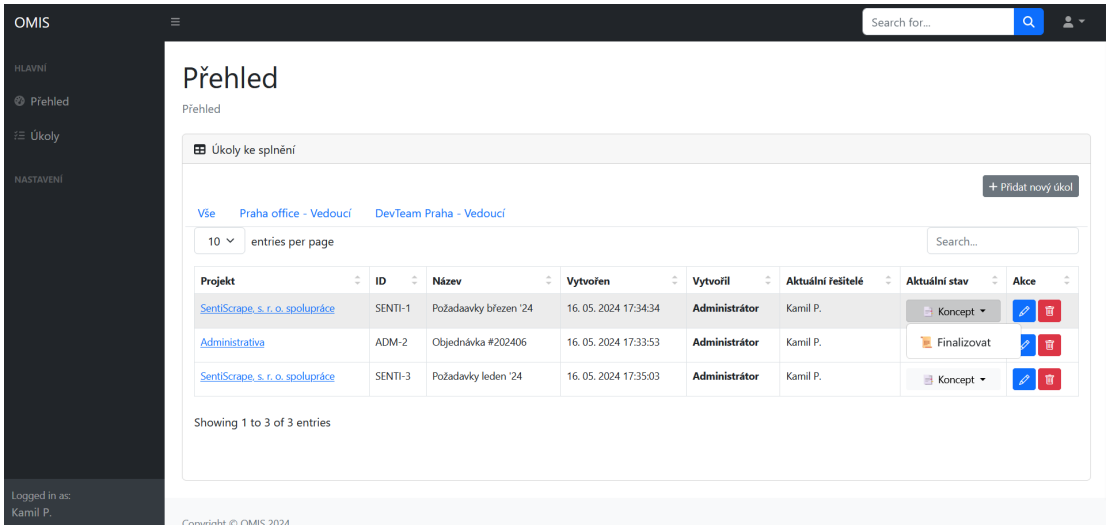
Obrázek 4.3 Snímek obrazovky aplikace z pohledu administrátora

- Přejde do sekce úkoly, vybere sekci Ke splnění a vybere pod kterou rolí chce úkol zobrazit.
- Prohlédne informace o stavu u vybraného úkolu.
- Klikne na rozvalovací seznam vedle informace o stavu u vybraného úkolu.
- Vybere požadovanou akci.

Během testování jsem se postaral o dokumentaci prostřednictvím zachycení dvou snímků obrazovky (obrázky 4.3, 4.4), které ilustrují seznam úkolů v různých uživatelských rozhraních. Tyto snímky poskytují pohled na aktuální stav úkolů a jejich prezentaci uživatelům.

4.2.2 Výsledky testování

Během testování bylo potvrzeno bezproblémové fungování aplikace v souladu se stanovenými požadavky. Všechny testovací scénáře byly úspěšně absolvovány a aplikace prošla bez významných nedostatků. Všichni zúčastnění na testování vyjádřili svoji spokojenost s aplikací. I přesto, že se v průběhu testování objevily drobné nedostatky, celkový dojem z aplikace jimi nebyl ovlivněl. Výsledkem je pozitivní hodnocení aplikace ze strany všech zúčastněných a očekávání brzkého nasazení.



Obrázek 4.4 Snímek obrazovky aplikace z pohledu aktivního uživatele

Kapitola 5
Zhodnocení

V této kapitole zhodnotím použití mnou navrhovaného řešení, porovnám s existujícími alternativami a možnostmi dalšího rozšíření.

5.1 Možnosti reálného použití

Workflow procesor, cíl této práce, je vhodný pro nasazení v segmentu malých a středních podniků, kde může efektivně automatizovat a řídit různé pracovní procesy. Díky své flexibilitě a konfigurovatelnosti je schopen pokrýt širokou škálu úloh a oblastí podnikání. Byl navržen a implementován s využitím moderních technologií, což zajišťuje jeho snadnou integraci do existujících nebo nově vznikajících systémů. Tato flexibilita umožňuje firmám přizpůsobit workflow procesor specifickým potřebám a požadavkům, což v konečném důsledku přináší zvýšení produktivity, efektivity a konkurenceschopnosti podniku.

