

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

WEB APLIKÁCIA NA ALOKOVANIE ÚLOH PRE
RIEŠITEĽOV NA ZÁKLADE ICH KVALIFIKÁCIE A
PREFERENCIÍ
BAKALÁRSKA PRÁCA

2025
TOMÁŠ MAGULA

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

WEB APLIKÁCIA NA ALOKOVANIE ÚLOH PRE
RIEŠITEĽOV NA ZÁKLADE ICH KVALIFIKÁCIE A
PREFERENCIÍ
BAKALÁRSKA PRÁCA

Študijný program: Aplikovaná Informatika
Študijný odbor: Informatika
Školiace pracovisko: Katedra Aplikovanej informatiky
Školiteľ: Mgr. Peter Náther, PhD.

Bratislava, 2025
Tomáš Magula



Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Tomáš Magula
Študijný program: aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Web aplikácia na alokovanie úloh pre riešiteľov na základe ich kvalifikácie a preferencií
Web application for allocating tasks to assignees based on their qualifications and preferences

Anotácia: Aplikácia by mala slúžiť pre manažérov na priradovanie úloh pre riešiteľov na základe ich voľného času, kvalifikácie na danú úlohu ale aj ich preferencie tak aby jednotlivé úlohy boli splnené do zadaného času, resp. v správnom poradí

Cieľ: Aplikácia by mala slúžiť pre manažérov na priradovanie úloh pre riešiteľov na základe ich voľného času, kvalifikácie na danú úlohu ale aj ich preferencie tak aby jednotlivé úlohy boli splnené do zadaného času, resp. v správnom poradí

Vedúci: Mgr. Peter Náther, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: doc. RNDr. Tatiana Jajcayová, PhD.
Dátum zadania: 10.10.2024

Dátum schválenia: 14.10.2024
doc. RNDr. Damas Gruska, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Pod'akovanie: Na tomto mieste by som chcel poďakovať svojmu školiťovi Mgr. Peter Náther, PhD. za odborné vedenie a cenné rady.

Čestne vyhlasujem, že celú bakalársku prácu na tému Web aplikácia na alokovanie úloh pre riešiteľov na základe ich kvalifikácie a preferencií, vrátane všetkých jej príloh a obrázkov, som vypracovala samostatne, a to s použitím literatúry uvedenej v priloženom zozname a nástrojov umelej inteligencie. Vyhlasujem, že nástroje umelej inteligencie som použila v súlade s príslušnými právnymi predpismi, akademickými právami a slobodami, etickými a morálnymi zásadami za súčasného dodržania akademickej integrity.

Bratislava, 2025

.....

Tomáš Magula

Abstrakt

Táto bakalárska práca sa venuje odstráneniu neefektívneho priradovania účastníkov k úlohám pri plánovaní športových vysielaní a iných podujatí. Na tento účel bol pre systém Redmine vytvorený plugin *Eventer*, ktorý automaticky zostavuje rozvrh s ohľadom na zručnosti, dostupnosť a preferencie zamestnancov. Jadro riešenia tvorí model celočíselného lineárneho programovania (ILP) implementovaný v knižnici PuLP so solverom CBC, rýchla greedy heuristika slúži ako záložný prístup, ak ILP nedokáže nájsť úplné riešenie. Plugin je integrovaný prostredníctvom Redmine Hooks a Patches, čo umožňuje doplniť evidenciu voľných dní a hodnotenie kvalifikácií bez zásahu do jadra systému. Testovaním na dátach sa potvrdilo, že ILP spoľahlivo spĺňa všetky obmedzenia a nachádza optimálne výsledky. Riešenie je navrhnuté univerzálne, takže ho možno prispôbiť rôznym typom podujatí.

Kľúčové slová: automatizované plánovanie, Redmine plugin, celočíselné lineárne programovanie, športové vysielanie

Abstract

This bachelor's thesis focuses on eliminating inefficient assignment of participants to tasks in the planning of sports broadcasts and other events. To this end, a plugin named *Eventer* was developed for the Redmine system, which automatically generates schedules while considering employees' skills, availability, and preferences. The core of the solution is an Integer Linear Programming (ILP) model implemented using the PuLP library with the CBC solver, while a fast greedy heuristic serves as a fallback approach if the ILP fails to find a complete solution. The plugin is integrated through Redmine Hooks and Patches, enabling the addition of features such as tracking days off and assessing qualifications without modifying the system's core. Testing on data confirmed that the ILP reliably meets all constraints and produces optimal results. The solution is designed to be versatile, allowing adaptation to various types of events.

Keywords: automated scheduling, Redmine plugin, integer linear programming, sports broadcasting

Obsah

Úvod	1
1 Úvod do problematiky	3
1.1 Motivácia	3
1.2 Analýza existujúcich softvérových riešení	4
1.2.1 Jira	4
1.2.2 Cvent	4
1.2.3 Ganttice	5
1.2.4 Redmine	5
1.2.5 Záver	6
2 Návrh	7
2.1 Interface	7
2.2 Dátový model	8
2.3 Spôsob priradenia úloh	10
2.4 Uživatelské prostredie (UI/UX)	10
2.4.1 Mockupy	11
3 Spôsob priradenia úloh	15
3.1 Úvod do algoritmu priradovania	15
3.2 Definícia problému	15
3.3 Počiatočné algoritmické prístupy	16
3.3.1 Základný algoritmus	16
3.3.2 Greedy algoritmus	17
3.3.3 Backtracking algoritmus	18
3.4 Priradenie na základe ILP	19
3.4.1 Prečo ILP?	19
3.4.2 Knižnica a optimalizačný modul	19
3.4.3 Modelovacia logika	20
3.4.4 Logika	22
3.4.5 Optimalita a Zložitosť	23

3.4.6	Testovanie	24
3.5	Porovnanie	25
4	Architektúra	29
4.1	Redmine	29
4.1.1	Identifikované medzery vo funkciách Redmine	30
4.1.2	Integrácia vlastných funkcií pomocou Redmine Hooks a Patches	31
4.2	Ruby on Rails	32
4.3	Dátová vrstva a databáza	33
4.4	Služba na pridelovanie úloh	34
4.5	Prostredie nasadenia	36
5	Implementácia	37
5.1	Štruktúra a nastavenie pluginu	37
5.1.1	Inicializácia pluginu a integrácia menu	37
5.1.2	Konfigurácia ciest (Routes)	39
5.2	Rozšírenia vrstvy modelu a databázy	40
5.2.1	Vlastné migrácie databáz	40
5.2.2	Model vlastných preferencií	41
5.3	Úprava komponentov Redmine	41
5.4	Integrácia používateľského rozhrania cez Hooks	43
5.5	Integrácia a implementácia algoritmu priradovania	44
6	Vývoj a testovanie	45
6.1	Prehľad funkcionalít pluginu Eventer	45
6.2	Používateľské testovanie	47
	Záver	49
	Príloha A	53

Zoznam obrázkov

2.1	UML Use-case diagram	8
2.2	Entitno-relačný diagram	10
2.3	Mockup - Dni voľna	11
2.4	Mockup - Formulár preferencií	12
2.5	Mockup - Nová úloha	12
2.6	Mockup - Spustenie priradenia	13
4.1	MVC diagram	32
4.2	Zjednodušená databázová schéma - ukážka rozšírenia	34
4.3	UML Sekvenčný diagram - Komunikácia s API	35
5.1	Navigácia	38
6.1	Ukážka logu priradenia	45
6.2	Ukážka zoznamu priradených úloh	46
6.3	Ukážka detailu úlohy	46

Zoznam tabuliek

1.1	Porovnanie existujúcich softvérov	6
3.1	Porovnanie výkonu algoritmov	26

Úvod

V dnešnej dobe sú organizačné schopnosti a efektívne spravovanie používateľov na rôznych typoch úloh kľúčové pre úspešnú realizáciu projektov. Organizátori musia často čeliť výzvam, ako zabezpečiť to, že používatelia budú správne priradení k úlohám odpovedajúcim ich zručnostiam, preferenciám a dostupnosti. Táto bakalárska práca sa zameriava na vývoj inovatívneho softvéru, ktorý využíva algoritmy na efektívne pridelovanie úloh.

V súčasnosti je administratívna organizácia zapojená do manažmentu úloh na mojom existujúcom pracovisku vysoko neefektívna a chýba jej automatizácia. Všetky údaje a úlohy súvisiace s používateľmi sa zadávajú manuálne pomocou tabuliek, čo zaberá značný čas na riadenie a často vedie k chybám v dôsledku veľkého objemu údajov a nejasného, ťažkopádneho systému.

Na vyriešenie týchto problémov bola vykonaná analýza na vyhodnotenie niekoľkých existujúcich softvérových riešení. Každý preskúmaný softvér vyniká v špecifických aspektoch, žiadny však neponúka mechanizmus na efektívne, automatizované riadenie úloh a pridelovanie používateľov.

Na prekonanie týchto obmedzení bol potrebný vývoj špecializovaného softvérového riešenia zameraného na automatizáciu priradovania, ktorý zvýši spokojnosť na strane organizátorov aj používateľov. Softvérovému nástroju možno pridať rôzny typ úloh, ako projekty, pracovné aktivity, vysielania atď., pričom sa vezmú do úvahy stanovené požiadavky a preferencie.

Používatelia aj organizátori majú jednoduchý prístup k softvéru cez webové rozhranie. Organizátori môžu vytvárať, spravovať úlohy a definovať kvalifikácie používateľov. Tí môžu uviesť svoje preferencie a ďalšie dôležité údaje na to, aby softvér efektívne rozdelil prácu.

Práca sa zameriava na vývoj algoritmu, ktorý rýchlo nájde čo možno najlepší výsledok z poskytnutých dát a špecifikácií. Aby algoritmus mohol pracovať s rôznymi typmi úloh a prispôbiť sa meniacim požiadavkám organizátorov. Okrem toho je dôležité zabezpečiť, aby algoritmus zohľadňoval individuálne preferencie používateľov. Toto umožní dosiahnuť ešte kvalitnejšie výsledky projektov.

Táto iniciatíva má schopnosť výrazne zvýšiť účinnosť a spokojnosť všetkých zainteresovaných strán tým, že sa zlepší organizácia a spravovanie úloh. Tento softvér by

mohol nájsť využitie vo viacerých oblastiach a priniesť svet organizovania úloh novú úroveň automatizácie a optimalizácie.

Kapitola 1

Úvod do problematiky

Efektívne rozdelenie používateľov na úlohy je nevyhnutné pre hladký priebeh a celkový úspech podujatí. Existujúci manuálny prístup k riadeniu používateľov má často za následok neefektívnosť, administratívne chyby a nespokojnosť v dôsledku zlého zosúladenia úloh s individuálnymi schopnosťami a preferenciami. Odpoveďou na tieto nedostatky je potreba softvéru schopného automatizovať pridelovanie úloh pri zohľadnení špecifických zručností, preferencií a dostupnosti každého používateľa.

Ideálne softvérové riešenie tohto problému by malo ponúkať rozsiahle možnosti prispôbenia, ktoré organizátorom umožnia dynamicky definovať požiadavky na úlohy a preferencie používateľov. Musí podporovať pokročilé funkcie, ktorými sú hodnotenie používateľov administrátormi, mechanizmy priradovania založené na zručnostiach a preferenciách a takisto prispôsobivosť v reálnom čase. Tieto možnosti by výrazne zefektívnili proces riadenia úloh.

1.1 Motivácia

Motivácia, ktorá stojí za týmto problémom, pramení z praktických výziev, ktoré zažívam na mojom súčasnom pracovisku, v televíznom a športovom vysielaní. Naša spoločnosť vysiela viacero športových líg, z ktorých každá organizuje minimálne jedno kolo za týždeň. V dôsledku toho sa často zaoberáme plánovaním a prideleniami personálu na viac ako 30 zápasov za jeden deň, čo zahŕňa rôzne úlohy, ako sú kameramani, komentátori, režiséri, technici a ďalší personál. V súčasnosti je tento proces riadenia značne chaotický a neefektívny. Napríklad dni voľna sú koordinované prostredníctvom mesačných e-mailov, kde manažéri následne manuálne zapisujú odpovede do tabuliek. Okrem toho pri pridelovaní personálu do zápasov musia manažéri zvažovať rôzne komplexné faktory a závislosti, ako je kompatibilita zručností a dostupnosť. Preferencie personálu, ako napríklad individuálne požiadavky na čas alebo kolegov, či náhle zmeny, ešte viac komplikujú tento už aj tak chaotický manuálny proces.

Tieto administratívne neefektívnosti nielen znižujú spokojnosť personálu, ale negatívne ovplyvňujú aj kvalitu výroby, pretože vznikajú nesprávne priradenia a konflikty v plánovaní. Odhalenie týchto nedostatkov poukazuje na potrebu implementácie automatizovaného a systematického prístupu. Správne softvérové riešenie by mohlo podstatne zefektívniť procesy pridelovania úloh, znížiť administratívnu záťaž, znížiť chybovosť a výrazne zlepšiť spokojnosť zamestnancov aj celkovú kvalitu výroby.

1.2 Analýza existujúcich softvérových riešení

V súčasnom prostredí manažmentu úloh je efektívne pridelovanie používateľov rozhodujúce pre úspech akéhokoľvek podujatia či projektu. Tento proces zabezpečuje, že každý používateľ je zapojený do práce, podľa jemu odpovedajúcich schopností, kvalifikácií a preferencií, čím sa zvyšuje celková spokojnosť aj kvalita. Rôzne softvérové riešenia boli vyvinuté na zlepšenie a automatizáciu tohto procesu, z ktorých každé je navrhnuté na rôzne pole pôsobnosti a ponúka iné vlastnosti. Táto analýza sa ponorí do štyroch prominentných platforiem - Jira, Cvent, Ganttlic a Redmine - aby sa vyhodnotila ich účinnosť pri pridelovaní používateľov v kontexte riadenia úloh.

1.2.1 Jira

Jira, vyvinutá spoločnosťou Atlassian, je robustný nástroj na riadenie projektov predovšetkým v odvetví softvérového inžinierstva. Ponúka možnosti na správu pracovných postupov, manažovanie spolupráce v reálnom čase a integráciu s rôznymi známymi vývojovými nástrojmi.

Keďže ale Jira je program navrhnutý priamo pre vývoj softvéru, chýbajú viaceré špecifické funkcie ako pre udalosti, tak aj pre užívateľov. Napriek týmto obmedzeniam sa Jira snaží o flexibilitu a integráciu doplnkov tzv. „pluginov“. Napríklad plugin *Plánovač udalostí pre Jira* umožňuje používateľom vytvárať a upravovať udalosti v rámci väčších celkov a ponúka jasný prehľad o nadchádzajúcich udalostiach. Tak ako táto integrácia, tak aj celkový softvér sa primárne zameriava skôr na plánovanie než na priradovanie účastníkov na základe ich zručností a preferencií.[1]

1.2.2 Cvent

Cvent je všestranná platforma na správu udalostí navrhnutá tak, aby zvládala veľké podujatia. Poskytuje sadu nástrojov na registráciu, výber miesta konania a riadenie účastníkov. Silnými stránkami sú: možnosť prispôsobenia materiálov udalostí tak, aby boli v súlade s značkou organizácie, široká podpora rôznych jazykov a bezproblémová integrácia s rôznymi CRM systémami a platformami marketingovej automatizácie, čo

zvyšuje účinnosť práce. Pokročilé možnosti vytvárania prehľadov od spoločnosti Cvent ponúkajú detailný prehľad o výkonnosti na jednotlivých podujatiach a prostredníctvom mobilnej platformy je možné jednoduchšie zapojenie účastníkov do optimalizácie interných procesov plánovania.

Aj keď Cvent ponúka robustné nástroje na organizáciu a realizáciu udalostí, neponúka podrobné funkcie priradovania účastníkov potrebné na zosúladenie úloh s individuálnymi zručnosťami a preferenciami.[4]

1.2.3 Gantt

Gantt je výkonný softvér na plánovanie zdrojov, ktorý sa zameriava na efektívne prideľovanie ľudí, zariadení a priestorov k úlohám pomocou vizuálnych Ganttových diagramov. Ponúka pokročilé funkcie, ako je prispôsobiteľné plánovanie založené na zručnostiach a dostupnosti, drag-and-drop rozhranie na rýchle priradovanie úloh a rôzne pohľady (Gantt, Kanban, kalendár) na správu pracovných tokov. Gantt umožňuje definovať vlastné dátové polia pre zdroje, čo uľahčuje priradovanie úloh podľa špecifických požiadaviek, ako sú kvalifikácie alebo pracovné hodiny.

Napriek týmto silným stránkam Gantt nepodporuje pluginy, čo obmedzuje jeho rozšíriteľnosť a prispôbenie špecifickým potrebám, ako je pokročilá kategorizácia úloh podľa komplexných kritérií. Ďalším nedostatkom je obmedzená podpora zadávania individuálnych preferencií používateľov, ako napríklad preferovaných kolegov alebo časových slotov. Tieto obmedzenia bránia plnému využitiu potenciálu automatizovaného prideľovania úloh v dynamickom prostredí s častými zmenami.[6]

1.2.4 Redmine

Redmine je open source aplikácia na správu projektov, ktorá ponúka funkcie, ako je sledovanie problémov, wiki projektu, fóra a sledovanie času. Jeho flexibilita a rozšíriteľnosť z neho robia vhodnú voľbu pre rôzne potreby projektového manažmentu, vrátane riadenia udalostí. Umožňuje prispôbenie pracovných tokov tak, aby vyhovovali špecifickým požiadavkám. Uľahčuje prideľovanie rôznych povolení rôznym rolám používateľov, čím sa zvyšuje bezpečnosť a organizácia. Veľkou výhodou je podpora pluginov, či už existujúcich, tak aj vlastných. Pridané návody na vytvorenie jednoduchého základu pluginu sú perfektnou pomocou pri prispôbovaní na špecifické potreby.

Hoci základné funkcie Redmine nie sú primárne určené na správu udalostí a nespĺňajú na 100 % všetky naše potreby, jeho open-source povaha a rozšíriteľnosť prostredníctvom pluginov poskytujú značnú flexibilitu. To umožňuje prispôbiť systém rôznym špecifickým potrebám a využiť ho na mnoho rozmanitých účelov, čo z neho robí životaschopnú voľbu aj pre projekty, ktoré vyžadujú správu udalostí.[10]

Identifikované medzery a zdôvodnenie nového riešenia

Tabuľka 1.1: Porovnanie existujúcich softvérov

Feature	Jira	Cvent	Gantt	Redmine
Manažment účastníkov	Limitovaný	Áno	Áno	Limitovaný
Prideľovanie úloh (Znalosti)	Nie	Nie	Áno	Nie
Prispôsobiteľnosť pracovného toku	Áno	Limitovaný	Áno	Áno
Podpora pluginov	Áno	Nie	Nie	Áno
Open-Source	Nie	Nie	Nie	Áno

Analýza týchto platforiem odhaľuje spoločné medzery, ktorými sú nedostatok špecializovaných funkcií na priradovanie používateľov k úlohám na základe individuálnych zručností, preferencií a dostupnosti. Existujúce riešenia vynikajú v oblastiach, ako je registrácia, predaj vstupeniek a všeobecný projektový manažment, ale zaostávajú pri automatizácii procesu zosúladenia úloh medzi používateľmi.

Vývoj špecializovaného softvérového riešenia, ktoré vyplní tieto medzery, môže výrazne zvýšiť efektivitu plánovania a realizácie úloh. Takéto riešenie by zautomatizovalo proces prideľovania, znížilo administratívne zaťaženie a zlepšilo spokojnosť používateľov tým, že by sa zabezpečilo prideľovanie úloh v súlade s individuálnymi kompetenciami a preferenciami. Tento cielený prístup by zvýšil celkovú kvalitu podujatí a poskytol konkurenčnú výhodu v oblasti organizácie manažmentu úloh.

1.2.5 Záver

Zatiaľ čo platformy ako Jira, Cvent, Gantt a Redmine ponúkajú cenné nástroje pre rôzne aspekty riadenia udalostí a projektov, žiadna sa plne nezaobera špecifickými potrebami automatizovaného pridelenia úloh účastníkom na základe ich dát. To potvrdzuje potrebu vývoja špecializovaného softvérového riešenia, ktoré sa hladko integruje s existujúcimi procesmi správy udalostí a zároveň poskytuje flexibilitu a automatizáciu potrebnú na optimalizáciu zapojenia účastníkov pre úspech podujatia.

Kapitola 2

Návrh

Návrh je kritickým komponentom cyklu vývoja softvéru, ktorý výrazne ovplyvňuje efektivitu, spoľahlivosť a spokojnosť používateľov s konečným produktom. Dobrý návrh zabezpečuje, že softvér spĺňa funkčné aj nefunkčné požiadavky, podporuje škálovateľnosť a poskytuje maximálnu používateľskú skúsenosť.

Táto kapitola predstavuje komplexný prehľad navrhovaného systému. Načrtáva rôzne základné aspekty, ako je používateľské rozhranie(UI), na zabezpečenie intuitívnej interakcie, dátový model pre bezpečnú a efektívnu správu údajov, integrácia a používateľských preferencií pre presnejšie priradovanie, optimalizačný algoritmus na zabezpečenie efektívneho pridelovania zdrojov a backendové komponenty dôležité pre celkovú funkčnosť systému. Okrem toho sa venuje aj stratégiám nasadenia, aby sa zabezpečilo, že riešenie sa dá jednoducho aplikovať a prispôbovať meniacim sa požiadavkám a prostrediam.

2.1 Interface

Use-case organizátora

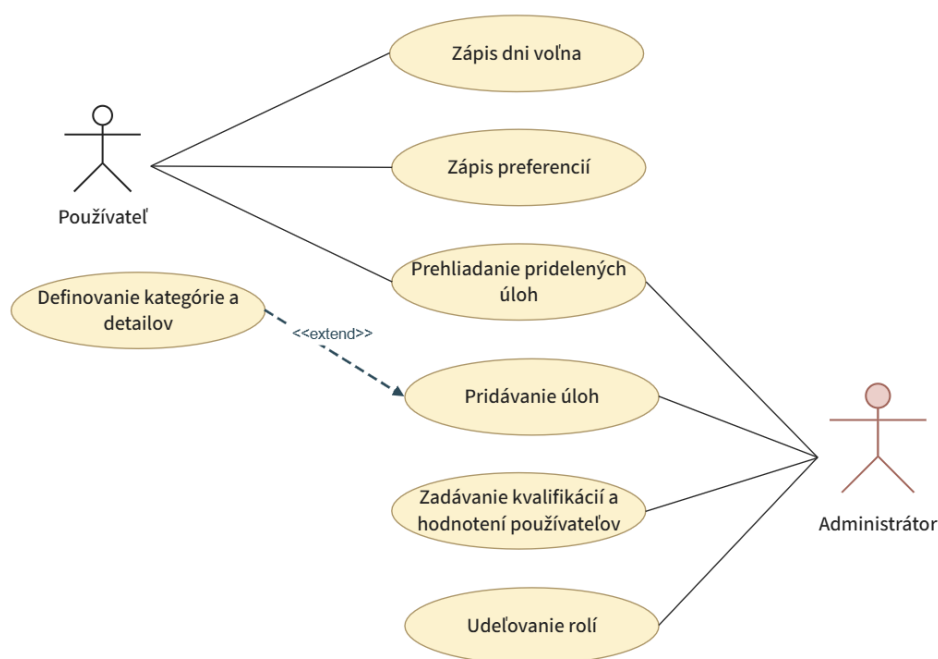
Organizátori či administrátori majú možnosť využiť nasledujúce funkcie:

- Vytváranie a riadenie úloh vrátane podrobných definícií a súvisiacich požiadaviek.
- Zadávať údaje používateľov, udeľovaním rolí, definovaním kvalifikácií a hodnotení jednotlivým užívateľom.
- Zobrazíť panely monitorovania ukazujúce stavy úloh.
- Možnosť spúšťať priraďovací algoritmus a následne vidieť výsledky priradenia.

Use-case účastníka

Rozhranie účastníka ponúka nasledujúce funkcie:

- Nastaviť dostupnosť, respektíve nedostupnosť v konkrétnych časových úsekoch.
- Zvoliť podmienené či nepodmienené osobné preferencie.
- Zobrazíť jasné prehľady pridelených úloh vrátane zodpovedností, rolí a súvisiacich informácií.



Obr. 2.1: UML Use-case diagram administrátora a účastníka

2.2 Dátový model

Efektívny dátový model je rozhodujúci pre správne ukladanie, vyhľadávanie a manipuláciu s údajmi. Vzhľadom na zložitosť správy viacerých vzájomne prepojených bodov, musí byť model robustný a schopný spracovávať súbežné transakcie a dotazy. Dobře navrhnutá štruktúra priamo prispieva k presnosti a rýchlosti algoritmu priradovania, čím výrazne zvyšuje celkový výkon systému.

Entity

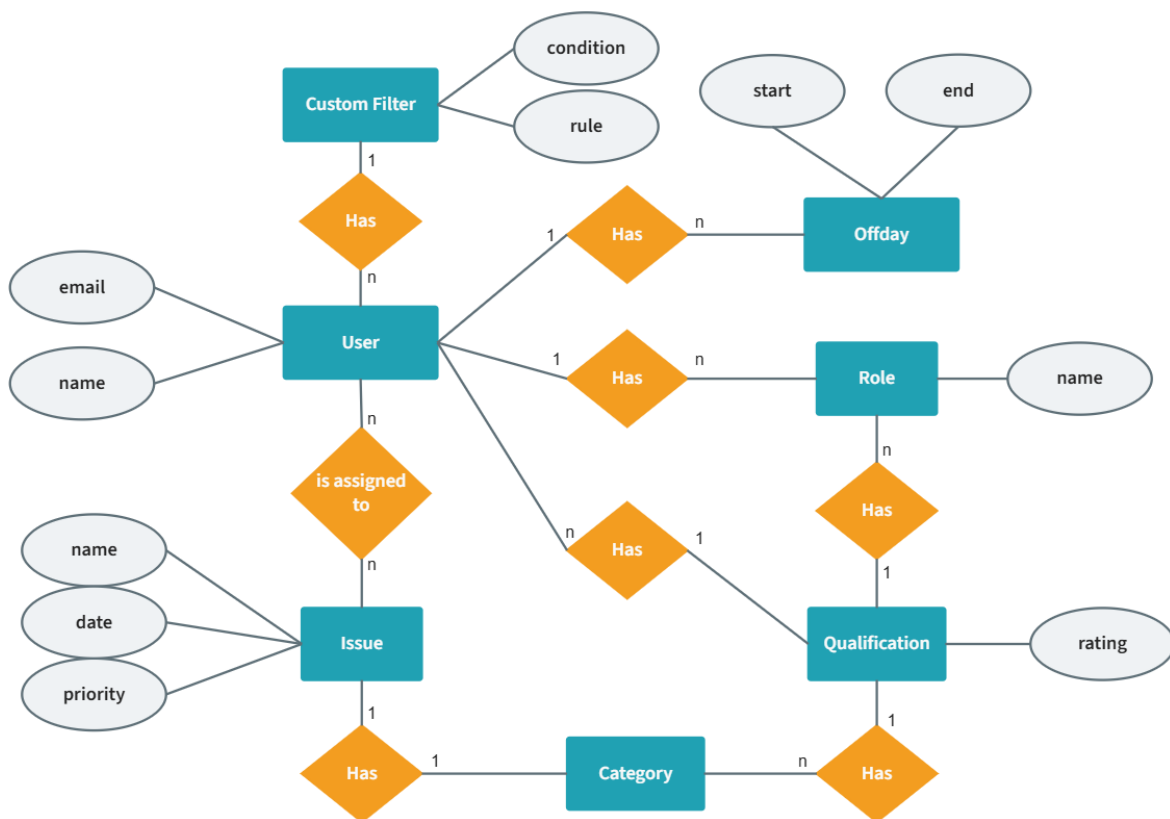
Entity predstavujú komponenty, v ktorých sú zapuzdrené nevyhnutné dátové prvky. Každá entita je navrhnutá tak, aby obsahovala odlišné kategórie informácií relevantné pre rôzne akcie a interakcie.

- **Používateľ**(User): Ukladá informácie o používateľoch vrátane osobných údajov, kontaktných informácií, priradených rolí a autentifikačných údajoch.
- **Rola**(Role): Definuje možné pozície alebo funkcie používateľov v systéme, povolenia úprav a zobrazení v rozhraní a názov ktorý špecifikuje typ role (napr. kameraman, komentátor).
- **Kvalifikácia**(Qualification): Udržiava údaje o roliach, zručnostiach a hodnoteniach vo vzťahu k užívateľom.
- **Preferencia**(Custom Filter): Zachytáva informácie špecifické pre každého užívateľa, týkajúce sa preferencií úloh.
- **Úloha**(Issue): Obsahuje informácie o kontexte úloh, dátumoch, mieste, vyžadovanom personáli a celkových parametroch.
- **Kategória**(Category): Slúži na triedenie úloh do skupín, s atribútom názov, ktorý definuje typ kategórie (napr. futbalový zápas, hokejový zápas).
- **Dni voľna** (Offday): Uchováva informácie o nedostupnosti používateľov, s atribútmi ako začiatok, koniec, ktorý označuje dátum alebo časový rozsah nedostupnosti.

Vzťahy

Jasne definované vzťahy medzi entitami zabezpečujú logickú konzistenciu a integritu údajov. Tieto vzťahy uľahčujú plynulú navigáciu vzájomne prepojených informácií.

- Používatelia môžu byť priradení ku viacerým úlohám, podobne aj úlohy môžu mať viacerých priradených používateľov.
- Každá preferencia má práve jedného používateľa, ale každý používateľ môže mať mnoho preferencií.
- Viacerí používatelia môžu byť kvalifikovaní na výkon tej istej role, rovnako aj konkrétny používateľ môže mať viacero rôznych kvalifikácií.
- Každý používateľ môže mať viacero dní voľna, pričom každý deň voľna je priradený iba jednému používateľovi.
- Každá úloha patrí do jednej kategórie, pričom jedna kategória môže zahŕňať viacero úloh.



Obr. 2.2: Entitno-relačný diagram ilustruje logickú dátovú štruktúru, ktorá je základom navrhovaného softvéru, jasne definuje entity, ich atribúty a vzťahy medzi nimi.

2.3 Spôsob priradenia úloh

Jadro systému sa spolieha na priraďovací algoritmus, ktorý priraďuje používateľov k problémom na základe vopred definovaných obmedzení. Pri určovaní optimálnych priradení zvažuje vhodnosť používateľa a požiadavky na problémy. Algoritmus zaisťuje logickú konzistenciu a rieši potenciálne konflikty, ako sú prekrývanie dostupnosti alebo nesúlad kvalifikácií. Vstupom algoritmu sú úlohy a používatelia s ich detailmi, výstupom sú úlohy s priradenými používateľmi, ak je to možné, alebo aspoň čiastočne priradené problémy. Bližšie sa venujeme otázke algoritmu v kapitole 3.

2.4 Uživatelské prostredie (UI/UX)

Pri navrhovaní používateľského rozhrania pluginu bolo primárnym cieľom poskytnúť intuitívne a efektívne prostredie pre manažerov aj bežných používateľov. Celkové rozloženie kladie dôraz na jasnosť a jednoduchosť s cieľom minimalizovať počet krokov potrebných na vykonávanie bežných akcií. Použitím jasných popisov, logickej postupnosti a konzistentnosti v navigácii sa rozhranie snaží znížiť krivku učenia a riziko chýb.

2.4.1 Mockupy

Nasledujúca časť ilustruje postupnosť úkonov v systéme, od nastavenia používateľov až po automatické pridelovanie úloh. Každý krok je navrhnutý tak, aby riešil konkrétnu funkčnosť a viedol k plynulému pracovnému procesu. Tieto formuláre sa nachádzajú na hlavnej lište používateľa a pre administrátorské úkony sa nachádzajú na administrátorskej lište, prípadne pri zozname všetkých úloh (pozri obr. 5.1).

Najprv si používatelia nastavujú svoje dni voľna. Návrh formulára, ktorý je zobrazený nižšie, sa použije na registráciu dní voľna alebo nedostupnosti používateľa. Formulár obsahuje dátum a čas začiatku a ukončenia voľna. Pod formulárom sa nachádza zoznam existujúcich dní voľna, každý s tlačidlom na odstránenie v prípade potreby.

New Off Day:

Start Date & Time:

Year Month Day - Hour : Time

End Date & Time:

Year Month Day - Hour : Time

Existing Off Days:

24-03-2025 from 10:39 to 19:39

24-03-2025 from 10:39 to 19:39

24-03-2025 from 10:39 to 19:39

Obr. 2.3: Mockup - Dni voľna

Ďalej používatelia vytvoria svoje vlastné preferencie. Formulár na nasledujúcom obrázku sa používa na vytvorenie vlastných preferencií. Obsahuje názov filtra a voliteľné časti *Podmienky* a *Pravidlá*, ktoré sa môžu kombinovať pomocou logických operátorov (AND / OR). Každý riadok podmienky sa skladá z hodnoty, porovnávacieho operátora (napríklad „=“), poľa (napr. „kategória“) a ďalšieho logického operátora (napr. „A ZÁROVEŇ“ alebo „ALEBO“). Existuje aj tlačidlo *+ Pridať podmienku*, aby bolo možné pridať viacero riadkov. Formulár sa odošle po nastavení všetkých podmienok a pravidiel.

Create new filter:

Filter name:

Conditions(optional):
☒ AND ☐ OR

Value == category AND

+ Add condition

Rules:
☒ AND ☐ OR

Value == category AND

+ Add condition

[Back](#)

Obr. 2.4: Mockup - Formulár preferencií

Potom administrátor zadá kvalifikácie používateľom a nahrá do systému konkrétne úlohy. Mockup nižšie umožňuje používateľovi vytvoriť novú úlohu zadaním počiatočného dátumu a času ukončenia. Zahŕňa štandardné polia, ako je názov, popis, priorita a kategória. Ponúka sekciu *Požadované roly*, kde si používateľ môže stanoviť rolu, uviesť, koľko ľudí je potrebných, prípadne priradiť k danej role dostupných používateľov. Má prepojenie *Pridať ďalšiu rolu*, aby bolo možné dynamicky pridať toľko sekcií s požadovanými rolami, koľko je potrebné. Tento formulár vychádza z existujúceho rozhrania Redmine a je doplnený o vstupy: čas začiatku, čas ukončenia a požadované roly.

New Issue:

Start Datetime:

End Datetime:

Name:

Description:

Priority:

Category:

Required Roles:

Role 1

Assignee



[Add another role](#)

Obr. 2.5: Mockup - Nová úloha

Na záver administrátor stlačí tlačidlo na automatické priradenie úloh, čím spustí optimalizačný algoritmus. Po dokončení behu algoritmu môže administrátor vidieť výsledky pridelovania, ktoré zohľadňujú dni voľna, preferencie používateľov a požiadavky úloh.

✓ Solution was found

☒ Enable Date Range

04/23/2025  04/23/2025 

☒ Allow partial solution if no complete solution is found

Assign Tasks

Show logs

Logs...

Obr. 2.6: Mockup - Spustenie priradenia

Kapitola 3

Spôsob priradenia úloh

3.1 Úvod do algoritmu priradovania

Priradenie používateľov k problémom a rolám sa môže rýchlo stať zložitým v dôsledku rôznych interakčných obmedzení. Scenáre reálneho sveta vyžadujú zváženie viacerých faktorov, ako sú prekrývajúce sa časy problémov, požadovaný počet používateľov na rolu, kvalifikácia používateľov, dostupnosť a prispôsobené podmienky špecifikované používateľmi.

Spočiatku sme problém riešili jednoduchšími metódami, ako jednoduchý základný algoritmus, greedy heuristika a backtrackingový algoritmus. Každý z týchto prístupov odhalil isté nevýhody, najmä pri riešení náročných závislostí alebo veľkých dátových tokov.

Ako vhodné riešenie sa ukázalo celočíselné lineárne programovanie (ILP), ktoré umožňuje efektívne zohľadniť komplexné obmedzenia. Jeden zo vstupov, vlastné preferencie používateľov, sa využívajú ako nástroj na zlepšenie celkovej kvality, preto sú v algoritme začlenené ako flexibilné obmedzenia (soft constraint).

Táto časť podrobne popisuje postup od prvých experimentov až po vývoj konečného riešenia, pričom zdôrazňuje integráciu flexibilných obmedzení prostredníctvom vlastných filtrov a vysvetľuje praktické výhody tohto nuansového prístupu.

3.2 Definícia problému

Cieľom je prideliť každej úlohe používateľov, ktorí majú potrebnú kvalifikáciu, sú práve k dispozícii a zároveň spĺňajú všetky definované obmedzenia. Vstupom je zoznam úloh (niektoré môžu byť už čiastočne obsadené) spolu so zoznamom používateľov a ich podrobnými údajmi. Výstupom je zoznam úloh doplnený o správne priradených používateľov.

Obmedzenia

Prísne obmedzenia:

- Používatelia musia mať zodpovedajúcu kvalifikáciu.
- Každá rola vyžaduje presne definovaný počet používateľov.
- Používateľ môže mať v konkrétnej úlohe priradenú najviac jednu rolu.
- Nie sú povolené prekrývajúce sa úlohy.
- Používateľ musí byť dostupný v naplánovanom čase.

Flexibilné obmedzenia:

- Vlastné používateľské preferencie ovplyvňujúce priradenia.

3.3 Počiatočné algoritmické prístupy

Pred prijatím celočíselného lineárneho programovania (ILP) sme preskúmali jednoduchšie stratégie priradovania používateľov k úlohám. Tieto prvé pokusy: základný, greedy a backtracking, nám pomohli pochopiť zložitosť problému a obmedzenia neoptimálnych metód. Každý prístup sa hodnotí najmä na základe jeho schopnosti zvládnuť obmedzenia uvedené v časti 3.2.

3.3.1 Základný algoritmus

Základný alebo naivný prístup využíva priame cykly na priradovanie používateľov k rolám, pričom kontroluje len tie najzákladnejšie obmedzenia, kvalifikáciu a dostupnosť. Ignoruje prekrytia, hodnotenia a vlastné filtre, čo z neho robí minimálnu základnú líniu.

Logika: Pre každý problém a rolu vyberieme prvého kvalifikovaného a dostupného, kým sa nenaplní požadovaný počet.

```
1 for each issue:
2     for each role needing count users:
3         assigned = empty set
4         for each user:
5             if user qualified for role and available at start:
6                 add user to assigned
7                 if assigned size equals count:
8                     stop
9         record assigned for issue and role
```

Silné stránky:

- Jednoduché na implementáciu a pochopenie.
- Funguje pre triviálne prípady bez problémov.

Slabé stránky:

- Zlá kontrola prekrývajúcich sa úloh.
- Ignoruje používateľské preferencie a hodnotenia.
- Žiadna záruka splnenia počtov rolí.

Tento prístup sa rýchlo považoval za nepraktický pre scenáre potrebné na pokrytie.

3.3.2 Greedy algoritmus

Greedy prístup vylepšuje základný algoritmus uprednostňovaním používateľov na základe ich kvalifikačných hodnotení, kontrolou prekrývania a preferencií. Spracúva problémy v poradí a ku každej roli priradzuje používateľov s najvyšším hodnotením, ktorí spĺňajú obmedzenia.

Logika: Zoradte problémy a potom pre každú rolu vyberte používateľov s najvyšším hodnotením, ktorí sú kvalifikovaní, dostupní, nie sú priradení v tom čase na iný problém a prešli preferenčnými filtrami (tu sa považujú za prísne obmedzenia).

```
1 initialize assignment as empty dictionary
2 for each issue in issues:
3     initialize assignment[issue.id] as empty dictionary
4
5     for each required_role in issue.required_roles:
6         set role = required_role.role
7         set count = required_role.required_count
8         initialize assigned as empty set
9
10        for each user in users:
11            if user qualified for role and available at issue.
               start_datetime:
12                add user.id to assigned
13                if assigned size equals count:
14                    stop
15            record assigned as assignment[issue.id][role]
16 return assignment
```


Silné stránky:

- Rýchle, so zložitou $O(N \log N)$ vďaka triedeniu.
- Rešpektuje viac obmedzení ako základný prístup.

Slabé stránky:

- Žiadna globálna optimalizácia, skoré voľby môžu blokovať neskoršie priradenia.
- Môže zlyhať pri hľadaní uskutočniteľného riešenia v scenároch s vysokým počtom konfliktov.
- Zaobchádzanie s filtrami ako s prísnyimi obmedzeniami obmedzuje flexibilitu.

Rýchlosť greedy algoritmu z nej urobila kandidáta na záložnú logiku, ale jej rozhodovanie podčiarklo potrebu globálneho prístupu.

3.3.3 Backtracking algoritmus

Backtracking používa rekurzívne vyhľadávanie do hĺbky (DFS) na vyskúšanie všetkých platných priradení, pokiaľ sa porušia obmedzenia, skúsi inú cestu. Systematicky skúma kombinácie používateľských rolí a ruší neúspešné pokusy.

Logika: Priradíte používateľov k každej role, skontrolujte obmedzenia a podľa potreby pokračujte alebo sa vráťte.

```
1 current_issue = first issue
2 while issues remain:
3     for each role in current_issue:
4         for each user:
5             if user qualified, available, not overlapping, and
               passes filters:
6                 assign user to role
7                 if role fully assigned:
8                     move to next issue
9                 if all issues assigned:
10                    return success
11            unassign user
12        if no users fit:
13            backtrack to previous issue
14 if no solution:
15     return failure
```

Silné stránky:

- Nájde správne riešenie, pokiaľ nejaké existuje
- Pokrýva všetky prísne obmedzenia

Slabé stránky:

- Exponenciálna zložitosť, nerealizovateľná pre rozsah.
- Môže prekročiť limit rekurzie pre Python.
- Prísne obmedzené filtre znižujú realizovateľnosť riešenia.

Výčerpávajúce vyhľadávanie pomocou backtrackingu by bolo pre produkciu príliš pomalé a riskantné.

3.4 Priradenie na základe ILP

Obmedzenia jednoduchších prístupov si vyžiadali robustnejšie riešenie. Celočíselné lineárne programovanie sa objavilo ako metóda, ktorá ponúka globálnu optimalizáciu a systematické zaobchádzanie s obmedzeniami. Táto časť vysvetľuje, prečo bol vybraný algoritmus ILP, použité nástroje a logiku.

3.4.1 Prečo ILP?

ILP modeluje problém priradenia ako súbor lineárnych rovníc a nerovností, ktoré sa riešia tak, aby sa maximalizovali hodnotenia kvalifikácie používateľov pri rešpektovaní všetkých obmedzení. Na rozdiel od predošlých algoritmov, ILP zaisťuje globálne optimálne riešenie, vyhýba sa limitom rekurzie a integruje flexibilné obmedzenia.

3.4.2 Knižnica a optimalizačný modul

Na vytvorenie modelu celočíselného lineárneho programovania (ILP) používame knižnicu PuLP [9]. Optimalizačný modul(solver) je nástroj, ktorý nachádza najlepšie riešenie matematických modelov, ako je priradenie používateľov k úlohám. Používame modul CBC (Coin-or Branch and Cut), ktorý rýchlo spracováva modely a pri zlyhaní generuje diagnostické súbory *.lp*, spúšťajúce záložný greedy algoritmus [2]. Modul optimalizuje sústavu lineárnych rovníc, systematicky testuje kombinácie priradení a vyvažuje hodnotenia s obmedzeniami, aby dosiahol optimálne riešenie [11].

3.4.3 Modelovacia logika

Celočíselné lineárne programovanie formalizuje priradenie používateľov k úlohám ako problém optimalizácie. Vyberá kvalifikovaných dostupných používateľov pre roly každého problému, maximalizuje hodnotenie kvalifikácie a zároveň zabráňuje prekryvaniu a rešpektuje počet rolí. Vlastné filtre fungujú ako flexibilné obmedzenia, ktoré ovplyvňujú preferencie bez vylúčenia používateľov. Model používa binárne premenné na reprezentáciu možností, kde cieľom je uprednostniť vysokokvalitné priradenia a obmedzenia presadzujúce pravidlá.

Použité premenné:

- U - množina všetkých úloh.
- P - množina všetkých používateľov.
- R_u - množina potrebných rolí v úlohe u .

Rozhodovacie premenné

$$x_{u,p,r} = \begin{cases} 1 & \text{ak používateľ } p \text{ je priradený k role } r \text{ v úlohe } u, \\ 0 & \text{inak,} \end{cases} \quad \forall u \in U, p \in P, r \in R_u$$

(Premenné)

Vysvetlenie: Premenná $x_{u,p,r}$ je ako prepínač: 1 znamená, že používateľ je priradený k určitej roli v úlohe, a 0 znamená, že nie je. Napríklad, ak je Eva vybraná ako „testerka“ pre úlohu č. 5, $x_{5,\text{Eva},\text{tester}} = 1$. To umožňuje modelu vyskúšať každú možnú kombináciu priradení na nájdenie najlepšieho.

Premenné pre porušenie filtrov

$$y_{u,p,r} = \begin{cases} 1 & \text{ak používateľ } p \text{ nespĺňa filtre pre rolu } r \text{ v úlohe } u, \\ 0 & \text{inak,} \end{cases} \quad \forall u \in U, p \in P, r \in R_u$$

(Porušenie filtrov)

Vysvetlenie: Premenná $y_{u,p,r}$ sleduje, či používateľ nespĺňa preferenčné filtre pre úlohu. Ak napríklad Eva nespĺňa filter (napr. preferované miesto), $y_{5,\text{Eva},\text{tester}} = 1$, čo spôsobí penalizáciu v cieľi. Toto pomáha modelu uprednostniť používateľov, ktorí lepšie vyhovujú, bez ich vylúčenia.

Cieľová funkcia

$$\text{Maximalizovať } \sum_{u \in U} \sum_{p \in P} \sum_{r \in R_u} (r_{p,r,c_u} \cdot x_{u,p,r} - \kappa \cdot y_{u,p,r}) \quad (\text{Cieľ})$$

Vysvetlenie: Cieľom je dosiahnuť najvyššie skóre priradení. Hodnotenie každého používateľa (r_{p,r,c_u}) závisí od jeho zručností pre rolu a kategóriu úlohy. Ak používateľ nespĺňa filtre, penalizácia $\kappa \cdot y_{u,p,r}$ (napr. $\kappa = 100$) znižuje jeho príspevok. Je to

ako výber najschopnejších ľudí, ale s malou pokutou, ak niekto nie je ideálny podľa preferencií, bez jeho vyradenia.

Omedzenie počtu rolí

$$\sum_{p \in P} x_{u,p,r} = c_{u,r}, \quad \forall u \in U, r \in R_u \quad (\text{Počet rolí})$$

Vysvetlenie: Toto zabezpečuje, že každá rola dostane presne toľko používateľov, koľko potrebuje ($c_{u,r}$). Ak úloha potrebuje dvoch vývojárov, model priradí presne dvoch. Je to naplnenie kvóty pre prácu, aby nebola rola ani prázdna, ani preplnená.

Omedzenie bez prekrývania

$$\sum_{u' \in U: u' \text{ sa prekrýva s } u} \sum_{r \in R_{u'}} x_{u',p,r} \leq 1, \quad \forall p \in P, u \in U \quad (\text{Bez prekrývania})$$

Vysvetlenie: Toto bráni priradeniu používateľa k rolám v úlohách, ktoré sa konajú súčasne. Ak sa dve úlohy prekrývajú, používateľ môže mať rolu iba v jednej. Je to ako zabezpečenie, aby nikto nebol naplánovaný na dve úlohy v rovnakom čase.

Omedzenie kvalifikácie

$$x_{u,p,r} = 0, \quad \text{ak } p \text{ nie je kvalifikovaný pre } r \text{ a kategóriu } c_u \quad (\text{Kvalifikácia})$$

Vysvetlenie: Toto bráni priradeniu používateľov bez potrebných zručností pre rolu a kategóriu úlohy. Ak rola vyžaduje znalosti sietí a používateľ ich nemá, je vylúčený. Je to ako zabezpečenie, aby iba kvalifikovaní technici pracovali na špecializovaných úlohách.

Omedzenie dostupnosti

$$x_{u,p,r} = 0, \quad \text{ak } p \text{ nie je dostupný v čase } u \quad (\text{Dostupnosť})$$

Vysvetlenie: Toto bráni priradeniu nedostupných používateľov, napríklad tých na dovolenke. Ak úloha prebieha v stredu a niekto je mimo, nepriradí sa. Je to ako kontrola kalendára, aby sa predišlo plánovacím konfliktom.

Všetky tieto komponenty spolu zabezpečujú, že model vytvorí platný a kvalitný plán vyvažujúci prísne požiadavky s preferenčným skórovaním. Používa binárne premenné na reprezentáciu priradení, maximalizačnú funkciu uprednostňujúcu vysoko-kvalitné riešenia a súbor obmedzení presadzujúcich pravidlá. Následne optimalizačný modul dostane túto cieľovú funkciu spolu so všetkými definovanými obmedzeniami a hľadá optimálne riešenie.

3.4.4 Logika

```

1  create model
2  for each issue:
3      for each role:
4          for each user:
5              if user qualified for role and issue category:
6                  if user available during issue time:
7                      define binary variable x[issue,user,role]
8                      check preference filters
9                      if filters fail:
10                         define binary variable y[issue,user,role]
11 set objective to maximize sum of ratings minus filter penalties
12 add constraint: each role gets required number of users
13 add constraint: no user assigned to overlapping issues
14 add constraint: only qualified users assigned
15 add constraint: only available users assigned
16 add constraint: at most one role per user per issue
17 solve model with solver
18 if solution found:
19     extract assignments from variables
20 else:
21     export model for diagnostics
22     if partial assignment allowed:
23         run fallback algorithm
24     return empty result

```

Vysvetlenie: Logika začína vytvorením modelu ILP. Pre každý problém a rolu kontroluje kvalifikovaných a dostupných používateľov, pričom definuje binárne premenné $x_{u,p,r}$ na označenie priradení a $y_{u,p,r}$ na sledovanie porušení filtrov. Cieľom je maximalizovať hodnotenie r_{p,r,c_u} znížené o pokuty za zlyhania filtra (napr., $\kappa = 100$). Obmedzenia zabezpečujú správne počty rolí, žiadne prekryvanie, kvalifikáciu, dostupnosť a maximálne jednu rolu na používateľa a úlohu. Po vyriešení sa úlohy extrahujú alebo sa pri zlyhaní modelu použije núdzový algoritmus (greedy).

Silné stránky:

- Globálne optimalizuje úlohy, vyrovnáva hodnotenia a preferencie.
- Systematicky spracováva všetky obmedzenia.
- Podporuje záložné riešenie pre nerealizovateľné prípady.

Slabé stránky:

- Čas riešenia závisí od veľkosti problému.
- Zložitosť modelu rastie s problémami a používateľmi.

3.4.5 Optimalita a Zložitosť

Algoritmus celočíselného lineárneho programovania (ILP), popísaný v sekciách 3.4.3 a 3.4.4, zaručuje optimálne priradenie používateľov k rolám v úlohách, maximalizujúc hodnotenia pri dodržaní obmedzení. Táto sekcia dokazuje optimalitu riešenia a analyzuje výpočtovú zložitosť algoritmu a poskytuje pohľad na jeho teoretický a praktický výkon.

Optimalita

Model ILP maximalizuje cieľovú funkciu:

$$\sum_{u \in U} \sum_{p \in P} \sum_{r \in R_u} (r_{p,r,c_u} \cdot x_{u,p,r} - \kappa \cdot y_{u,p,r}),$$

kde $x_{u,p,r} \in \{0,1\}$ označuje priradenia, r_{p,r,c_u} je hodnotenie používateľa p pre rolu r v kategórii úlohy u , $y_{u,p,r} \in \{0,1\}$ sleduje porušenia filtrov a $\kappa = 100$ penalizuje nepreferované priradenia (pozri sekciu 3.4.3). Obmedzenia: počet rolí, žiadne prekrytia, kvalifikácia, dostupnosť a maximálne jedna rola na používateľa v úlohe, definujú priestor platných priradení.

Implementácia algoritmu CBC rieši kombinujúcu relaxáciu lineárneho programovania s celočíselnými obmedzeniami. Keďže cieľ je lineárny a premenné sú binárne, ILP nachádza globálne maximum v priestore riešení. Konkrétne, pre každé platné priradenie $\{x_{u,p,r}\}$ optimalizačný modul implicitne vyhodnotí všetky kombinácie (pomocou rezov), čím zaručuje, že žiadne iné priradenie nemá vyššiu hodnotu cieľa. To zaisťuje najvyšší možný súčet hodnotení, upravený o penalizácie filtrov, pri splnení všetkých obmedzení.

Napríklad, ak dvaja používatelia súperia o jednu rolu v úlohe, solver vyberie používateľa s vyšším $r_{p,r,c_u} - \kappa \cdot y_{u,p,r}$, pokiaľ to obmedzenia (napr. prekryvanie) nezakazujú. Globálna optimalita je v kontraste s greedy metódami (pozri sekciu 5.2), ktoré môžu skončiť v lokálnom maxime.

Zložitosť

Rozhodovací problém ILP, teda určenie, či existuje platné priradenie, je NP-úplný, pretože ho možno zredukovať na problém plánovania úloh s obmedzeniami zdrojov. Nech $|U|$ je počet úloh, $|P|$ počet používateľov a $|R_u|$ počet rolí v úlohe. Model má až $|U| \cdot |P| \cdot \max_u |R_u|$ premenných $(x_{u,p,r}, y_{u,p,r})$ a približný počet obmedzení je:

- **Počet rolí:** $|U| \cdot \max_u |R_u|$,
- **Prekryvanie:** $|P| \cdot |U|$,
- **Kvalifikácia/dostupnosť:** $|U| \cdot |P| \cdot \max_u |R_u|$,
- **Jedna rola na používateľa v úlohe:** $|U| \cdot |P|$.

Najhorší prípad časovej zložitosti metódy branch-and-cut je exponenciálny, pretože môže preskúmať $2^{|U| \cdot |P| \cdot \max_u |R_u|}$ riešení. Praktický výkon je však závislý od rôznych parametrov:

- **Riedkosť:** Málo používateľov je kvalifikovaných ($r_{p,r,c_u} > 0$), čím sa znižuje počet premenných.
- **Obmedzenia:** Prekrývania a dostupnosť obmedzujú priestor hľadania.
- **Efektivita riešiteľa:** CBC používa heuristiky (napr. pedspracovanie, rezné roviny) na rýchlejšie skonvergovanie.

Pre typické inštancie Redmine (napr. $|U| = 100$, $|P| = 50$, $\max_u |R_u| = 3$) riešiteľ často skončí za sekundy, no husté problémy (veľa prekrývaní) môžu trvať dlhšie.

Literatúra

Dôkaz optimality vychádza zo štandardnej teórie ILP [3]. Pre výkon optimalizačného modulu(solver) pozri [2].

Vysvetlenie: ILP zaisťuje najlepšie možné priradenia vyhodnotením všetkých platných plánov, na rozdiel od greedy algoritmov, ktoré môžu prehliadnúť optimálne riešenia. Jeho zložitosť je teoreticky vysoká, no pre Redmine je zvládnuteľná, čo sekcia 3.5 ďalej overí benchmarkmi.

3.4.6 Testovanie

Algoritmus bol prísne testovaný, aby sa zabezpečilo optimálne priradenie úloh medzi používateľmi pri dodržaní obmedzení z 3.2. Vyhodnocované testy:

- **Základná funkčnosť:** Správne priraďovanie k rolám, uprednostňovanie vyšších hodnotení.
- **Krajné situácie:** Scenáre obsahujúce prázdne vstupy a podobne ($U = \emptyset$, $P = \emptyset$).
- **Obmedzenia:** Zabránenie dvojitej rezervácií.
- **Používateľské preferencie:** Presná aplikácia filtrov definovaných používateľom.

Pomocou frameworku *unittest* bola testovaná funkcia *match_issues_to_users* so simulovanými údajmi. Vstupy zahŕňali problémy s rolami, kvalifikáciami a časovými obmedzeniami.[5]

Funkcia diagnostiky, ktorá zaznamenáva dôvody nerealizovateľnosti, ponúka spätnú väzbu pre vývojárov a správcov Redmine na rýchle riešenie konfliktov.

3.5 Porovnanie

Cieľom tejto sekcie je porovnať výkon algoritmov celočíselného lineárneho programovania (ILP), greedy prístupu a backtrackingu na základe času behu a kvality priradenia úloh. Benchmarkové testy kvantifikujú efektivitu algoritmov pri priradovaní používateľov k rolám $(x_{u,p,r})$ s ohľadom na kvalifikácie (r_{p,r,c_u}) a obmedzenia (napr. žiadne prekrývanie).

Metodológia

Benchmarky boli vykonané pomocou viacerých skriptov, ktoré testujú funkciu `match_issues_to_users` na simulovaných dátach. Testovacie sady obsahovali:

- **Malá sada (Test 1):** $|U| = 200$ úloh, $|P| = 100$ používateľov.
- **Veľká sada (Test 2):** $|U| = 1000$ úloh, $|P| = 250$ používateľov.
- **Test s veľkým počtom závislostí (Test 3):** $|U| = 24$ úloh, $|P| = 7$ používateľov.
- **Test pre záložný scenár (Test 4):** $|U| = 25$ úloh, $|P| = 8$ používateľov.

Význam testov:

- Test 1: Overenie rýchlosti a kvality na menšej sade pre základnú funkčnosť.
- Test 2: Test škálovateľnosti algoritmov na veľkej sade dát.
- Test 3: Skúmanie zvládania komplexných závislostí s obmedzeným počtom používateľov.
- Test 4: Simulácia scenára s nedostatkom používateľov na demonštráciu potreby záložného mechanizmu.

Algoritmy boli spustené na Python 3.13, pričom ILP využívalo PuLP a CBC modul.

Výsledky

Tabuľka 3.1 sumarizuje výkon algoritmov pre obe dátové sady.

Tabuľka 3.1: Porovnanie výkonu algoritmov

Algoritmus	Test	Čas behu	Plne vyriešené (%)
ILP	Test 1	0.43 s	100%
Greedy	Test 1	0.04 s	100%
Backtracking	Test 1	1.15 s	100%
ILP	Test 2	25.62 s	100%
Greedy	Test 2	3.03 s	100%
Backtracking	Test 2	77.69 s	100%
ILP	Test 3	65.26 ms	100%
Greedy	Test 3	0.32 ms	25%
Backtracking	Test 3	1.53 ms	100%
ILP	Test 4	201.19 ms	0%
Greedy	Test 4	0.27 ms	52%
Backtracking	Test 4	0.62 ms	0%

Analýza

- **ILP:** Rýchlejší ako backtracking v testoch 1 a 2 (0.43s oproti 1.15s pre 200 úloh, 25.62s oproti 77.69s pre 1000 úloh), ale pomalší v teste 3 (65.26 ms oproti 0.32 ms greedy). V teste 4 zlyhal (0% plne vyriešených úloh), pretože scenár bol nerealizovateľný kvôli nedostatku používateľov. Zaručuje optimálne priradenie $(\sum_r p, r, c_u \cdot x_{u,p,r} - \kappa \cdot y_{u,p,r})$ napriek vyššej zložitosti.
- **Greedy:** Najrýchlejší vo všetkých testoch (0.04s pre 200 úloh, 3.03s pre 1000 úloh, 0.32 ms pre 24 úloh, 0.27 ms pre 25 úloh). V teste 3 dosahuje iba 25% plne vyriešených úloh kvôli suboptimálnym rozhodnutiam, ale v teste 4 poskytol čiastočné riešenie (52%) napriek nerealizovateľnosti.
- **Backtracking:** Najpomalší pri väčších dátach (1.15s pre 200 úloh, 77.69s pre 1000 úloh, 1.53 ms pre 24 úloh), ale spoľahlivý pri realizovateľných vstupoch a zvláda komplexné závislosti v teste 3, ale v teste 4 zlyhal (0.62 ms, 0%) kvôli nerealizovateľnosti scenára.

Záver porovnania

Na základe vykonaných benchmarkov sme za primárnu metódu priradenia vybrali ILP. Síce na väčších dátach vyžaduje viac času (25,62 s pri 1000 úlohách), no zaručuje vždy optimálne a úplné riešenie úloh v rámci všetkých realizovateľných testovacích scén a spoľahlivo zvláda aj zložité závislosti medzi úlohami (Test 3). Výber ILP považujeme za najvhodnejší kompromis medzi kvalitou riešenia a výpočtovým časom, najmä pre prípady, kde je kľúčové dosiahnuť optimálnu alokáciu zdrojov.

Ako záložný mechanizmus, ktorý zabezpečí aspoň čiastočné priradenie priradenie v nerealizovateľných scenároch, spúšťame greedy prístup. Greedy síce nedosahuje optimálnu hodnotu pri silne závislých dátach (len 25 % vyriešených úloh v Teste 3), no za zlomok času poskytne rýchle a dobre využiteľné čiastočné riešenie, ktoré môže poslúžiť ako východiskový bod pre ďalšie doladovanie.

Kapitola 4

Architektúra

Podľa vykonanej analýzy a našich načrtnutých požiadaviek v návrhu softvéru je najefektívnejším prístupom prijať existujúce softvérové riešenie a vylepšiť ho vývojom pluginu. Využitie zavedeného softvéru ako základu výrazne skracuje počiatočný čas vývoja, znižuje potenciálne riziká spojené s vytvorením úplne nového systému a ťaží z existujúcej stability a podpory komunity. Tento prístup nám umožňuje zamerať sa priamo na riešenie našich špecifických potrieb v oblasti automatizácie a prispôsobenia, a nemíňať značné zdroje na budovanie základnej infraštruktúry.

Napriek tomu, že žiadna z analyzovaných platforiem dokonale nespĺňa všetky požiadavky, Redmine sa ukázal ako najvhodnejší kandidát, najmä vďaka svojej open source povahe, rozsiahlej prispôsobiteľnosti a robustnému ekosystému pluginov. Výberom Redmine ako našej základnej platformy môžeme efektívne vyvinúť prispôsobené riešenie, ktoré priamo rieši jedinečné výzvy a zložitosti, s ktorými sa potrebujeme vyrovnávať v našom softvéri.

4.1 Redmine

Redmine je bezplatná webová open-source aplikácia špeciálne vytvorená na riadenie projektov a sledovanie problémov, postavená na robustnom frameworku Ruby on Rails. Softvér pôvodne vyvinul Jean-Philippe Lang a prvá oficiálna verzia bola verejne vydaná 25. júna 2006. Od svojho počiatku sa Redmine zameral na ponúkание vysoko prispôsobiteľného a flexibilného prostredia na riadenie projektov, vhodného pre rôzne organizačné potreby, od tímov vývoja softvéru až po všeobecnú administratívnu správu.

Jednou z definujúcich vlastností Redmine je jeho multiplatformová a medzi databázová podpora. Bezproblémovo funguje na rôznych operačných systémoch vrátane Linuxu, Windowsu a macOS a podporuje viacero databázových backendov, ako sú MySQL, PostgreSQL, SQLite a SQL Server. To zaisťuje širokú kompatibilitu a jednoduchosť nasadenia v rôznych prostrediach infraštruktúry, vďaka čomu je Redmine

dostupný tak malým organizáciám, tak aj veľkým podnikom.

Internacionalizácia a jazyková podpora predstavuje ďalšiu významnú silu Redmine. Aplikácia v súčasnosti podporuje 49 jazykov, vďaka čomu je široko prístupná a prispôbiteľná pre globálne skupiny používateľov a nadnárodné tímy. Tento dôraz na lokalizáciu výrazne prispel k jeho popularite a uľahčil jeho široké prijatie v mnohých krajinách a odvetviach.

V priebehu času si platforma vybudovala silnú a aktívnu komunitu, ktorá neustále prispieva k jej rozvoju. Open-source, teda softvér s prístupným zdrojovým kódom, si vyžaduje neustále zlepšovanie, opravy chýb a vývoj inovatívnych pluginov členmi komunity na celom svete. Tento komunitou riadený prístup zaisťuje neustále zdokonaľovanie, spoľahlivú podporu a bohatý ekosystém doplnkov a integrácií, ktoré výrazne rozširujú jeho základné funkcie.[10]

Okrem toho existencia rôznych vetiev a odvodených projektov, ako sú Easy Redmine a OpenProject, naznačuje prispôbivosť a popularitu softvéru. Tieto deriváty ponúkajú špecifické vylepšenia, najmä teda zlepšené používateľské rozhrania, mobilnú podporu a sofistikované analýzy, ktoré rozširujú potreby nad rámec pôvodných možností Redmine.

Celkovo Redmine vyniká ako vysoko flexibilné, rozšíriteľné a adaptabilné riešenie schopné vyhovieť rôznym požiadavkám na riadenie projektov. Vďaka výkonnej kombinácii natívnych funkcií, rozsiahlej architektúre doplnkov a aktívnej komunitnej podpore je veľmi priaznivou voľbou pre organizácie, ktoré hľadajú efektívnu a prispôbiteľnú platformu na riadenie projektov.

4.1.1 Identifikované medzery vo funkciách Redmine

Zatiaľ čo Redmine poskytuje robustnú sadu základných funkcií projektového manažmentu a značnú flexibilitu, niekoľko kritických funkcií vyžadovaných špeciálne pre náš scenár manažmentu udalostí chýba alebo je nedostatočne implementovaných:

- **Správa dní voľna:** Redmine nemá podporu na sledovanie a správu obmedzení dostupnosti. Táto funkcia je nevyhnutná pre chod nášho systému.
- **Mechanizmus vlastných požiadaviek:** Hoci isté filtrovanie existuje, pokročilé funkcie filtrovania založené na dynamických atribútoch účastníkov nie je podporované.
- **Hodnotenie používateľských kvalifikácií:** Redmine neposkytuje vstavané funkcie na hodnotenie používateľských kompetencií, kvalifikácií a rolí, ktoré sú rozhodujúce pre optimalizované priradenie úloh.
- **Algoritmus automatického priradenia:** Neexistuje žiadna vstavaná funkcia pre automatizáciu priradovania úloh účastníkom na základe ich parametrov.

4.1.2 Integrácia vlastných funkcií pomocou Redmine Hooks a Patches

Na efektívnu implementáciu chýbajúcich funkcií v našom systéme založenom na Redmine sme sa rozhodli využiť vstavané mechanizmy rozšírenia, teda najmä Hooks(háčky) a Patches(záplaty). Tento prístup zaisťuje, že vylepšenia sa hladko integrujú s existujúcou infraštruktúrou bez úpravy jej základných súborov, čím sa zachová stabilita systému, zjednodušia sa aktualizácie a zníži sa budúca záťaž na údržbu.

Hooks

Hooks sú preddefinované body v rámci interných procesov Redmine, kde je možné spustiť vlastný kód z pluginu. Využitie Hooks nám umožní bezproblémovo vložiť ďalšie prvky používateľského rozhrania, overenia alebo backendovú logiku práve tam, kde je to potrebné. Napríklad výber požadovaných rolí ku konkrétnej úlohe možno vložiť priamo do existujúceho formulára vytvárania úloh. Vďaka tomu sú úpravy izolované v rámci nášho pluginu, čím sú jasne oddelené od základnej logiky základného systému.

Patches

Patches na druhej strane poskytujú možnosť dynamicky prepísať alebo rozšíriť existujúce funkcie Redmine. Táto technika je obzvlášť užitočná na rozšírenie možností existujúcich modelov, ovládačov alebo zobrazení nad rámec toho, čo môžu dosiahnuť Hooks. Napríklad používame Patches na zlepšenie mechanizmu rozdelenia problémov tak, aby zahrňal logiku založenú na kvalifikáciách a priradujeme nové atribúty triedam. Pozorným použitím pripojenia modulu, reťazenia metód aliasov alebo metód vložených na začiatok môžeme bezpečne a reverzibilne upraviť správanie Redmine a bez toho, aby sme riskovali dlhodobé problémy s kompatibilitou.

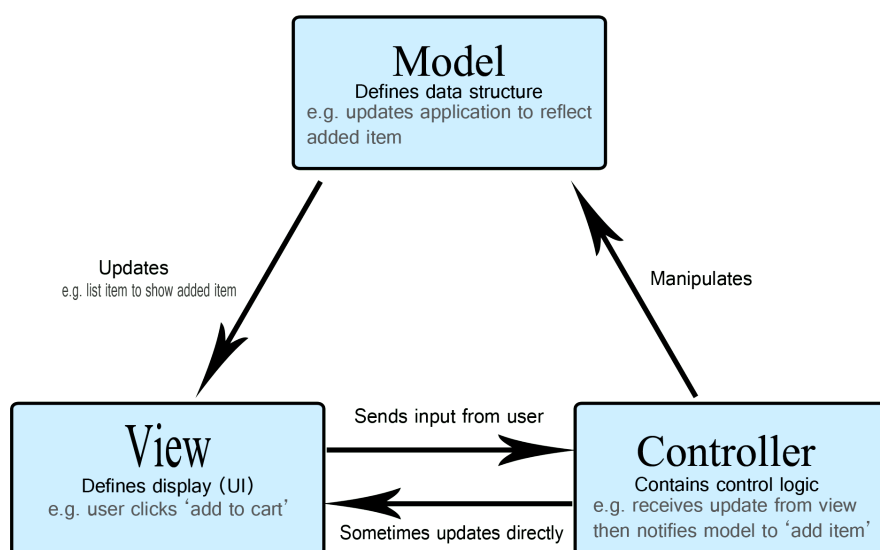
Kombinácia týchto dvoch metód zaisťuje robustnú a zároveň minimálne invazívnu integráciu. Naša architektúra sa preto strategicky spolieha na Patches na zlepšenie backendovej logiky a Hooks pre inováciu používateľských rozhraní. Tento dizajn je nielen v súlade s osvedčenými postupmi odporúčanými vývojármi Redmine, ale tiež vidno benefity v udržateľnosti a škálovateľnosti.

Tento vrstvený a modulárny prístup rozšírenia nám umožňuje využívať budúce aktualizácie Redmine s minimálnym narušením a jasne izoluje našu vlastnú logiku pre jednoduchšie ladenie, testovanie a vývoj.

4.2 Ruby on Rails

Redmine využíva vývojový rámec (framework) Ruby on Rails, preto bude plánovaný plugin vyvíjaný predovšetkým v tomto prostredí. Ruby on Rails (RoR) je rozšírený open-source framework pre vývoj webových aplikácií v jazyku Ruby, ktorý kladie dôraz na vysokú produktivitu a dlhodobú udržiavateľnosť robustných riešení. Prvýkrát ho predstavil David Heinemeier Hansson v roku 2004 a odvtedy si získal široké uplatnenie pre svoj dôraz na jednoduchosť, produktivitu a udržiavateľnosť. Rails si zakladá na filozofii konvencie pred konfiguráciou, čo znamená, že sa spolieha na štandardizované štruktúry a vzory, aby sa znížila potreba explicitnej konfigurácie, čo výrazne urýchľuje vývojové procesy.

Framework je štruktúrovaný okolo architektúry Model-View-Controller (MVC), ktorý jasne rozdeľuje aplikáciu na tri vzájomne prepojené komponenty:



Obr. 4.1: Diagram znázorňuje tok riadenia a údajov medzi tromi primárnymi komponentmi vzoru MVC. Ovládač spracuje vstup používateľa, manipuluje s modelom a určuje, ktoré zobrazenie sa má aktualizovať.

- **Model:** Model spravuje údaje a logiku aplikácie, pričom zahŕňa pravidlá a správanie súvisiace s údajmi.
- **View:** Zobrazenie zodpovedá za vykreslenie používateľského rozhrania, zobrazenie informácií používateľovi a prezentáciu údajov poskytovaných modelom.

- **Controller:** Ovládač pôsobí ako sprostredkovateľ, spracováva užívateľské vstupy zo zobrazenia, interaguje s modelom na aktualizáciu alebo získavanie údajov a určuje vhodnú odpoveď pre zobrazenie.

Prístup MVC v RoR ponúka významné výhody, ako lepšia udržiavateľnosť, jasnejšie oddelenie problémov a možnosť znovupoužitelnosti. Každý komponent má špecifickú zodpovednosť, čo zjednodušuje orientáciu v systéme.

4.3 Dátová vrstva a databáza

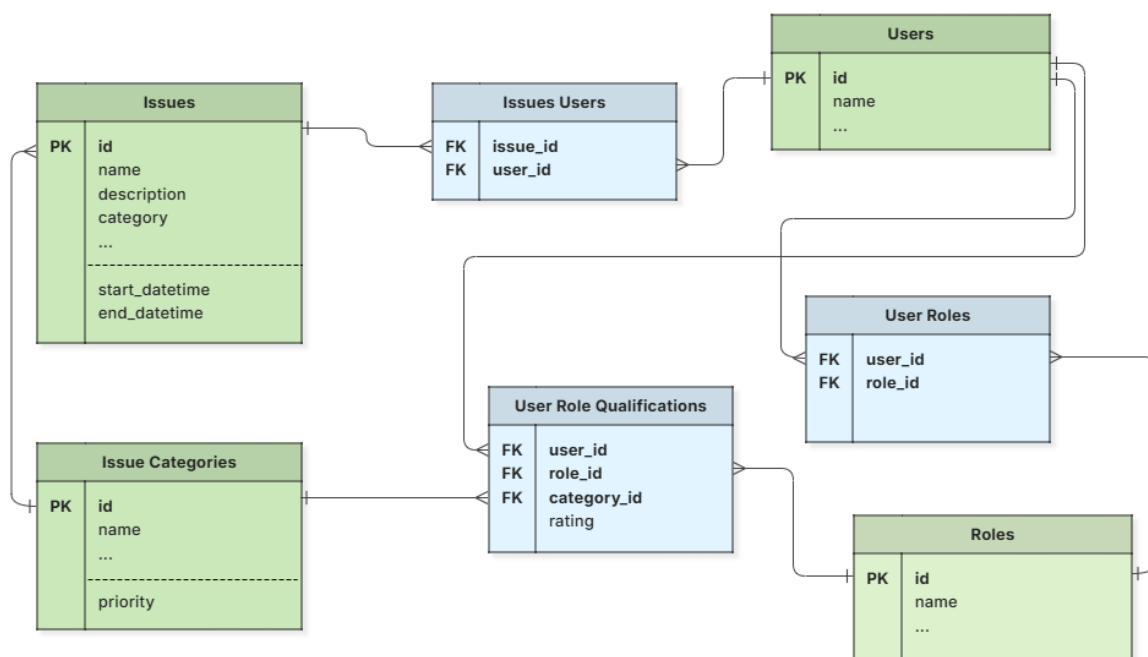
Na efektívne ukladanie a správu aplikačných údajov bol PostgreSQL vybraný ako naše databázové riešenie kvôli niekoľkým funkciám, ktoré ponúka. Napríklad pokročilá optimalizácia výkonu, pružnosť a špecifická podpora pre stĺpce JSON.

Schopnosť spracovávať stĺpce typu JSON rieši našu požiadavku na dynamické a štruktúrované údaje, najmä pre prispôsobiteľné kritériá užívateľov. Využitím takýchto dátových typov dosahujeme flexibilitu v dátových štruktúrach bez obetovania výkonu a integrity dotazov.

Naša databázová architektúra je rozšírením existujúcej schémy Redmine, pokým zachováva všetky pôvodné tabuľky, tak ako boli vytvorené. Zavedením niekoľkých nových, potrebných tabuliek sa vyplní medzera, tak aby spĺňali identifikované funkčné medzery:

- **Dni voľna :** Ukladá časové dáta, teda obmedzenia dostupnosti používateľov.
- **Používateľské kvalifikácie :** Zachytáva podrobné zručnosti a kvalifikácie pre každého používateľa.
- **Vlastné preferencie :** Zaznamenáva voliteľné používateľské preferencie, ukladané do stĺpcov typu JSON.

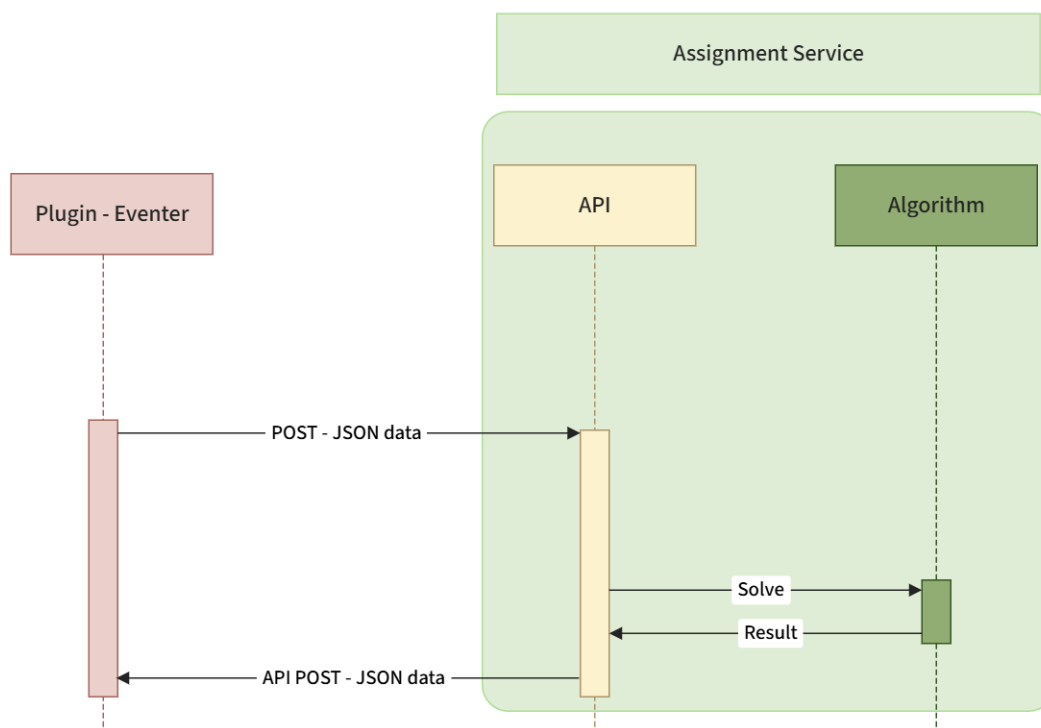
Okrem toho bolo niekoľko existujúcich tabuliek mierne rozšírených pridaním nových stĺpcov, aby vyhovovali ďalšej požadovanej funkcionalite, bez toho, aby bola narušená pôvodná štruktúra.



Obr. 4.2: Skrátená a zjednodušená schéma databázy jasne ilustrujúca rozdiel medzi pôvodnými a novo pridanými komponentami, pričom sú zvýraznené nové tabuľky modrou farbou a v rámci existujúcich zelených tabuliek sú pridané potrebné stĺpce oddelené prerušovanou čiarou.

4.4 Služba na pridelovanie úloh

Na efektívne zvládnutie výpočtových úloh bol priradovací algoritmus implementovaný ako samostatné API založené na Pythone. Toto oddelenie poskytuje jasné architektonické výhody, najmä zlepšenú modularitu a testovateľnosť. Python bol vybraný na základe existujúcich skúseností a jeho širokej podpory knižníc na vývoj a testovanie algoritmov.



Obr. 4.3: Diagram znázorňuje komunikáciu medzi pluginom a službou na pridelovanie úloh

Ako je znázornené na obr 4.3, komunikácia medzi pluginom - *Eventer* a službou na priradenie úloh prebieha podľa štruktúrovanej sekvencie riadenej udalosťami:

1. **Odoslanie údajov** : Akcia používateľa alebo systémová udalosť v rámci pluginu odošle POST informáciu aj s dátami uloženými vo formáte JSON.
2. **Spracovanie** : Po zhromaždení všetkých potrebných údajov ich API odovzdá vstavanému algoritmu, ktorý na základe poskytnutých parametrov vypočíta optimálne priradenia.
3. **Odovzdanie výsledku** : Po nájdení riešenia API odošle výsledky požiadavkou POST späť do pluginu a poskytne výsledky priradenia vo formáte JSON. Tieto výsledky potom plugin uloží a premietne do systému.

Táto oddelená architektúra zaisťuje čisté oddelenie zodpovedností: Plugin sa stará o ukladanie údajov a interakciu s používateľom, zatiaľ čo služba na pridelovanie úloh vykonáva algoritmickú logiku. Umožňuje tiež testovať algoritmus nezávisle od prostredia, čím zefektívňuje vývoj a zabezpečuje dlhodobú udržiateľnosť.

4.5 Prostredie nasadenia

Naša aplikácia používa na nasadenie Docker, ktorý poskytuje reprodukovateľné, izolované a ľahko spravovateľné vývojové prostredie. Nastavenie založené na Dockeri umožňuje efektívnu spoluprácu, rýchle testovacie cykly a konzistentné správanie v rôznych vývojových prostrediach a systémoch. Medzi primárne služby definované v rámci nastavenia Docker Compose patria:

- **Redmine Container** : Tento kontajner spustený na oficiálnom obrázku (image) spracováva všetku logiku riadenia projektu, najmä používateľské rozhranie a interakcie. Sprístupňuje port 3000 hostiteľovi, čo umožňuje ľahký prístup počas vývoja. Všetky prispôsobenia a plugíny sú oddelené od základných súborov a uložené vo vyhradenom adresári pluginov, čo zaisťuje jednoduchú aktualizáciu a údržbu.
- **PostgreSQL Database Container** : Vyhradený kontajner so systémom PostgreSQL, ktorý používa robustné ukladanie údajov. Trvalé ukladanie údajov sa dosiahne prepojením lokálneho adresára (`./db`), čím sa zabezpečí, že databázové údaje zostanú konzistentné počas reštartov kontajnera, čo uľahčuje testovanie a nepretržitý vývoj.
- **Assignment Service Container** : Tento kontajner zapuzdruje API a logiku zodpovedajúceho algoritmu. Komunikácia medzi Redmine a servisom prebieha interne v rámci siete Docker prostredníctvom názvu kontajnera (`http://assignment-service:5000`). Kontajner automaticky nainštaluje potrebné závislosti a knižnice, prostredníctvom súboru požiadaviek a spustí službu API.

Táto modulárna kontajnerová architektúra výrazne zjednodušuje nasadenie a podporuje jasné oddelenie záujmov. Využitím Dockera sa navyše zaisťuje, že potenciálne budúce produkčné nasadenia môžu využívať rovnakú infraštruktúru s malými úpravami.

Kapitola 5

Implementácia

Táto kapitola poskytuje podrobný prehľad procesu implementácie pluginu pre Redmine vyvinutého v tomto projekte. Zatiaľ čo predchádzajúca kapitola sa zameriavala na architektonický dizajn a odôvodnenie kľúčových komponentov, táto časť opisuje praktické aspekty budovania systému. Nakoľko je algoritmus priraďovania samostatným a komplexným komponentom, je popísaný vo vlastnej kapitole 3. Tu popisujeme integráciu pluginu do Redmine, rozšírenie základných modelov a zobrazení, manipuláciu s údajmi a podobne. V prípade potreby sú k zahrnuté aj vzorky kódu či snímky obrazovky pre lepšiu ilustráciu implementácie.

5.1 Štruktúra a nastavenie pluginu

5.1.1 Inicializácia pluginu a integrácia menu

Na efektívnu integráciu vlastných funkcií do Redmine bez úpravy základných súborov bol zvolený prístup založený na pluginoch. To zaisťuje kompatibilitu s budúcimi aktualizáciami Redmine a zjednodušuje údržbu, čo umožňuje izolovaný, modulárny vývoj.

Plugin sa primárne inicializuje prostredníctvom súboru *init.rb*, ktorý sa nachádza v koreňovom adresári. Tento súbor definuje kľúčové informácie o plugine, ako je názov, verzia, autor a popis:

```
1 # init.rb
2 Redmine::Plugin.register :eventer do
3   name 'Eventer plugin'
4   author 'Tomáš Magula'
5   description 'This is a plugin for Redmine'
6   version '4.2.0'
7   url 'https://github.com/magula12/eventer'
8   author_url 'http://example.com/about'
```

Tento inicializačný súbor tiež špecifikuje položky ponuky, ktoré sa nachádzajú v

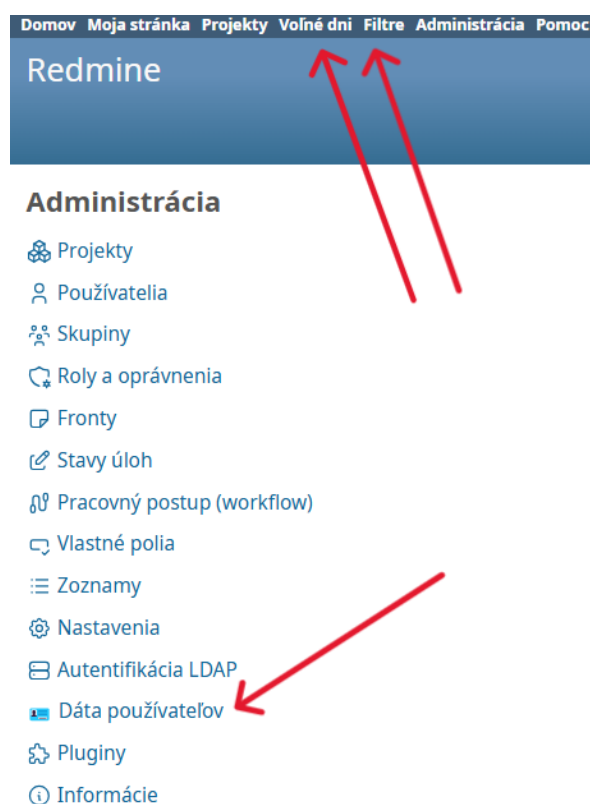
rozhraní. Metóda ponuky integruje vlastné stránky do existujúcich ponúk, ako je navigácia na najvyššej úrovni alebo administrátorská ponuka. Pre aktuálny plugin boli pridané tri nové položky ponuky:

```

9  menu :top_menu, :offdays,
10    { controller: 'offdays', action: 'index' },
11    caption: :label_offdays
12
13  menu :top_menu, :filters,
14    { controller: 'custom_filters', action: 'index' },
15    caption: :label_custom_filters
16
17  menu :admin_menu, :users_data,
18    { controller: 'users_data', action: 'index' },
19    caption: :label_users_data
20  end

```

Tieto položky umožňujú jednoduchú navigáciu priamo k hlavným funkciám pluginu.



Obr. 5.1: Upravené navigačné menu v Redmine

Ak by sme chceli obmedziť prístup k ponuke na základe používateľských rolí a povolení, je možné definovať ďalšie parametre, ako príklad *if* alebo *permission*. Tým sa zabezpečí, že ponuky sa zobrazia len určitým používateľom, ale nakoľko ponuka *Dáta používateľov* sa nachádza už priamo v administrátorskom menu, nebolo potrebné použiť tieto parametre.

5.1.2 Konfigurácia ciest (Routes)

Tento plugin predstavuje niekoľko vlastných ciest, ktoré vyhovujú funkciám orientovaným na používateľa (CRUD operácie) a špecializovaným integráciám backendu (API). Všetky cesty špecifické pre plugin sa nachádzajú v súbore *config/routes.rb* v adresári pluginu. Kombináciou RESTful a non-RESTful riešení vieme dosiahnuť požadovanú funkcionálnosť.

RESTful smerovanie s resources

Pre primárne operácie, ktoré dodržiavajú štandardné vzory vytvorenia - čítania - aktualizácie - vymazania (CRUD), plugin využíva konvenčnú metódu zdrojov Rails. To generuje súbor trás spojených so špecifikovanými ovládačmi a akciami, čo zjednodušuje vývoj a údržbu používateľského rozhrania.

```
1 # config/routes.rb
2 resources :users_data, only: [:index, :show]
3
4 resources :qualifications, only: [:new, :create, :edit,
5   :update, :destroy]
6
7 resources :offdays, only: [:index, :create, :destroy]
```

Flexibilné koncové body s match

Nie všetky funkcie pluginov presne spadajú do paradigmy RESTful. Napríklad spúšťače externých skriptov alebo prenosy údajov založené na JSON môžu vyžadovať vlastné trasy, formáty alebo metódy. V týchto prípadoch plugin využíva metódu zhody na definovanie ciest s väčšou kontrolou nad metódami HTTP, názvami ciest a predvolenými parametrami:

```
8 match 'eventer_api',
9   to: 'eventer_api#index',
10  via: :get,
11  defaults: { format: 'json' }
12
13 match 'algorithm_runner',
14   to: 'algorithm_runner#index',
15   via: :get,
16   as: :algorithm_runner
```

Súhrn

Kombinácia stratégií RESTful (resources) a vlastných (match) smerovania umožňuje doplnku zvládnuť typickú manipuláciu s údajmi (dni voľna, kvalifikácie) spolu s po-

kročilými scenármi (spúšťače algoritmov, prenosy údajov JSON). [8]

5.2 Rozšírenia vrstvy modelu a databázy

Plugin predstavuje nové vlastné modely a rozšírenia databázových schém na splnenie požiadaviek pridaných funkcionalít.

5.2.1 Vlastné migrácie databáz

Na rozšírenie funkcií doplnku a pokrytie nových požadovaných tabuliek či stĺpcov v existujúcich tabuľkách podľa obrázku 4.2, je potrebné využitie migrácií definovaných konvenciami Rails.

Napríklad pridáme novú tabuľku na ukladanie kvalifikácií používateľov súvisiacich s konkrétnymi rolami a kategóriami. Cudzie kľúče sú explicitne definované na zachovanie referenčnej integrity:

```
1 class CreateUserRoleQualifications < ActiveRecord::Migration[7.2]
2   def change
3     create_table :user_role_qualifications do |t|
4       t.integer :user_id, null: false
5       t.integer :role_id, null: false
6       t.integer :category_id, null: false
7       t.integer :rating, default: 0, null: false
8
9       t.timestamps
10    end
11    add_foreign_key :user_role_qualifications, :users,
12      column: :user_id
13    # ...other keys and unique constraint
14  end
15 end
```

Alebo pridanie polí dátumu a času(*start_datetime* a *end_datetime*) priamo do existujúcej tabuľky *issues* v rámci Redmine:

```
1 class AddDatetimeFieldsToIssues < ActiveRecord::Migration[7.2]
2   def change
3     add_column :issues, :start_datetime, :datetime
4     add_column :issues, :end_datetime, :datetime
5   end
6 end
```

5.2.2 Model vlastných preferencií

Na umožnenie flexibilnej a dynamickej logiky filtrovania bol predstavený model *CustomFilter*. Tento model ukladá používateľom definované kritériá pomocou dátových štruktúr JSON.

Každý *CustomFilter* je priradený priamo k používateľovi, čo zabezpečuje nezávislú správu údajov. Formát úložiska JSON pozostáva z dvoch kľúčových komponentov: pravidiel a podmienok. Oba sú objekty JSON špecifikujúce logické vzťahy (AND a OR) a rôzne kritériá porovnávania (`==`, `<`, `>`, `in`, ...).

Aby sa zabezpečila integrita údajov a zabránilo sa chybám, model obsahuje robustnú logiku overovania, ktorá kontroluje, či je poskytnutá štruktúra JSON platná. Príklad, ktorý demonštruje, ako štruktúra JSON vyzerá:

```
1 {  
2   "conditions": {  
3     "and": [  
4       {"==": [{"var": "issue.priority"}, "High"]},  
5       {">=": [{"var": "issue.estimated_hours"}, 8]}  
6     ]  
7   },  
8   "rules": {  
9     "or": [  
10      {"in": [{"var": "user.skills"}, "Ruby"]}]  
11   ]  
12 }  
13 }
```

5.3 Úprava komponentov Redmine

Doplnok rozširuje základné triedy Redmine pomocou *Patch* za behu, čo je bezpečná a efektívna metóda na pridávanie nových funkcií bez priamej zmeny zdrojového kódu. Tento prístup je rozhodujúci pre udržanie kompatibility s budúcimi aktualizáciami Redmine a pre izoláciu logiky špecifickej pre plugin.

Použitie *Patch* v Ruby, najmä pre doplnky Redmine, zvyčajne zahŕňa vloženie dodatočnej logiky alebo metód do existujúcich tried. Plugin primárne využíva dve techniky:

1. **include a ActiveSupport::Concern** - na pridávanie nových priradení, overení a vlastných metód.
2. **alias_method s class_eval** - na selektívne prepísanie existujúcich metód ale zachovanie pôvodného správania.

Tieto záplaty sa aplikujú podmiennečne a načítajú sa pomocou metódy

Rails.configuration.to_prepare odporúčanej spoločnosťou Redmine, ktorá zaisťuje, že rozšírenie sa aplikuje práve raz.

```
1 Rails.configuration.to_prepare do
2   unless Issue.included_modules.include?(Eventer::IssuePatch)
3     Issue.send(:include, Eventer::IssuePatch)
4   end
5 end
```

Použitie include s ActiveSupport::Concern

Tento prístup zavádza nové metódy, asociácie a overenia existujúcich modelov alebo ovládačov Redmine.

```
1 module Eventer
2   module IssuePatch
3     extend ActiveSupport::Concern
4
5     included do
6       has_and_belongs_to_many :assigned_users, class_name: 'User'
7       validates :start_datetime, presence: true
8
9       def assigned_to
10         assigned_users.first
11       end
12     end
13   end
14 end
```

Použitie alias_method s class_eval

Tieto techniky sa používajú na selektívne prepísanie špecifických metód v rámci ovládačov a pomocných súborov. Tento prístup podobne zachováva pôvodné správanie, kde umožňuje volanie pôvodnej funkcie a zároveň zavádza prispôbenú logiku.

```
1 module Eventer
2   module IssueHelperPatch
3     def self.included(base)
4       base.class_eval do
5         alias_method :original_issue_assigned_to_details,
6           :issue_assigned_to_details
7
8         def issue_assigned_to_details(issue)
9           if issue.assigned_users.any?
```

```

10         issue.assigned_users.map { |u| link_to u.name,
           user_path(u) }.join(', ').html_safe
11     else
12         original_issue_assigned_to_details(issue)
13     end
14 end
15 end
16 end
17 end
18 end

```

V tomto príklade je pôvodná pomocná metóda *issue_assigned_to_details* zachovaná prostredníctvom *alias_method*. Vlastná logika pluginu kontroluje viacerých priradených používateľov, čím zlepšuje používateľské rozhranie zobrazením všetkých priradených používateľov namiesto iba jedného.

5.4 Integrácia používateľského rozhrania cez Hooks

Ako bolo načrtnuté v kapitole Architektúra 4.1.2, Redmine poskytuje Hooks, teda preddefinované integračné body, ktoré pluginom umožňujú bezproblémovo zavádzať vlastné prvky používateľského rozhrania a funkcie bez priamej úpravy základného kódu. Náš plugin využíva Hooks na vylepšenie štandardných formulárov, pridanie zobrazenia rozširujúcich detailov, či pomocou CSS a JS skrýva nepotrebné polia a informácie.

```

1 module Eventer
2   module Hooks
3     class ViewRoleAssignmentsHook < Redmine::Hook::ViewListener
4       render_on :view_issues_form_details_bottom,
5                 partial: 'eventer/role_assignments'
6     end
7   end
8 end

```

Tento Hook vloží časť definovania potrebných rolí zo súboru *views/eventer/role_assignments.html.erb* na definované miesto *:view_issues_form_details_bottom*.

Hooks poskytujú čistú a efektívnu metódu na integráciu vlastných funkcií do Redmine bez ohrozenia základného systému. Redmine definuje mnoho Hooks pre rôzne body integrácie, všetky sú komplexne zdokumentované na oficiálnej wiki Redmine [7].

5.5 Integrácia a implementácia algoritmu priradovania

Plugin Redmine odosiela zadanie problému prostredníctvom POST požiadavky na REST API endpoint `/algorithm_runner/run` vo formáte JSON. Tento JSON obsahuje údaje o úlohách (**issues**) a používateľoch (**users**), vrátane detailov ako časový rozsah úloh, požadované role, kvalifikácie používateľov a ich dostupnosť.

Naša Python knižnica obsahuje implementácie troch algoritmických modelov na riešenie priradovania: backtracking, greedy a celočíselné lineárne programovanie (ILP). Backtracking a greedy algoritmy sú implementované vlastnou logikou bez externých knižníc, zatiaľ čo ILP využíva knižnicu PuLP na efektívne spracovanie komplexných obmedzení, ako sú časové prekrytia, požadované role a preferencie používateľov. Služba vyberá najvhodnejšie riešenie na základe vstupných dát a výsledné priradenia odosiela späť do pluginu Redmine.

Kapitola 6

Vývoj a testovanie

6.1 Prehľad funkcionalít pluginu Eventer

V rámci našej práce sme vytvorili plugin *Eventer* pre systém Redmine, ktorý rozširuje funkcionalitu platformy. Plugin obohacuje profil používateľa o nové záložky, ako sú „Dostupnosť“ a „Preferencie“, kde používatelia zadávajú svoje dni voľna a individuálne požiadavky. Rovnako rozširuje detail udalosti o sekcie „Požadované role“ a „Dátum a čas“, ktoré umožňujú manažérom definovať špecifické požiadavky na úlohy.

Po vyplnení týchto údajov administrátor stlačí tlačidlo, čím sa spustí optimalizačný algoritmus. Tento algoritmus analyzuje zadané údaje a nájde čo najlepšie možné priradenie používateľov k úlohám tak, aby boli dodržané všetky podmienky. Výsledok je zobrazený v logu, ako ukazuje nasledujúci obrázok 6.1, kde sú zhrnuté priradenia.

```
=== FINAL RESULTS ===
Issue ID 76:
  - Role 'Režisér': [5]
  - Role 'Komentátor': [7]
Issue ID 77:
  - Role 'Režisér': [6]
  - Role 'Komentátor': [9]
Issue ID 78:
  - Role 'Režisér': [11]
  - Role 'Komentátor': [10]
Issue ID 79:
  - Role 'Režisér': [5]
  - Role 'Komentátor': [7]

📁 Sending assignments to Redmine...
✅ Assignments posted successfully! Status: 200
Server response: {'status': 'Assignments updated'}
```

Obr. 6.1: Ukážka znázorňuje výsledky priraďovacieho algoritmu v logu

Používatelia môžu vidieť svoje priradenia po prihlásení do svojho profilu, ako je znázornené na obrázku 6.2, zatiaľ čo detaily úloh s priradenými osobami sú dostupné v detaile udalosti, ako je vidno na obrázku 6.3.

Eventer Hľadať: Prejsť na projekt... ▼

Projekty Aktivita **Úlohy** Kalendár

Úlohy [Nová úloha](#)

▼ Filtre

☒ Stav Pridať filter

→ Nastavenia

✓ Použiť ↺ Späť na pôvodné ☐ Save vlastný filter

<input type="checkbox"/>	▼ #	Projekt	Front	Stav	Priorita	Predmet	Priradené	Aktualizované
<input type="checkbox"/>	79	Our TV	Web	Pripravená	Normálna	Match 4	Adam Doe	2025-05-31 13:43 ...
<input type="checkbox"/>	76	Our TV	Web	Pripravená	Normálna	Match 1	Adam Doe	2025-05-31 13:41 ...

Obr. 6.2: Ukážka zoznamu priradených úloh prihláseného užívateľa

Match 4 « Predchádzajúce | 1 z 2 | Ďalšie »

Pridané používateľom Redmine Admin pred 20 minút.

Priorita: Normálna
Kategória: Hokej Extraliga

Požadované Role

Režisér Požadovaný Počet: 1
Priradení Používatelia:
• Adam Doe

Komentátor Požadovaný Počet: 1
Priradení Používatelia:
• Sam Babbs

Priradení Používatelia: Adam Doe, Sam Babbs

Event Duration
Start: 2025-06-02 17:43
End: 2025-06-02 19:43

Obr. 6.3: Ukážka detailu úlohy s priradenými používateľmi

6.2 Používateľské testovanie

Na vyhodnotenie pluginu bolo uskutočnené používateľské testovanie s cieľom získať spätnú väzbu od skutočných používateľov na pracovisku vo vysielacej spoločnosti. Účastníci nastavovali vlastné preferencie a dostupnosť, kde hodnotili použiteľnosť rozhrania. Spätná väzba bola zozbieraná cez neformálne rozhovory.

Výsledky preukázali vysokú spokojnosť s novými funkciami, najmä s vlastnými filtermi, ktoré umožňujú prispôbiť priradenia podľa individuálnych preferencií, ako sú obľúbené pracovné časy alebo kolegovia, a zohľadňujú dni voľna. Manažéri ocenili flexibilitu pri plánovaní, ktorá nahrádza manuálne tabuľky a znižuje chybovosť, čím šetrí čas pri koordinácii. Naopak, používatelia vyjadrili túžbu po atraktívnejšom rozhraní a navrhli ďalšie vylepšenia, ako je import úloh zo súboru (napr. CSV), integráciu s Google Calendar a samostatnú sekciu pre používateľov aj administrátorov, kde by sa ukladali informácie o zárobkoch.

Testovanie potvrdilo prínosy pluginu, ako je zvýšená efektivita plánovania a spokojnosť organizátorov, v súlade s cieľmi práce. Na základe spätnej väzby je vhodné do budúcnosti vylepšiť dizajn rozhrania, aby bolo vizuálne príťažlivejšie a jednoduchšie na navigáciu, a implementovať navrhnuté funkcie, ako import/export dát a integráciu s externými nástrojmi.

Záver

Táto bakalárska práca vyvinula plugin *eventer* pre Redmine, ktorý automatizuje priradenie účastníkov k úlohám na podujatiach, ako je plánovanie športových vysielaní, kde manuálne procesy spôsobujú neefektívnosť a chyby. Na integráciu nových funkcionalít do systému Redmine sa využili Patches a Hooks. Implementovali sme algoritmy celočíselného lineárneho programovania (ILP) s knižnicou PuLP a optimalizačným modulom CBC, greedy algoritmus a backtracking. ILP dosahuje optimálne priradenia aj pri zložitých obmedzeniach, ako sú dni voľna, zatiaľ čo greedy algoritmus a backtracking majú limity, ako čiastočné riešenia či absenciu výsledkov.

Hlavným prínosom pluginu je zefektívnenie riadenia podujatí na mojom pracovisku – vysielacej spoločnosti. *Eventer* eliminuje manuálne tabuľky a e-maily, šetrí manažérom čas a znižuje chyby pri priradení potrebných kvalifikovaných ľudí. Automatizácia zohľadňuje zručnosti, dostupnosť a preferencie, čím zvyšuje spokojnosť zamestnancov a kvalitu vysielania. Plugin je navrhnutý univerzálne, aby podporoval rôzne odvetvia, kde organizátori benefitujú z flexibilného priradzovania účastníkov k úlohám, prispôbeného ich požiadavkám.

Napriek prínosom má riešenie obmedzenia, ako časová náročnosť ILP pri veľkých dátach a nedokonalosti greedy algoritmu. Budúci vývoj by mal optimalizovať výkon ILP, vylepšiť filtre a integrovať dynamické webové rozhranie. Plugin *eventer* poskytuje robustné a škálovateľné riešenie, ktoré zlepšuje organizáciu podujatí a otvára možnosti pre širšie uplatnenie v riadení projektov.

Literatúra

- [1] Atlassian Corporation. Jira Software: Project Management and Issue Tracking. <https://www.atlassian.com/software/jira>. Accessed: 2025.
- [2] COIN-OR. CBC (Coin-or Branch and Cut) Solver Documentation. <https://github.com/coin-or/Cbc>. Accessed: 2025.
- [3] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 3rd edition, 2009.
- [4] Cvent, Inc. Cvent: Event Management Software. <https://www.cvent.com/>. Accessed: 2025.
- [5] Python Software Foundation. Unit Testing Framework (Python 3.13 Documentation). <https://docs.python.org/3/library/unittest.html>. Accessed: 2025.
- [6] Ganttlic. Ganttlic: Resource Planning and Scheduling Software. <https://www.ganttlic.com>. Accessed: 2025.
- [7] Jean-Philippe Lang. Redmine plugin hooks list. https://www.redmine.org/projects/redmine/wiki/Hooks_List, 2008. Accessed: 2025.
- [8] Ruby on Rails Core Team. Rails routing from the outside in. <https://guides.rubyonrails.org/routing.html>. Accessed: 2025.
- [9] PuLP Team. PuLP: A Linear Programming Toolkit for Python. <https://coin-or.github.io/pulp/>. Accessed: 2025.
- [10] Redmine Community. Redmine: Flexible Project Management Web Application. <https://www.redmine.org>. Accessed: 2025.
- [11] Laurence A. Wolsey. *Integer Programming*. Wiley, New York, NY, 1998.

Príloha A: obsah elektronickej prílohy

V elektronickej prílohe priloženej k práci sa nachádza zdrojový kód programu spolu s používateľskou a inšalačnou príručkou. Všetky súbory sú zverejnené aj na stránke <https://github.com/magula12/eventer>.