5.1.1 Porovnání s existujícími řešeními

Prototyp workflow processoru, implementovaný v této práci, přináší výhody především v oblasti uživatelské přívětivosti. Uživatel nemusí mít detailní znalost podmínek a pravidel pracovního procesu, aby se v něm pohyboval. Díky tomu odpadá nutnost provádět rozsáhlá školení a dlouhodobé instruktáže, které mohou být časově náročné a neefektivní. Za svou praxi jsem byl svědkem mnoha případů, kdy uživatelé ztráceli přehled a často se stávalo, že nesprávně interpretovali pracovní postupy. Workflow processor tento problém úplně eliminuje tím, že pracovní proces automatizuje a zajišťuje jeho spolehlivý průběh bez nutnosti manuálního zásahu uživatele. Díky tomu může uživatel pracovat efektivněji a bez obav z chyb ve správném vykonávání úkolů.

5.2 Možnosti rozšíření

V rámci návrhu a implementace spojovacího systému jsem vytvořil funkční mechanismy pro přepínání stavů na základě uživatelské role a časového limitu. Nicméně, cílem je vytvořit systém, který bude flexibilní a schopen se přizpůsobit různým podmínkám a potřebám uživatelů. Proto je jedním z možných směrů rozšíření systému implementace

dalších typů podmínek pro přepínání stavů. To může zahrnovat například podmínky závislé na konkrétních vlastnostech úkolu, podmínky vyžadující interakci více uživatelů nebo podmínky prováděné na základě externích událostí. Tato rozšíření by umožnila uživatelům lépe modelovat a řídit pracovní procesy podle specifických potřeb jejich projektů.

Kromě toho je další možností rozšíření systému integrace mezitýmové komunikace. Momentálně systém umožňuje přepínat mezi různými stavy uvnitř jednoho workflow. Nicméně, v praxi se často setkáváme s potřebou koordinace mezi různými týmy nebo pracovními skupinami. Proto je možné zvážit rozšíření systému o možnost přepínání stavů mezi různými workflow, které reprezentují práci různých týmů. Tím by se vytvořily nové možnosti pro spolupráci a koordinaci mezi různými částmi organizace, což může vést k efektivnějšímu a plynulejšímu průběhu pracovních procesů.

Kapitola 6
Závěr

Cílem mé práce bylo provést analýzu, navrhnout, prototypově implementovat a otestovat workflow processor pro stávající manažerský informační systém „Operativní Manažerský Informační Systém“ (OMIS).

V rámci bakalářské práce jsem provedl analýzu vzorku stávajících řešení. Z tohoto rozboru vyplývá, že stávající řešení nedostatečně vyhovují uživatelským požadavkům. Na základě těchto zjištění jsem navrhl architekturu a funkcionalitu nového workflow processoru, který by byl schopen efektivně zpracovávat pracovní procesy v rámci manažerského systému OMIS.

Před samotnou implementací jsem na základě analýzy požadavků vybral technologie, na kterých prototyp postavím. Zohlednil jsem zejména požadavky na robustnost, škálovatelnost a snadnou integraci s existujícím systémem OMIS. Jako vhodný nástroj jsem zvolil programovací jazyk Java ve spojení s moderním frameworkem Spring pro tvorbu webových aplikací. Pro popis požadavků na prostředí, ve kterém bude aplikace nasazena, jsem použil technologii Docker.

Dále jsem vypracoval testovací strategii a provedl testování celého systému, včetně integračních testů a uživatelských testů. Pro uživatelské testy jsem připravil testovací scénáře, které byly následně otestovány zadavatelem. Dokumentaci API rozhraní jsem vytvořil ve standardizovaném formátu OpenAPI, který byl automaticky generován ze zdrojového kódu aplikace.

Po úspěšném dokončení implementace a otestování jsem prototyp předal zadavateli k manuálnímu testování. Zadavatel hodnotil prototyp kladně a plánuje jeho nasazení v produkčním prostředí. Dále se zvažuje možnost dalšího rozšíření a zdokonalení prototypu na základě uživatelských zpětných vazeb a požadavků zákazníků.

Bibliografie

1. LUKASÍK PETR, Procházka Jaroslav. *Procesní řízení: Text pro distanční studium*. Ostravská univerzita, [b.r.]. Dostupné také z: https://web.archive.org/web/20131228075751/http://www1.osu.cz/~prochazka/rpri/skripta_ProcesniRizeni.pdf.
2. IVO, Vondrák. *Metody byznys modelování pro kombinované a distanční studium*. Vysoká škola báňská - Technická univerzita Ostrava, [b.r.]. Dostupné také z: http://vondrak.cs.vsb.cz/download/Metody_byznys_modelovani.pdf.
3. ATLASSIAN, Inc. *JIRA Server platform REST API reference*. 2020. Dostupné také z: https://docs.atlassian.com/jira/REST/server/.
4. ATLASSIAN, Inc. *Integrating with Jira Server*. 2020-12-01. Dostupné také z: https://developer.atlassian.com/server/jira/platform/integrating-with-jira-server/.
5. *Themes vs. Epics vs. Stories vs. Tasks in Scrum: Aha! software*. 2023-09-29. Dostupné také z: https://www.aha.io/roadmapping/guide/agile/themes-vs-epics-vs-stories-vs-tasks.
6. ATLASSIAN, Inc. *Workflows and statuses for the board: Jira Work Management Cloud*. 2023. Dostupné také z: https://support.atlassian.com/jira-work-management/docs/workflows-and-statuses-for-the-board/.
7. LEARN, Microsoft. *Add custom work item type to inherited process - Azure DevOps Services*. 2023. Dostupné také z: https://learn.microsoft.com/en-us/azure/devops/organizations/settings/work/add-custom-wit?view=azure-devops.
8. TEAM, MantisBT Development. *Admin Guide: Reference for Administrators*. 2016. Dostupné také z: https://mantisbt.org/docs/master/en-US/Admin_Guide/Admin_Guide.pdf.
9. BV., TIOBE SOFTWARE. *TIOBE Index for January 2024* [online]. 2024. [cit. 2024-01-03]. Dostupné z: https://www.tiobe.com/tiobe-index/.
10. *Spring Framework Overview* [online]. 2024. [cit. 2024-01-03]. Dostupné z: https://docs.spring.io/spring-framework/reference/overview.html.

11. BUSH, Thomas. *CRUD vs. REST: What's the Difference?* [online]. 2023. [cit. 2024-01-03]. Dostupné z: `https://nordicapis.com/crud-vs-rest-whats-the-difference/`.

12. AMAZON WEB SERVICES, Inc. *What is a RESTful API?* [online]. 2024. [cit. 2024-01-03]. Dostupné z: `https://aws.amazon.com/what-is/restful-api/`.

13. *MAPSTRUCT 1.6.0.beta2 reference guide* [online]. 2024. [cit. 2024-01-03]. Dostupné z: `https://mapstruct.org/documentation/1.6/reference/html/`.

14. *GreenMail* [online]. 2024. [cit. 2024-01-03]. Dostupné z: `https://greenmail-mail-test.github.io/greenmail/`.

Obsah příloh

- `README.md` ..... stručný popis obsahu média
- `docker-compose.yml` konfigurace kontejnerů použitých k sestavení a spuštění práce
- `.gitlab-ci.yml` ..... konfigurace pro Gitlab CI/CD
- `.editorconfig` ..... konfigurace formátování pro editor
- `exe/` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl/` ..... zdrojové kódy implementace
  - `thesis/` ..... zdrojová forma práce ve formátu LaTeX
- `text/` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF