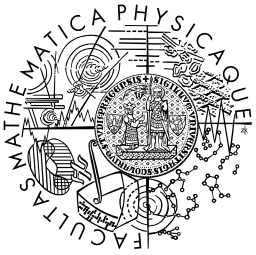


MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Ondřej Varga

Hybridní doporučování pro doménu knih

Katedra softwarového inženýrství

Vedoucí bakalářské práce: Mgr. Peška Ladislav, Ph.D.

Studijní program: Informatika

Studijní obor: Informatika se specializací
Programování a vývoj software

Praha 2023

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ....

Podpis autora

Rád bych poděkoval panu Mgr. Ladislavu Peškovi, Ph.D, vedoucímu mé bakalářské práce, za odborné vedení, podmětné rady a vstřícnost. Dále děkuji rodině, přítelkyni, přátelům a kolegům, za jejich trpělivost a podporu při studiu.

Název práce: Hybridní doporučování pro doménu knih

Autor: Ondřej Varga

Katedra: Katedra softwarového inženýrství

Vedoucí bakalářské práce: Mgr. Peška Ladislav, Ph.D., Katedra softwarového inženýrství

Abstrakt: Bakalářská práce se zabývá tématem doporučovacích systémů, které jsou důležité zejména v e-commerce oblasti. Hlavním cílem práce bylo implementovat doporučovací systém, který by pokrýval potřeby doporučování v knižní doméně (na e-shopu zabývající se prodejem knih). Těžištěm práce je implementovaný doporučovací systém, který zpracovává data dostupná pro doménu knih, nicméně je navržen obecněji tak, aby bylo možné jeho nasazení jako Recommendation-as-a-Service. Systém obsahuje jak samotné doporučovací algoritmy (kolaborativní, content-based i hybridní), tak i podporu pro jednotlivé fáze životního cyklu doporučovaní, monitorování výkonosti doporučení a snadnou administraci pomocí interaktivního webového rozhraní. Doporučovací algoritmy jsme následně vyhodnotili, přičemž z experimentů plyne, že kolaborativní metody, zejména ALS faktorizace matic a ELSA, dosahují lepších výsledků s ohledem na metriky relevance. Nicméně hybridní přístupy a content-based metody mohou mít výhody, co se týká metrik nad rámec přesnosti, zejména Coverage a Novelty.

Klíčová slova: Hybridní doporučovací systémy knihy NLP BERT

Title: Hybrid recommender system for books domain

Author: Ondřej Varga

Department: Department of software engineering

Supervisor: Mgr. Peška Ladislav, Ph.D., Department of software engineering

Abstract: The bachelor thesis deals with the topic of recommender systems, which are especially important in e-commerce field. The main goal of the thesis was to implement a recommender system that would cover the needs of recommending in the book domain (on an e-shop dealing with book sales). The focus of the work is the implemented recommender system that handles the data available for the book domain, however, it is designed more generally to be deployable as Recommendation-as-a-Service. The system includes both the recommendation algorithms themselves (collaborative, content-based and hybrid), as well as support for the different phases of the recommendation lifecycle, recommendation performance monitoring and easy administration through an interactive web interface. We then evaluated the recommendation algorithms, with experiments showing that collaborative methods, in particular ALS matrix factorization and ELSA, perform better with respect to relevance metrics. However, hybrid approaches and content-based methods may have advantages with respect to beyond accuracy metrics, especially Coverage and Novelty.

Keywords: Hybrid recommender systems books NLP BERT

Obsah

Úvod

Doporučovácí systémy se za posledních pár let staly běžnou součástí našich životů. Jejich hlavní náplní je doporučovat perzonalizovaný obsah uživatelům na základě jejich preferencí a zájmů. V době, kdy je svět zahlcen daty, se doporučovací systémy začali hojně využívat prakticky všude, kde je potřeba uživateli prezentovat proporčně menší část obsahu z jinak velikého katalogu položek. Setkat se s nimi můžeme například při výběru filmu, na který se večer podíváme, hledání článků o událostech, které se staly ve světě nebo při nákupu knižních titulů na e-shopových stránkách. Právě na poslední zmíněnou doménu se zaměříme.

V této práci navrhneme doporučovací systém pro doménu knížek. Knížky budeme doporučovat pomocí známých metod a modelů kolaborativního filtrování, pro které využijeme interakce zákazníků. V našem případě se bude jednat o historii nákupů. Dále se také zaměříme na moderní technologie pro analýzu obsahu knížek a jejich použitelnost pro content-based doporučování. Obsahem knížek pro nás budou textové popisy nebo krátké úryvky na základě kterých, určíme míru jejich podobnosti.

Doporučovací systém vytváříme pro středně velkou společnost poskytující věrnostní programy zákazníkům a zabývající se zpracováním zákaznických dat. Naší motivací je poskytnout společnosti možnost personalizovaného přístupu ke svým zákazníkům, pomocí různorodých metod a technik doporučování. Doporučovací systém implementujeme jako webovou službu s architekturou Recommendation-As-a-Service (RaaS) a REST rozhraním pro vnější komunikaci. Funkčními požadavky systému jsou:

A) **Doporučování**

B) **Manipulace s daty**

C) **Prostředí a správa systému**

D) **REST rozhraní pro vnější komunikaci**

E) **Webové rozhraní pro demo prezentaci**

Nakonec doporučovací systém vyhodnotíme pomocí experimentů, kde hlavní výzkumnou otázkou bude použitelnost textových popisů knížek pro content-based doporučování, jak samostatně, tak i kombinované s kolaborativním filtrováním v rámci hybridních přístupů.

1. Doporučovací systémy

V této kapitole se čtenář seznámí s doručovacími systémy, základními metodami doporučování a moderními technologiemi pro zpracování přirozeného jazyka.

1.1 Popis problematiky

V dnešní době je na internetu veliké množství produktů a vybrat si z nich, může být velice náročné, pokud je všechny dobře neznáme. Doporučovací systémy nám s tímto můžou pomoci. Doporučují produkty, které by se nám mohli líbit na základě našich preferencí a můžou nám tak pomoci například s výběrem knížky z našeho oblíbeného žánru. Při dalším popisu principů doporučování budeme používat označení položka pro objekty, které jsou subjektem doporučení.

Hlavním principem doporučovacích systémů je shromažďovaní a zpracování zpětné vazby od uživatelů a její následné využití k odhadování uživatelských preferencí. Zpětnou vazbu si definujeme pro jednoduchost jako hodnocení položek uživatelem. Uživatel si tak vytváří svůj profil, který určuje jeho preference k jím již hodnoceným položkám. Pomocí profilu uživatele doporučovací systémy dokáží odhadnout preference zatím nehodnocených položek a tím vytvořit pro uživatele výsledné doporučení. Doporučovací systémy se dělí na tři hlavní techniky doporučování podle metody určování uživatelských preferencí, které si nyní popíšeme a uvedeme základní reprezentanty.

1.1.1 Kolaborativní filtrovaní (CF)

Kolaborativní filtrovaní[1] je technika, využívající podobnosti mezi uživatelskými profily na základě jejich interakcí s položkami. Modely kolaborativního filtrování používají k trénování svého stavu matici interakcí. Sloupce matice reprezentují jednotlivé položky, řádky matice reprezentují uživatele a data matice udávají váhu hodnocení konkrétní položky daným uživatelem.

Prvním typem modelů využívající kolaborativní filtrování, je **Metoda K nejbližších sousedů (KNN)**[2] popsaná v práci (Ahuja a kol., 2019). Metoda je založena na principu podobností mezi uživateli (user-based) nebo položkami (item-based). Metoda určuje podobnost na základě zvolené podobnostní funkce. Nejčastěji se využívá Euklidovská vzdálenost, Kosinová podobnost nebo Pearsonův korelační koeficient. Ke každé položce nebo uživateli metoda nalezne k nejpodobnějších sousedů, které následně agreguje do výsledného doporučení. Pro uživatele se doporučují položky, které preferují jeho nejbližší sousedé nebo položky s vysokou podobností k jeho preferencím.

Velmi populární metodou CF je **Faktorizace Matic (MF)**[3]. Metoda rozkládá vstupní matici interakcí na dvě řádově menší, jejichž maticovým násobení opět získá matici původní. Menší matice reprezentují jednotlivé položky a uživatele v latentním prostoru o nižší dimenzi. Velikost dimenze latentního vektoru

[1] Collaborative Filtering
[2] K Nearest Neighbours
[3] Matrix Factorization

se udává počtem tak zvaných faktorů. Latentní vektor je kompaktní reprezentací vstupních dat v nízko rozměrném prostoru. Vektor obsahuje skryté vzorce a charakteristiky dat, které jsou relevantní pro modelování preferencí a generování doporučení. Typy faktorizace matic se odlišují použitou ztrátovou funkcí a způsobem trénování. Nejznámějšími jsou například Alternating Least Squares (ALS) (Pilászy a kol., 2010) nebo Bayesian Personalized Ranking (BPR) Rendle a kol., 2012.

Další velkou skupinou modelů CF, která se v dnešní době často používá, jsou lineární autoenkódery. Ty se skládají ze dvou částí kodér a dekodér. Kodér pomocí určené lineární transformace převádí uživatelské interakce do latentních vektorů. Dekodér následně převádí latentní vektory zpět na rekonstruovaná vstupní data opět pomocí lineární transformace. Trénování probíhá optimalizací ztrátové funkce, která minimalizuje rozdíli mezi vstupními a rekonstruovanými daty. Nejvýznamnějším modelem z této skupiny je **Embarrassingly Shallow Autoencoders for Sparse Data (EASE)** (Steck, 2019) a to především díky své jednoduchosti a přesnosti odhadů uživateslkých preferencí. EASE řeší optimalizační problém 1.1, kde $\mathbf{X}$ označuje matici interakcí, λ je volitelný parametr a $\mathbf{B}$ je matice latentního prostoru o rozměrech `N x N`, kde `N` označuje počet položek. Podmínka hodnot na diagonále matice `B` zabraňuje triviálním řešením, například identitě. Největší výhodou modelu je možnost jeho trénování pomocí explicitního vzorce 1.2. $\mathbf{X}$ označuje vstupní matici interakcí, $\hat{\mathbf{B}}$ je výstupní matice o rozměrech `N x N`. Operace $\bigotimes$ značí dělení po prvcích, jenž je dobře definované, protože $\hat{\mathbf{P}}$ je invertibilní.

$$\min_{\mathbf{B}} \|\mathbf{X} - \mathbf{X}\mathbf{B}\|_2^2 + \lambda \|\mathbf{B}\|_2^2 \quad \text{s.t.} \quad \text{diag}(\mathbf{B}) = 0 \tag{1.1}$$

$$\hat{\mathbf{B}} = \mathbf{I} - \hat{\mathbf{P}} \cdot \text{diagMat}\left(\overrightarrow{1} \bigotimes \text{diag}(\hat{\mathbf{P}})\right) \quad \text{kde} \quad \hat{\mathbf{P}} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1} \tag{1.2}$$

Navzdory všem jeho výhodám je model EASE nepoužitelný pro reálné nasazení na klasických datových sadách. Ve vzorci 1.2 se počítá s inverzní maticí $\hat{\mathbf{P}}$ jenž má rozměry `N x N`. Klasické datové sady mají většinou desítky tisíc položek, což tento výpočet znemožňuje. Existují však faktorizační metody, které matici $\hat{\mathbf{P}}$ adekvátně rozdělují na menší části. Příkladem je model **Scalable Linear Shallow Autoencoder for Collaborative Filtering (ELSA)** představený v článku (Vančura a kol., 2022).

1.1.2 Doporučení založené na obsahu (CB)

Doporučení založené na obsahu[4] se zaměřuje na analýzu obsahu položek a snaží se doporučovat položky podobné těm, se kterými uživatel již interagoval. Tento přístup nevyužívá data interakcí ostatních uživatelů, to přináší určitou výhodu oproti CF v situacích, kdy doporučujeme nové položky, které mají málo interakcí s uživateli. Takové situaci se v oblasti doporučovacích systémů říká cold-start problem. Základním zdrojem dat, na kterých CB doporučovací modely závisí, jsou obsahově orientované atributy, které popisují položky, a uživatelské

[4]Content-based

profily, jenž jsou vytvořené ze zpětných vazeb uživatelů k položkám. (Aggarwal, 2016).

Historicky se pro CB doporučování používala metoda **Term Frequency - Inverse document frequency (TD-IDF)**, která analyzuje textové data položek. Jedná o statistickou metodu, jenž slouží k ohodnocení důležitosti slov (termů) v textových dokumentech na základě jejich frekvence v celém korpusu (množině všech dokumentů). V kontextu doporučování položek si pod textovým dokumentem můžeme představit konkatenaci klíčových slov popisující danou položku. TF-IDF transformuje textový dokument položky do číselného vektoru, jenž reprezentuje danou položku v nízko dimenzionálním prostoru. Na základě podobností vektorů se následně vytváří výsledné doporučení. Použití této metody pro doporučování položek můžeme nalézt v článku (Chiny a kol., 2021).

Od té doby se v oblasti CB doporučování začaly využívat modernější přístupy pro zpracování obsahově orientovaných atributů. Jedním z nich je využití modelu **Bidirectional Encoder Representations from Transformers (BERT)** (Devlin a kol., 2018), jenž je založený na architektuře **Transfomer**, která byla představena v publikaci (Vaswani a kol., 2017). Jedná se o jednu z nejvýznamnějších architektur pro zpracování přirozeného jazyka (NLP)[5]. Tato architektura umožňuje efektivně zachytit vztahy mezi slovy v daném kontextu. Klíčovou inovací modelu BERT je jeho schopnost pracovat s oboustranným kontextem slov ve vetách. BERT model je tak schopen lépe pochopit význam slov a zachytit sémantickou podobnost textů. Pro CB doporučování se využívá upravená varianta modelu **Sentence BERT (SBERT)** (Reimers a Gurevych, 2019), jenž umožňuje efektivně převádět větší textové celky do kontextově významných vektorových reprezentací, které lze porovnávat pomocí kosinové podobností.

1.1.3 Hybridní přístupy

Hybridní doporučovací systémy kombinují dvě nebo více doporučovacích technik, za účelem dosáhnutí lepších výsledků při doporučování a minimalizace nevýhod kombinovaných technik. Nejčastěji se kombinuje CF s dalšími technikami doporučování ve snaze vyhnutí se cold-start problému. Podle (Burke, 2002) můžeme rozdělit hybridní přístupy do těchto základních kategorií.

- **Weighted**: hybridní přístup, který kombinuje dílčí doporučení jednotlivých modelů různých technik do jednoho výsledného. Přístup kombinuje doporučené položky například metodou **BordaCount**[6], nebo na základě jejich skóre pomocí váženého průměru.
- **Switching**: přístup přepíná mezi doporučovacími modely v závislosti na aktuálních potřebách. Například využití CB doporučovací techniky nebo nepersonalizovaného doporučování jako řešení cold-start problému.
- **Cascade**: přístup, jenž výstup jednoho modelu předává na vstup druhého modelu. Jinými slovy druhý model v kaskádě upřesňuje doporučení, které bylo poskytnuto prvním modelem respektive doporučovací technikou.

[5]Natural language processing

[6]`https://en.wikipedia.org/wiki/Borda_count`

- **Feature combination**: označovaný také někdy jako monolitický přístup, který kombinuje zdroje dat, jenž využívá v rámci trénování. Příkladem monolitického systému může být model **Visual Bayesian Personalized Ranking from Implicit Feedback (VBPR)** (He a McAuley, 2015), který kombinuje interakční data s vizuálními daty doporučovaných položek. Model vychází z metody faktorizace matic, přičemž latentní vektory položek vznikají konkatenací jejich vektorových reprezentací z různých zdrojů. Pro CB část modelu se navíc používá feature-selection, za účelem zredukování výsledné dimenze latentního vektoru. Pro splnění cílů této práce využijeme podobné řešení, které bude využívat pro CB část dříve zmíněný **SBERT**.

2. Analýza

V této kapitole si popíšeme, jaké výhody přináší personalizované doporučování produktů v oblasti e-commerce, pomocí jakých metod se doporučení dá použít, a jaké z toho plynou požadavky na námi vyvíjený doporučovací systém.

2.1 Doporučování pro e-commerce

Doporučování produktů v oblasti e-commerce se využívá hlavně z důvodu výhod, které poskytuje, a umožnění personalizovaného přístupu k zákazníků.

2.1.1 Výhody doporučování

Personalizované doporučení produktů přináší výhody jak pro prodávající, tak i pro zákazníky e-commerce platformy. Doporučování umožňuje personalizovat nabídku produktů na základě preferencí, chování a historie nákupů zákazníka. Personalizované nabídky zvyšují spokojenost zákazníku, protože jim umožňují snadněji najít a objevovat produkty, které odpovídají jejich zájmům a preferencím. Také napomáhají k zlepšení orientace zákazníků v rozsáhlém sortimentu produktů, jenž prodávající nabízejí. Obecně zákazníci mohou pomocí kvalitních doporučení nabývat pozitivních zkušeností, které vedou ke zvýšení jejich loajality a důvěry k e-commerce platformě.

Doporučování produktů vede ke zvýšení konverzí a prodejů. Zákazníkům jsou nabízené relevantní produkty, čímž se zvyšuje pravděpodobnost jejich nákupu. Častými technikami v oblasti e-commerce ve spojitosti s doporučování jsou cross-selling a up-selling. První z nich je technika, při které se zákazníkovi nabízejí doplňkové nebo podobné produkty. Pomocí druhé techniky se zas nabízejí zákazníkovi vyšší verze produktů nebo produkty, které jsou dražší. Zákazníkovi je tak nabízeno více relevantních produktů, což vede ke zvýšení diversity prodejů, jak uvádí (Fleder a Hosanagar, 2007), a zvýšení průměrné hodnoty objednávky.

Důležitou vlastností doporučování je také zvýšení konkurenceschopnosti e-commerce platforem. Kvalitní doporučování poskytuje v oblasti e-commerce konkurenční výhodu a pomáhá přilákat zákazníky, jenž preferují personalizovaný přístup.

2.1.2 Využití doporučení

Využití doporučení záleží na možnostech e-commerce platformy, jenž nabízí své produkty. Doporučování lze nejlépe využívat například na domácí stránce e-shopu, v detailu konkrétního produktu nebo pomocí emailových kampaní. Existují tři hlavní způsoby, jak lze doporučení využívat, což poté klade i požadavky na modely které doporučení realizují. Způsoby, kterými lze využít doporučování jsou odvozené od výhod, jenž doporučování umožňuje. Nyní popíšeme základní metody pro doporučování, které následně využijeme pro vlastní realizaci doporučovacího systému.

- **Items to user**: Metoda doporučující produkty zákazníkovi. Metoda vyžaduje znalost zákazníka, pro kterého vytváří doporučení produktů. Znalostí

myslíme například demografické údaje, chování nebo historii nákupů zákazníka. Příkladem využití je doporučení produktů v obecném kontextu. Například je-li zákazník na domácí stránce e-shopu, doporučení může být využito pro nabídku relevantní produktů. Dalším využitím je personalizovaná nabídka produktů v emailových kampaních.

- **Items to item**: Metoda, která doporučuje substitutivní produkty v kontextu konkretního referenčního produktu. Příkladem využití této metody je doporučování vhodných alternativ k právě zobrazenému produktu. Metoda napomáhá zákazníkovi představit širší nabídku produktů a umožňujeme mu snazší výběr produktu, který nejvíce odpovídá jeho preferencím. Metodu prodávající může využít k doporučení dražších produktů nebo produktů vyšší kategorie (technika up-selling).

- **Items to cart**: Metoda, která se využívá pro doporučení doplňkových produktů. Přímím příkladem je využití doporučování produktů do nákupního košíku. Zákazníkovy jsou nabízeny produkty, které se nejvíce hodí k již vybraným produktů. Metoda zákazníkovi usnadňuje hledání vhodných doplňků a zároveň zvyšuje průměrnou hodnotu objednávek.

2.2 Knižní doména

V naší práci se zabýváme doporučovaním pro doménu knížek. V této sekci si popíšeme hlavní vlastnosti této domény a datovou sadu, kterou v práci použijeme.

2.2.1 Vlastnosti domény

Doména knížek je specifická svojí rozsáhlou nabídkou a bohatou datovou strukturou. Nabízí širokou škálu knižních titulů, které jsou různých žánrů, témat a od různých autorů. K objektům v knižní doméně je obvykle možné přiřadit strukturovaná data, která obsahují informace jako jsou titul, stručný obsah děje, autor, žánr, klíčová slova nebo URL adresa obrázku knížky. Doména taktéž poskytuje dobře definované vazby mezi knížkami. Například že je knížka součástí série, stejného žánru nebo od stejného autora. Tyto vazby pomáhají určit míru vzájemné podobnosti knížek. Další vlastností domény je častá dlouhodobá relevance pro čtenáře. Knižní tituly jsou často dostupné čtenáři po delší dobu a jejich preference se nemění tak rychle, jako u jiných domén.

2.2.2 Použitá datová sada

Požadavkem této práce je využití reálných interakčních dat e-commerce společnosti, jenž se zabývá prodejem knih. Pro účely vývoje a vyhodnocení doporučovacího systému, máme k dispozici datovou sadu od námi vybrané společnosti která obsahuje skutečné interakce zákazníků a základní informace o nabízených knížkách. Datová sada je uložena v relační databázi společnosti, proto o kolekcích sady budeme mluvit jako o tabulkách.

Interakce zákazníků mají význam obchodních transakcí a udávají informace o nákupech knížek. Jednotlivé transakce se sdružují do objednávek, které zákazníci

vytvářejí. Objednávka obsahuje unikátní číslo a časovou známku, kdy byla vytvořena. Interakce se upraví do požadovaného formátu sloučením tabulek transakcí a objednávek. Výsledný záznam interakce obsahuje identifikátor zákazníka, identifikátor nakoupené knížky, číslo objednávky a časovou známku. Interakce mají implicitní charakter a v datové sadě platí, že se interakce zákazníka s konkrétní knížkou vyskytuje nejvýše jednou.

Datová sada obsahuje tabulku se základními informacemi o nabízených knížkách. Záznam knížky obsahuje unikátní číselný identifikátor, název a International Standard Book Number (ISBN)[1], jenž jednoznačně identifikuje vydání dané knížky. Tyto základní informace bychom potřebovali doplnit o další vlastnosti, jako jsou například textové popisy nebo krátké úryvky, které stručně popisují a reprezentují obsah knížky. Dostupná datová sada bohužel další vlastnosti knížek neobsahuje, a proto jsme nuceni je získat z externích datových zdrojů. I když pracujeme s jednou konkretní datovou sadou, podobné informace bychom našli i na jiných e-shopech s knihami.

2.2.3 Externí zdroje dat

Externí datové zdroje obsahují volně dostupná strukturovaná data, která jsou připravená k dalšímu zpracování. Externí zdroje dat jsme využili k doplnění dodatečných informací o knížkách, především jejich textových popisů. Pro naše účely jsme využili zdroje dat, kterými jsou Goodreads[2], DBpedia[3] a Wikidata[4]. Všechny tyto zdroje obsahují datové kolekce knížek, ze kterých jsme chtěli získat požadované doplňující informace k naší datové sadě mapováním pomocí ISBN. Bohužel jsme z použitých zdrojů nedokázali pokrýt dostatečně velkou část naší datové sady z důvodu obtížného mapování nebo chybějícím informacím. Proto jsem byli nuceni využít záložní řešení, kterým bylo získání dodatečných informací z e-shopu společnosti, jejíž interakční data máme k dispozici.

Dodatečné informace jsme získali extrakcí dat z webových stránek e-shopu pomocí implementovaných metod, jenž se nachází v příloze této práce. Extrakce dat spočívala ve zpracování HTML DOM (Document Object Model) struktury přehledu knížky, jenž jsme vyhledali na stránkách e-shopu pomocí známého ISBN. Hodnoty jednotlivých elementů DOM struktury jsme ukládali do datových souborů a následně sloučili s datovou sadou.

2.3 Požadavky na doporučovací systém

Cílem naší práce je navrhnout a implementovat doporučovací systém pro námi vybranou středně velkou e-commerce společnost, jenž se zabývá prodejem knih. Doporučovácí systém bude implementován jako webová služba poskytující doporučování pomocí Representational State Transfer (REST) rozhraní. Systém nebude vázaný na konkrétní doménu a bude poskytovat doporučení souběžně různým externím aplikacím, jenž s ním budou komunikovat. Pro tento monolitický návrh systému jsme se rozhodli kvůli požadavkům společnosti, pro kterou systém

[1] `https://cs.wikipedia.org/wiki/International_Standard_Book_Number`
[2] `https://www.goodreads.com/`
[3] `https://www.dbpedia.org/`
[4] `https://www.wikidata.org/wiki/Wikidata:Main_Page`

vytváříme. Tato architektura vychází z potřeby provozovat systém pro vícero cílových domén nebo e-shopů, Jedná se o typický požadavek kladený na systémy typu Recommendation as a Service (RaaS)[5], který je v praxi velmi rozšířený a odpovídá způsobu, jak chceme platformu v budoucnu nasazovat.

Souhrn požadavků systému uvedeme pomocí seznamu, jehož body následně více rozebereme a popíšeme.

A) Doporučování
   (a) kolaborativní filtrování (CF)
   (b) doporučení založené na obsahu (CB)
   (c) hybridní přístupy

B) Manipulace s daty
   (a) získání dat z nastavených zdrojů
   (b) zpracování dat

C) Prostředí a správa systému
   (a) administrace systému
   (b) automatizace doporučovacího cyklu

D) REST rozhraní pro vnější komunikaci

E) Webové rozhraní pro demo prezentaci

Co se týká požadavků na typy implementovaných algoritmů, výsledný systém by měl umožnit doporučení podle tří základních doporučovacích technik, kterými jsou CF, CB a hybridní doporučení, jenž kombinuje oba předchozí přístupy. CF se zaměřuje na vyhledávaní podobných zákazníků nebo produktů podle vzorců jejich předchozích interakcí. Například pokud zákazník A nakupuje podobně jako zákazník B, může CF zákazníkovi A doporučit produkty, které preferuje zákazník B. CB je založeno na analýze obsahu samotných produktů a porovnávání s preferencemi daného uživatele. CB využívá vlastnosti produktů k určení míry podobnosti mezi nimi a může například doporučit knížky stejného žánru nebo od stejného autora, jenž daný zákazník preferuje. CF a CB jsou výrazně různorodé, tím pádem bychom chtěli, aby měl systémový návrhář rozumnou volbu v tom, jaké doporučení použít pro danou situaci. Například pro doporučení metodou Items to item využít podobnosti produktů na základě jejich obsahu a pro doporučení do nákupního košíku využít podobnosti produktů na základě zákaznických interakcí. Zároveň bychom také chtěli zpřístupnit možnost syntézy obou hlavních typů podobností v hybridním přístupu.

Zbylé požadavky na systém si popíšeme spolu se životním cyklem modelů, jenž se zabývá doporučovacími modely od jejich návrhu, realizaci a trénování až po jejich nasazení a persistenci.

[5]Doporučení jsou generována a poskytována jako cloudová služba, což umožňuje různým e-commerce platformám integrovat doporučovací funkce do svých vlastních aplikací a webových stránek.

2.3.1 Životní cyklus doporučovacích modelů

Životní cyklus doporučovacích modelů se skládá ze čtyř hlavních fází, kterými jsou analýza požadavků, datově orientovaná fáze, modelově orientovaná fáze a fáze provozní, jak se uvádí v průzkumu Schlegel a Sattler 2023. Ten se zabývá obecnou formou modelů strojového učení, proto z něj můžeme vycházet i pro modely určené k doporučování.

První fáze má za cíl určit funkce, rozhraní a typ modelu, který nejlépe řeší danou problematiku. V našem případě bude sytém implementovat modely různých doporučovacích technik a principů, které budou poskytovat doporučení pomocí dříve zmíněných metod pro využití doporučení.

Druhá fáze se zaměřuje na získání a přípravu dat, které se použijí pro trénování doporučovacích modelů. Architektura systému není vázaná na konkrétní datovou sadu, čili systém bude umožňovat snadnou integraci různých datových sad, na kterých bude možné doporučovací modely trénovat. Dále bude možné pro každou externí aplikace, jenž komunikuje se systém, nastavit kontext její datové sady. Kontext bude udávat informace spojené se zdrojem datové sady a časový interval, po jehož uplynutí bude datová sada automaticky aktualizována. Různé datové sady se budou po získání transformovat do obecné formy, jenž mohou následně zpracovávat všechny implementované modely.

Třetí fáze se zabývá samotnými doporučovacími modely a je rozdělena do třech kroků. Prvním krokem je trénování modelů. Systém bude umožňovat jak manuální tak automatické spuštění trénovacího procesu. Každému modelu bude možné nastavit trénovací kontext, který bude udávat hodnoty jeho hyper-parametrů a časový interval, po jehož uplynutí bude model automaticky znovu natrénován. Dalšími kroky je vyhodnocení výkonosti doporučování modelů a jejich případná optimalizace. Systém bude poskytovat rozhraní pro spuštění experimentů, jenž budou natrénované modely vyhodnocovat na vybraných metrikách a následně optimalizovat výběr hodnot jejich hyper-parametrů.

Poslední fáze se zaměřuje na provoz doporučovacích modelů. Po natrénovaní modelů, je nutné zajistit jejich persistenci, aby bylo možné je později využít pro doporučování. Systém bude ukládat natrénované doporučovací modely do datového úložiště, odkud je bude následně načítat dle potřeby. Dále bude umožňovat kontrolu a správu verzí natrénovaných modelů. Pokud například bude mít nově natrénovaný model horší výkonost než jeho předchozí verze, bude možné se k této verzi vrátit. Novou verzí modelu v tomto případě myslíme ten samý natrénovaný na aktualizované datové sadě. Aby bylo možné zjistit výkonost doporučování modelu, systém bude umožňovat vytvoření zpětné vazby pro konkrétní doporučení a samotný přehled vyhodnocení výkonosti doporučování konkretního modelu dané verze.

2.3.2 Komunikace se systémem

Nejdůležitějšími požadavky systému je umožnění vnější komunikace. Systém bude poskytovat REST rozhraní pro vnější komunikaci, jenž bude umožňovat přístup k funkcím systému, které jsme si popsali dříve. Rest rozhraní je standardní způsob komunikace se systémem, které však není uživatelsky přívětivé. Doporučovací systém proto bude poskytovat také webové rozhraní, jenž slouží pro jeho snaží použitelnost a demonstraci. Webové rozhraní bude umožňovat přístup k funkcím

systému pomocí interaktivních webových prvků. Poslední možností komunikace se systém bude pomocí příkazové řádky. Systém bude poskytovat příkazy, které spouští proces trénování nebo systémové experimenty.

3. Architektura a návrh doporučovacího systému

V této kapitole se zaměříme na architekturu a návrh doporučovacího systému. Nejprve si připomeneme základní vlastnosti doporučovacího systému, které jsme definovali a následně analyzovali v předchozích kapitolách. A poté popíšeme strukturu systému a architekturu doporučovací aplikaci.

3.1 Základní vlastnosti systému

Doporučovací systém slouží jako služba doporučování pro externí aplikace, které se systémem komunikují pomocí Application Programming Interface (API). Externí aplikace předávají systému data, pomocí kterých systém vytváří doporučení. Data obsahují katalog položek s jejich vlastnostmi, které systém bude doporučovat, a záznamy o zpětných vazbách uživatelů externí aplikace k těmto položkám. Tyto data v systému označujeme jako datovou sadu. Výsledné doporučení obsahuje požadovaný počet položek uspořádaných sestupně podle hodnoty relevance položky k cílovému subjektu doporučení.

Systém vytváří doporučení položek s využitím různých technik doporučování zejména CF, CB nebo kombinaci obou, kterému říkáme hybridní doporučování. Systém dále umožňuje doporučovat položky pro různé cílové subjekty. Příkladem může být doporučování položek uživateli nebo doporučení do nákupního košíku. Externí aplikace mohou spravovat proces doporučování a sledovat jeho výkonost.

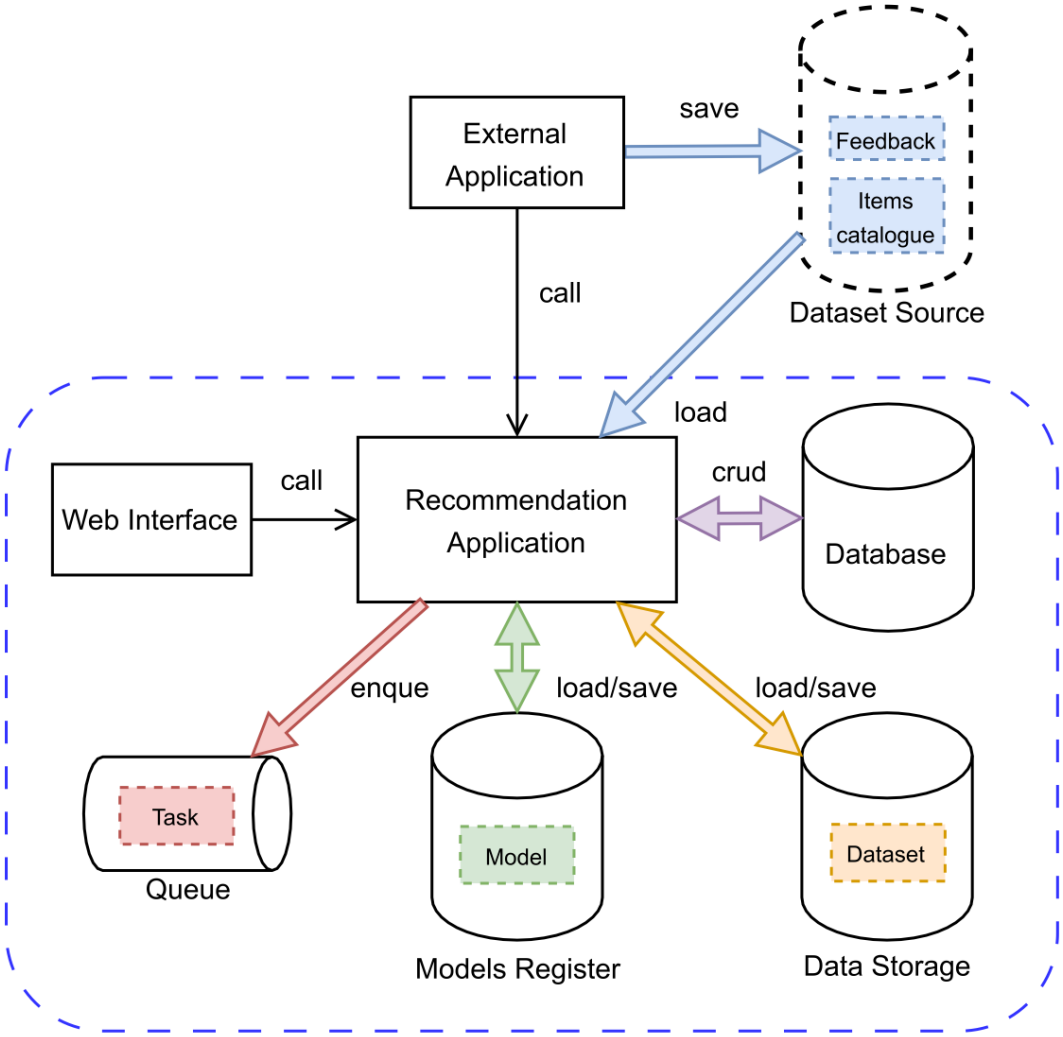
3.2 Struktura systému

Strukturu systému budeme popisovat spolu s příkladem externí aplikace. Externí aplikaci označujeme v systému jako projekt, jenž má datovou sadu, kterou předává systému za účelem vytvoření doporučení. Datová sada projektu je uložena v úložišti, jenž označujeme jako zdroj datové sady. Konkrétním příkladem může být projekt s databází kde jsou uloženy jednotlivé položky, které chce projekt doporučovat a záznamy hodnocení položek uživateli tohoto projektu.

Struktura systému obsahuje doporučovací aplikaci a databázi pro ukládání jejích dat, datové úložiště pro ukládání datových sad projektů, registr pro ukládání natrénovaných modelů, frontu pro zpracování úloh vedlejšími procesy a webové rozhraní pro komunikaci se systémem. Schéma struktury systému můžeme vidět na obrázku 3.1 spolu s příkladem externí aplikace a jejím zdrojem datové sady.

Jádrem doporučovacího systému je aplikace napsaná v Pyhonu[1], která implementuje hlavní byznys logiku. Aplikace načítá datové sady projektů z definovaných zdrojů. Sady následně zpracovává a ukládá do datového úložiště systému. Aplikace implementuje doporučovací algoritmy popsané v kapitole [odkaz], které v systému nazýváme jako modely. Modely se trénují na datových sadách projektů a vytvářejí doporučení položek. Natrénované modely ukládá aplikace do registru, čímž zajišťuje jejich persistenci. Modely se z registru opět načítají do aplikaci v

[1] `https://www.python.org/`



Obrázek 3.1: Schéma struktury doporučovacího systému

momentě, kdy jsou potřeba k vytvoření doporučení položek. Součástí systému je také databáze, která zajišťuje persistenci dat aplikace. Databáze ukládá data související s aplikací a parametry pro nastavení doporučovacího procesu jednotlivých projektů.

Systém pracuje s dvěma typy procesů doporučovací aplikace, hlavní a vedlejší. Hlavní proces aplikace zpracovává v reálném čase příchozí API požadavky, kterými jsou například vytvoření doporučení položek nebo augmentace datové sady. Účelem vedlejších procesů aplikace je zpracovávat úlohy na pozadí doporučovacího systému. Příkladem těchto úloh je trénování modelů nebo aktualizace datové sady ze zdroje projektu. Úlohy se v systému využívají pro zajištění automatizace životního cyklu doporučování a řadí se do fronty, kde čekají na zpracování.

Doporučovací systém poskytuje pro komunikaci s uživatelem webové rozhraní, které komunikuje s API aplikace a slouží především pro snazší administraci a demonstraci doporučování položek. Webové rozhraní je stejně jako doporučovací aplikace napsané v Pythonu a jeho strukturu a implementaci si popíšeme později v této kapitole.

3.3 Architektura doporučovací aplikace

Architektura aplikace se skládá z komponent, které budeme rozdělovat na moduly a vrstvy. Moduly implementují hlavní logiku a vrstvy propojují jednotlivé moduly mezi sebou a ostatními částmi systému. Schéma architektury aplikace můžeme vidět na obrázku 3.2.

Popis architektury začneme od nejdůležitější komponenty, kterou je doporučovací modul. Hlavní náplní modulu je implementování doporučovacích modelů a definování struktur datové sady. Doporučovací modely integrují algoritmy popsané v kapitole [odkaz] a implementují metody doporučovacího rozhraní. Modely se v aplikaci využívají k vytvoření doporučení položek pro konkrétní projekty. Struktura datové sady definuje schéma dat projektů. Určuje jaké atributy mají jednotlivé dokumenty datových kolekcí a jakých jsou datových typů. Modul dále implementuje metody pro úpravu a vyhodnocení výsledného doporučení.

Další komponentou doporučovací aplikace je vrstva přístupu k datům. Vrstva je zodpovědná za přenos dat mezi aplikací a datovými úložišti a vytváří abstrakci, která odstiňuje použité technologie ukládání dat. Vrstva načítá datové sady projektů z jejich zdrojů a stará se o přístup k těmto sadám uloženým v datovém úložišti systému. Krom datových sad se vrstva také stará o načítání a ukládání natrénovaných modelů v registru.

Nezbytnou komponentou aplikace je vrstva přístupu k databázi. Databáze slouží pro ukládání dat aplikace, které se řídí datovým modelem definovaným právě v této vrstvě. Vrstva implementuje strukturu repositářů pomocí kterých provádí CRUD[2] operace nad tabulkami databáze. Vrstva je také zodpovědná za udržení schématu databáze v aktuálním stavu pomocí principu migrací databázových schémat.

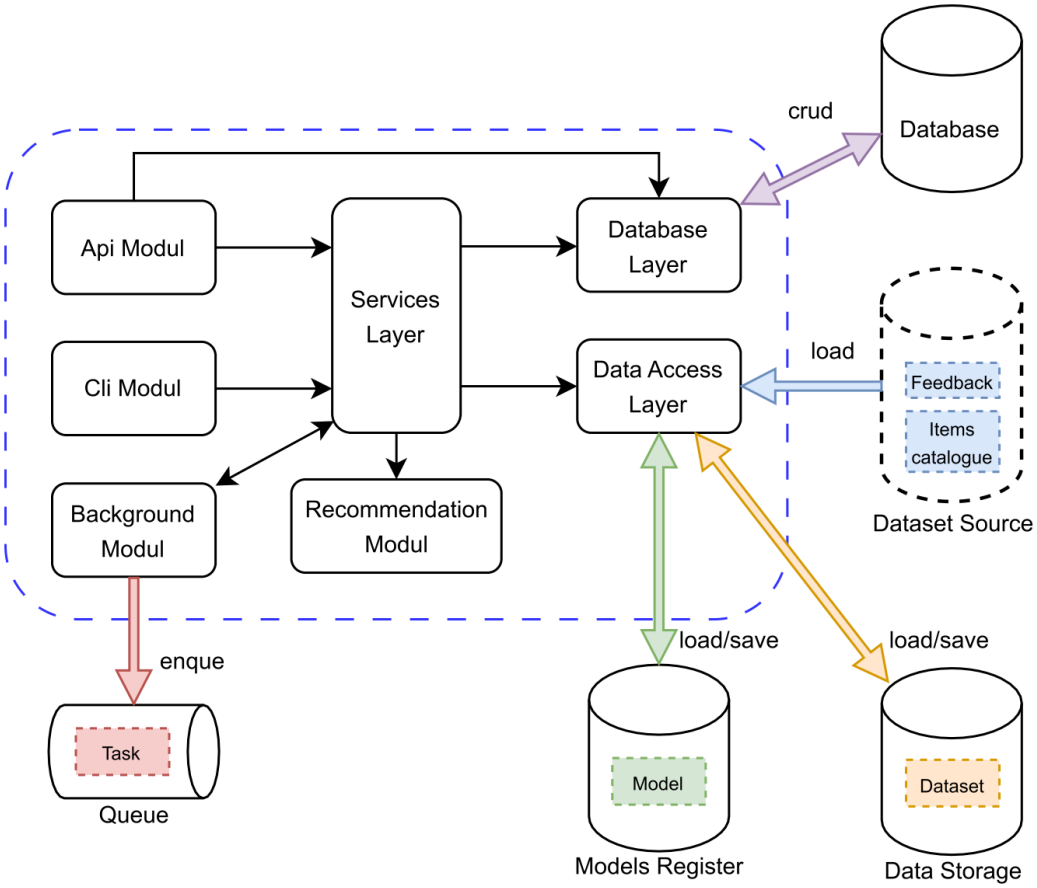
Dalším modulem doporučovací aplikace je Background modul. Tento modul definuje úlohy, které jsou zpracovávány vedlejšími procesy na pozadí doporučovací aplikace. Úlohy provádí operace trénování doporučovacích modelů a aktualizace datových sad projektů. Systém využívá úlohy pro zajištění automatizace a Background modul je zodpovědný za jejich řazení do fronty a sledování jejich stavu během zpracování.

Aplikace poskytuje pro vnější komunikaci definované rozhraní v modulech Api a Cli. Api modul implementuje Representational State Transfer (REST) aplikační rozhraní, které zpracovává a odpovídá na Hypertext Transfer Protocol (HTTP)[3] požadavky. Modul Cli implementuje rozhraní pro komunikaci pomocí příkazové řádky. Modul definuje sadu příkazů, které vykonávají operace v doporučovací aplikaci.

Poslední komponentou aplikace je servisní mezi vrstva, která propojuje ostatní komponenty. Vrstva zapouzdří hlavní funkce předchozích komponent a vytváří z nich ucelené procesy doporučovací aplikace. Příkladem jsou trénovací a doporučovací proces, jenž si podrobně popíšeme později v této kapitole.

[2]Shrnutí čtyř základních operací Create, Read, Update, Delete, které se provádějí nad záznamem uloženým v databázi.

[3]Hypertext Transfer Protocol - `https://httpwg.org/specs/rfc9110.html`



Obrázek 3.2: Schéma architektury doporučovací aplikace

3.4 Doporučovací modul

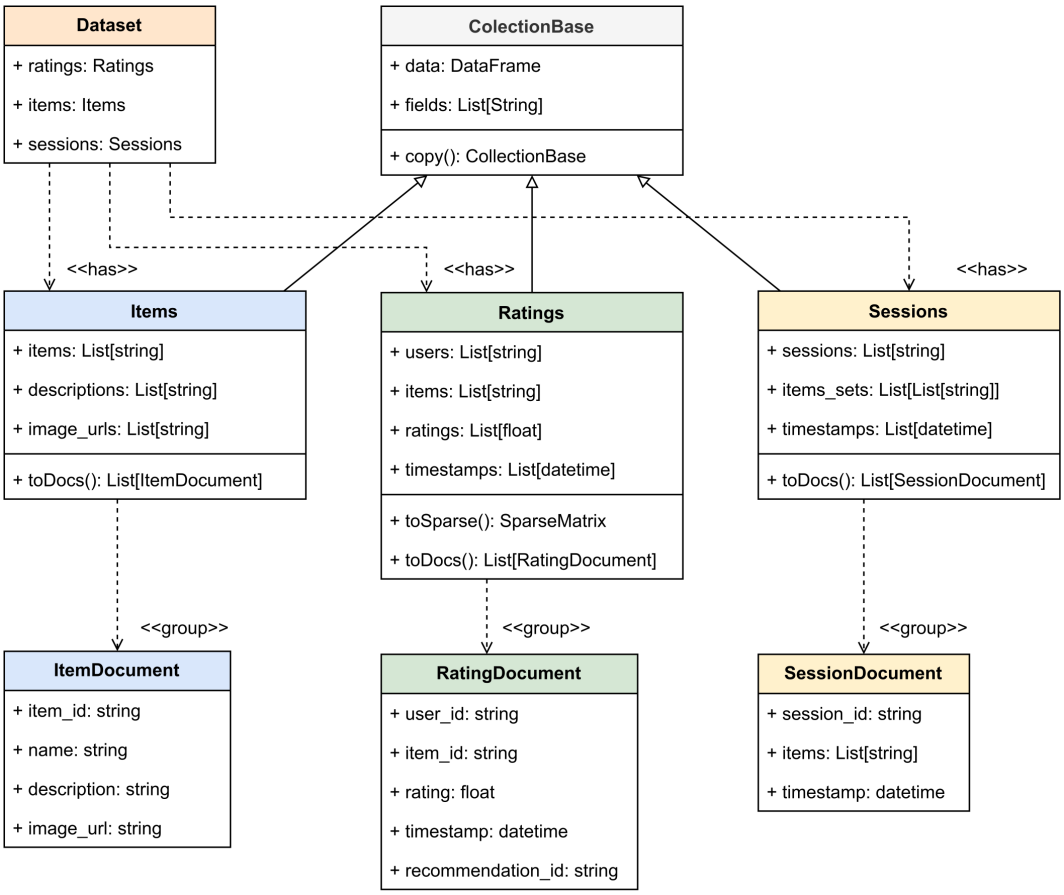
V této sekci si popíšeme základní komponentu aplikace, která implementuje hlavní logiku doporučování. Nejprve začneme strukturou datové sady, která je důležitá pro natrénování stavů doporučovacích modelů. Poté si popíšeme jednotlivé metody rozhraní pro doporučování, které definují kdo je cílovým subjektem doporučení. A nakonec si stručně popíšeme implementované modely a způsob jejich kombinace za účelem hybridního doporučování.

3.4.1 Struktura datové sady

Datová sada se skládá z různých kolekcí, které seskupují dokumenty. Dokument je datová struktura obsahující jednotlivé atributy, které mají předem definované datové typy. Data kolekce reprezentujeme tabulkou, kde jeden řádek představuje právě jeden dokument a sloupec hodnoty konkrétního atributu všech dokumentů v kolekci. Tato tabulka je implementovaná s využitím knihovny Pandas[4], která efektivně pracuje s daty pomocí vektorových operací a indexace.

Kolekce jsou v aplikaci implementovány jako třídy, které obalují zmíněnou tabulku dalšími metodami a atributy. Každá třída kolekce je potomkem abstraktní třídy **CollectionBase**, která definuje základní metody a atributy. Třídy

[4] `https://pandas.pydata.org/`



Obrázek 3.3: Schéma struktury datové sady.

kolekcí například implementují metody pro filtrování dokumentů nebo transformaci vnitřní tabulky do jiného formátu. Všechny implementované metody kolekcí jsou imutabilní. Každá třída kolekce definuje schéma, podle kterého jsou její data validovány. Schéma udává atributy a jejich datové typy, které dokumenty dané kolekce musejí mít.

Přehled kolekcí, které tvoří datovou sadu můžeme vidět ve schématu 3.3. Nyní si všechny kolekce krátce popíšeme.

- **Items**: kolekce položek. Každá položka má svůj unikátní identifikátor a název. Nepovinnými atributy jsou textový popis a url adresa obrázku dané položky. Vlastnosti položek se využívají pro doporučení založeném na obsahu.

- **Ratings**: kolekce hodnocení položek uživateli. Reprezentuje kladnou zpětnou vazbu uživatelů k položkám a využívá se k doporučení položek založené na kolaborativním filtrování. Každé hodnocení má číselnou hodnotu a časovou známku kdy bylo vytvořeno. Dokument hodnocení může obecně reprezentovat interakci uživatele s položkou a pokud není uvedena číselná hodnota interakce, počítá se s tím, že má kolekce binární charakter. Kolekci hodnocení můžeme snadno transformovat do struktury matice, kde indexy řádků představují uživatele, indexy sloupců jednotlivé položky a data matice číselné hodnoty interakcí. Tyto matice jsou při reálném použití velice řídké, a proto používáme efektivní reprezentaci zvanou řídká

matice z knihovny SciPy[5]. Dokument hodnocení navíc obsahuje nepovinný atribut identifikátoru doporučení. Ten je využit v případě, kdy je interakce hodnocení položky zpětnou vazbou uživatele na konkrétní doporučení této položky. Pomocí hodnoty tohoto atributu dokážeme zpětně vyhodnocovat výsledné doporučení.

- **Sessions**: nepovinná kolekce relací položek. Relace obsahuje sekvenci nebo skupinu položek, které byly hodnoceny pospolu. Krom toho obsahuje také unikátní identifikátor a časovou známku. Kolekce relací se používá například pro nalezení asociativních vazeb mezi položkami.

3.4.2 Doporučovací rozhraní

Rozhraní definuje metody, které mají na vstupu cílový subjekt a na výstupu jeho doporučení. Cílový subjekt doporučení reprezentuje pro koho nebo co je doporučováno a doporučení je datová struktura obsahující seznam položek uspořádaných sestupně podle relevance k cílovému subjektu. Všechny metody rozhraní mají na vstupu také základní parametry, které určují kolik se má doporučit položek a jsou-li k doporučení předem vybraní nějací kandidáti. Kandidáti určují množinu položek, ze které se skládá doporučení. Pokud není skupina kandidátu určena, doporučení se složí ze všech položek v datové sadě.

Doporučovací modely implementují metody tohoto rozhraní a pomocí natrénovaného stavu vyvářejí doporučení pro cílový subjekt. Nastane-li situace, že model není schopen vytvořit doporučení pro konkrétní cílový subjekt, je seznam doporučených položek prázdný a struktura uchovává informaci o tom, který z cílových subjektů je pro daný model neznámí. Neznámým subjektům je následně možné vytvořit doporučení pomocí jiného rozhraní nebo modelů. Metody rozhraní pro doporučení jsou následující.

- **Items2User**, doporučení položek uživateli, jenž je cílovým subjektem. Nepovinným vstupním parametrem je aktuální hodnocení položek uživatelem. Model implementující metodu tak může reagovat na aktuální stav uživatelské preference.

- **Items2Item**, doporučení položek k vybrané položce, která je cílovým subjektem. Tato metoda vrací nejrelevantnější substituty vybrané položky.

- **Items2Cart**, doporučení položek k vybrané skupině položek, například z nákupního košíku, jenž je cílovým subjektem. Metoda vrací nejrelevantnější komplementy k vybrané skupině.

- **NonPersonalized** je metoda, která jako jediná nemá cílový subjekt a je výjimkou v definovaném rozhraní. Metoda slouží jako řešení v situacích, kdy chceme doporučit položky ale cílový subjekt je neznámí. Příkladem může být situace, kdy chceme doporučit položky nově registrovanému uživateli.

Metoda **Items2User** je přizpůsobena k vytvoření doporučení pro skupinu uživatelů. Tím je myšleno, že doporučení je vytvořeno pro každého uživatele ze skupiny zvlášť, avšak náročná část výpočtu relevancí položek se provede současně pro všechny cílové subjekty. To stejné platí i pro metodu **Items2Item**.

[5] `https://scipy.org/`

3.4.3 Doporučovací modely

Doporučovací modely jsou implementace konkrétních doporučovacích algoritmů a metod popsaných v kapitole 1. Model je třída, jehož objekty trénují svůj stav na datové sadě za účelem vytvoření doporučení. Každý model je potomkem abstraktní třídy **ModelBase**, která definuje jeho základní metody a atributy.

Objekt modelu se inicializuje sadou parametrů, které každý model definuje. Tyto parametry můžou ovlivňovat proces trénování nebo vytváření doporučení a v oblasti strojového učení se jim říká hyper-parametry. Laděním hodnot hyper-parametrů můžeme zvýšit výkonost doporučování. Objekt modelu lze také inicializovat statickou metodou **fromState** a předáním dříve natrénovaného stavu modelu.

Modely trénují svůj stav voláním metody **train**, která má na vstupu datovou sadu projektu. Stavem modelu můžou být jakékoliv atributy modelu, jejichž hodnoty se mění v závislosti na datové sadě, a pomocí nichž je model schopen vytvořit seznam doporučených položek pro určitý cílový subjekt doporučení. Stav modelu lze získat voláním metody **getState**.

Natrénované modely můžeme použít pro vytvoření doporučení. K tomu je zapotřebí, aby implementovali jednu nebo více metod z definovaného rozhraní doporučování 3.4.2. Hierarchii tříd modelů lze vidět v diagramu 3.5. My si teď krátce popíšeme všechny modely, uvedeme co je jejich trénovaným stavem, jaké jsem využili knihovny pro jejich implementaci a zmíníme se i o základních hyper-parametrech. Seznam všech hyper-parametrů modelů a jejich podrobnější popis lze najít ve vývojové dokumentaci doporučovací aplikace.

- **MatrixFactorization**: model, který implementuje metodu faktorizace matic 1.1.1. Model využívá implementace z knihovny Implicit[6], která nabízí dva typy faktorizace matic odlišující se použitou ztrátovou funkcí **ALS** a **BPR**. Vstupními daty algoritmu jsou kolekce hodnocení ve formátu řídké matice, se kterou dokáže knihovna efektivně pracovat. Trénovaným stavem jsou latentní vektory uživatelů a položek. Model faktorizace matic implementuje rozhraní **Items2User** k doporučování položek uživatelům pomocí skalárního součinu daných latentních vektorů. Základními hyper-parametry modelu jsou velikost dimenze latentních vektorů, regularizační faktor použitý při trénování a počet trénovacích iterací.

- **ELSA**: model, který implementuje algoritmus lineárního autoencoderu Elsa 1.1.1. Model využívá implementaci z knihovny Elsarec[7]. Vstupními daty algoritmu je kolekce hodnocení ve formátu řídké matice. Trénovaným stavem jsou vektorové reprezentace doporučovaných položek s pevnou velikostí dimenze. Model implementuje rozhraní **Items2Cart** a **Items2Item** a pro doporučení položek využívá kosinové podobnosti vektorových reprezentací, přičemž doporučuje nejpodobnější položky k cílovému subjektu. Model také implementuje rozhraní **Items2User** k doporučení položek uživateli. Vektorové reprezentace se zkombinují s hodnocením položek uživatele, za účelem odhadu uživatelských preferencí. Základními hyper-parametry modelu jsou velikost dimenze vektorové reprezentace položek a počet trénovacích iterací.

[6] `https://benfred.github.io/implicit/`
[7] `https://github.com/recombee/ELSA`

- **AssociationRulesMining**: model implementující algoritmus pro nalezení asociativních vazeb mezi položkami na základě techniky **Market Basket Analysis (MBA)** (Hossain a kol., 2019). Model využívá implementaci algoritmu z knihovny MLxtend[8]. Vstupními daty algoritmu je kolekce relací položek. Algoritmus hledá vazby mezi relacemi a jejich podmnožinami, odvozuje pravidla a určuje jejich sílu. Tyto pravidla jsou trénovaným stavem modelu. Model implementuje metodu **Items2Cart** k doporučování položek do nákupního košíku. Model se snaží najít položku, která je součástí pravidla s co nejvíce položkami z košíku a zároveň, aby toto pravidlo mělo co největší sílu.

- **BertBased**: model zastupující doporučování založené na obsahu. Trénovaným stavem modelu jsou vektorové reprezentace doporučovaných položek. Tyto reprezentace jsou získány z vlastností položek z datové sady, konkrétně z jejich textových popisů. Textové popisy jsou transformovány do číselných vektorů pomocí [BERT] modelu, který je již natrénovaný, aby dokázal co nejpřesněji reprezentovat jejich kontext. Natrénovaný BERT model získáváme z repositáře knihovny Sentence Transformers[9]. Kosinovou podobností nad vektorovými reprezentacemi určujeme relevanci položek k uživatelským profilům nebo k vybraným položkám, pro které chceme vytvořit doporučení.

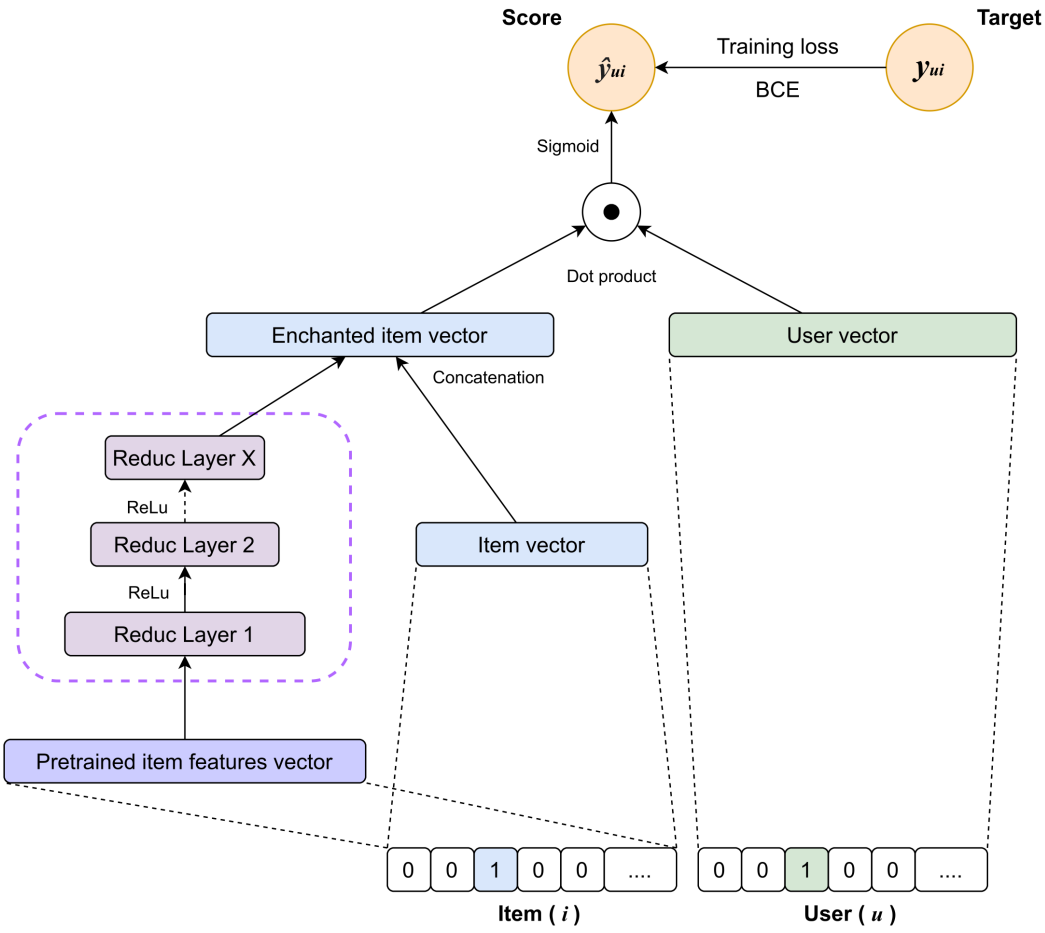
- **Monolith**: model, který reprezentují hybridní přístup feature-combination 1.1.3 .Jedná se o implementaci upraveného algoritmu faktorizace matic, jenž kombinuje trénované latentní vektory položek s již natrénovanými vektorovými reprezentacemi. Těmito vektorovými reprezentacemi myslíme především natrénovaný stav **BertBased** 3.4.3 modelu, který je popsaný výše. Schéma modelu můžeme vidět na obrázku 3.4. Model využívá skryté vrstvy pro redukci vektorových reprezentací BertBased modelu. Model tak ve výsledku kombinuje CF a CB doporučovací techniku a poskytuje hybridní doporučení uživatelům. Trénovaným stavem modelu jsou stejně jako u klasické faktorizace matic latentní vektory uživatelů a položek a výsledné doporučení je vytvořeno jejich skalárním součinem. Základními hyper-parametry modelu jsou velikost dimenze trénovaných latentních vektorů a počet trénovacích iterací.

- **PopularityBased**: model, který implementuje rozhraní **NonPersonalized** pro doporučování položek na základě jejich popularity. Popularita položky je určena podílem počtu unikátních uživatelů hodnotících položku a celkovým počtem uživatelů v datové sadě. Vypočítané hodnoty položek jsou trénovaným stavem modelu a vstupními daty trénování je kolekce hodnocení položek. Hyper-parametrem modelu je časové období hodnocení, ze kterého se počítá popularita položek.

3.4.4 Hybridní přístupy

O hybridním doporučování jsme již se zmínili při popisu **Monolith** modelu, který kombinuje principy doporučování během procesu trénování. Nyní si popí-

[8] `https://rasbt.github.io/mlxtend/`
[9] `https://www.sbert.net/`



Obrázek 3.4: Schéma monolitického modelu.

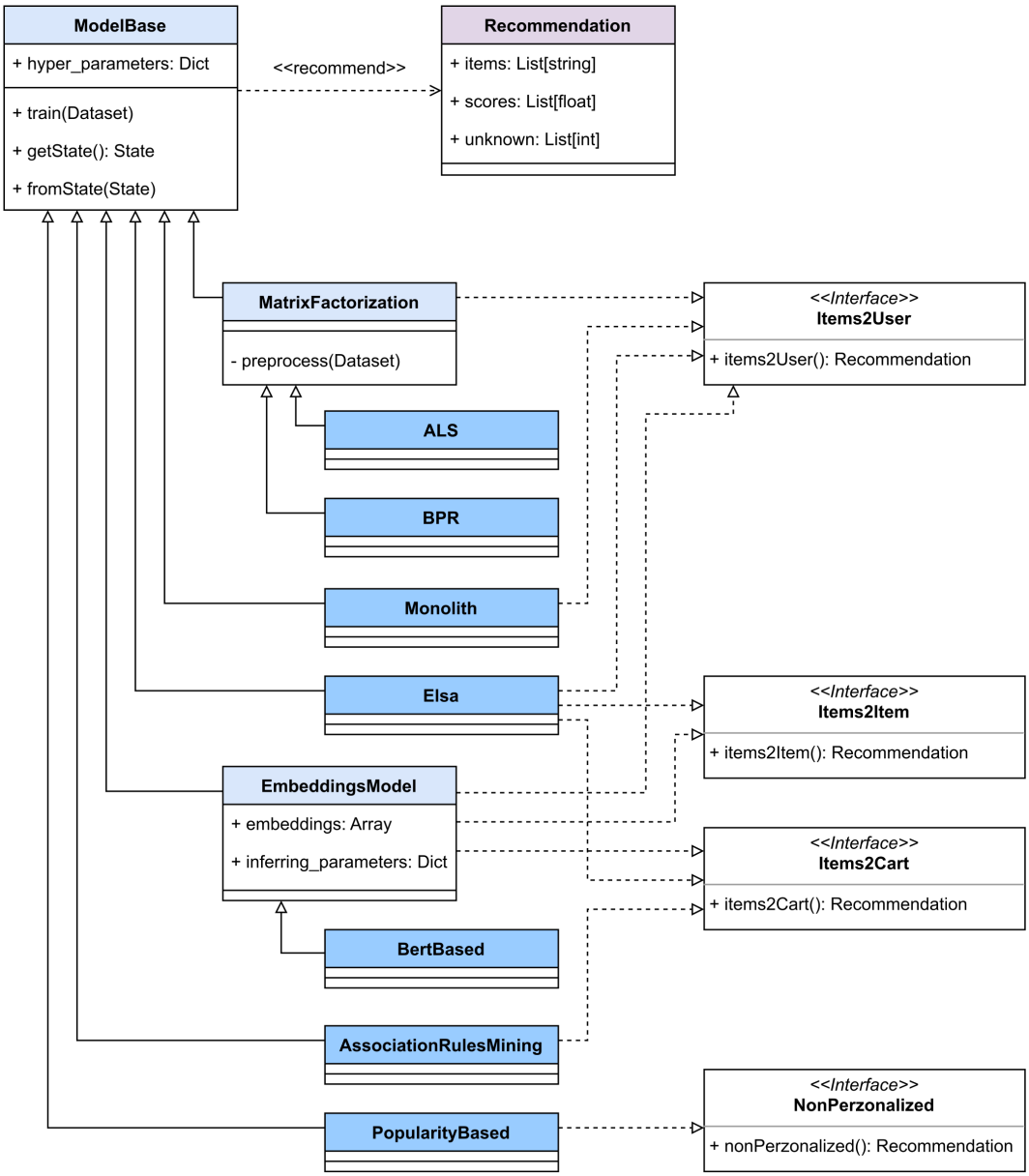
šeme implementace dalších hybridních přístupů, které vytváří výsledné doporučení kombinací dílčích doporučení různých modelů.

Hybridní přístupy jsou v modulu reprezentované třídami, které stejně jako modely implementují metody z rozhraní 3.4.2. Objekt hybridního přístupu se inicializuje seznamem natrénovaných modelů, které implementují stejné metody doporučovacího rozhraní. V této práci jsme se omezili pouze na metodu **Items2User** ale podobně bychom mohli implementovat i **Items2Item**. Každý přístup kombinuje nejméně dva modely a definuje sadu parametrů, které upřesňují způsob jejich kombinace. Tyto parametry se udávají také během inicializace. Hybridní přístupy implementované v aplikaci jsou následující.

- **Weighted**: hybridní přístup, který kombinuje dílčí doporučení jednotlivých modelů například pomocí techniky BordaCount[10]. Parametry tohoto přístupu jsou explorace a způsob agregace hodnot doporučených položek. Explorace určuje počet položek, které každý z dílčích modelů doporučí navíc oproti stanovenému počtu doporučení. Explorace napomáhá upřesnit výsledné doporučení během agregace.

- **Cascade**: hybridní přístup kombinující právě dva modely. Pomocí prvního modelu se vytvoří hrubé doporučení, které je následně uspořádané druhým

[10] `https://en.wikipedia.org/wiki/Borda_count`



Obrázek 3.5: Diagram tříd modelů a implementace rozhraní doporučování

modelem. První model tedy doporučí kandidáty, ze kterých druhý model vytvoří výsledné doporučení. Parametrem toho přístupu je opět explorace, která se vztahuje pouze na první z modelů. Pořadí modelů je určené pořadím v seznamu modelů během inicializace přístupu.

- **Switching**: hybridní přístup, který na základě předané distribuce vybírá model, jenž vytvoří výsledné doporučení. Distribuce výběru modelů pro doporučení je parametrem tohoto přístupu. Pokud není distribuce uvedena, všechny modely mají stejnou pravděpodobnost výběru.

3.5 Vrstva přístupu k datům

Vrstva přístupů k datům je zodpovědná přenos dat mezi aplikací a úložišti doporučovacího systému. Doporučovací systém má dvě úložiště, když nepočítáme databázi, o kterou se stará dedikovaná vrstva. Prvním je datové úložiště, které ukládá data projektů a druhým je registr, který ukládá natrénované stavy doporučovacích modelů. Doporučovací systém má ještě jeden speciální typ datových úložišť, kterými jsou zdroje datových sad projektů. Každý projekt registruje do systému svůj zdroj datové sady, z kterého systém datovou sadu načítá. Zdroj datové sady projektu se tak stává, během načítání dat, součástí systému.

3.5.1 Datové úložiště

Datové úložiště slouží v doporučovacím systému pro ukládání dat projektů. Projekt poskytuje systému svá data, které se zpracovávají do struktury datové sady 3.4.1. Datová sada se následně uloží do datového úložiště, odkud jí systém může libovolně načítat a pracovat s ní. Systém do datového úložiště ukládá také záznamy o doporučení, které obsahují informace o položkách datové sady projektu. Záznamy o doporučení se ukládají za účelem rekonstrukce a vyhodnoceni daného doporučení. Záznamy doporučení, podobně jako u datové sady, reprezentujeme kolekcí, jenž seskupuje dokumenty.

Vrstva vytváří abstrakci, která odstiňuje technologie ukládání dat v datovém úložišti. Definuje abstraktní třídu **DataStorage**, reprezentující datové úložiště, a abstraktní třídy, reprezentující různé kolekce, jenž definují základní metody pro manipulaci s daty. Schéma abstraktních tříd můžeme vidět na obrázku 3.6. Vrstva následně také implementuje třídy, které reprezentují kolekce datové sady pro konkrétní realizaci datového úložiště.

Datové úložiště je v systému realizované pomocí Mongo[11] databáze. Každý projekt má svoji vlastní dedikovanou databázi za účelem snazší správy.

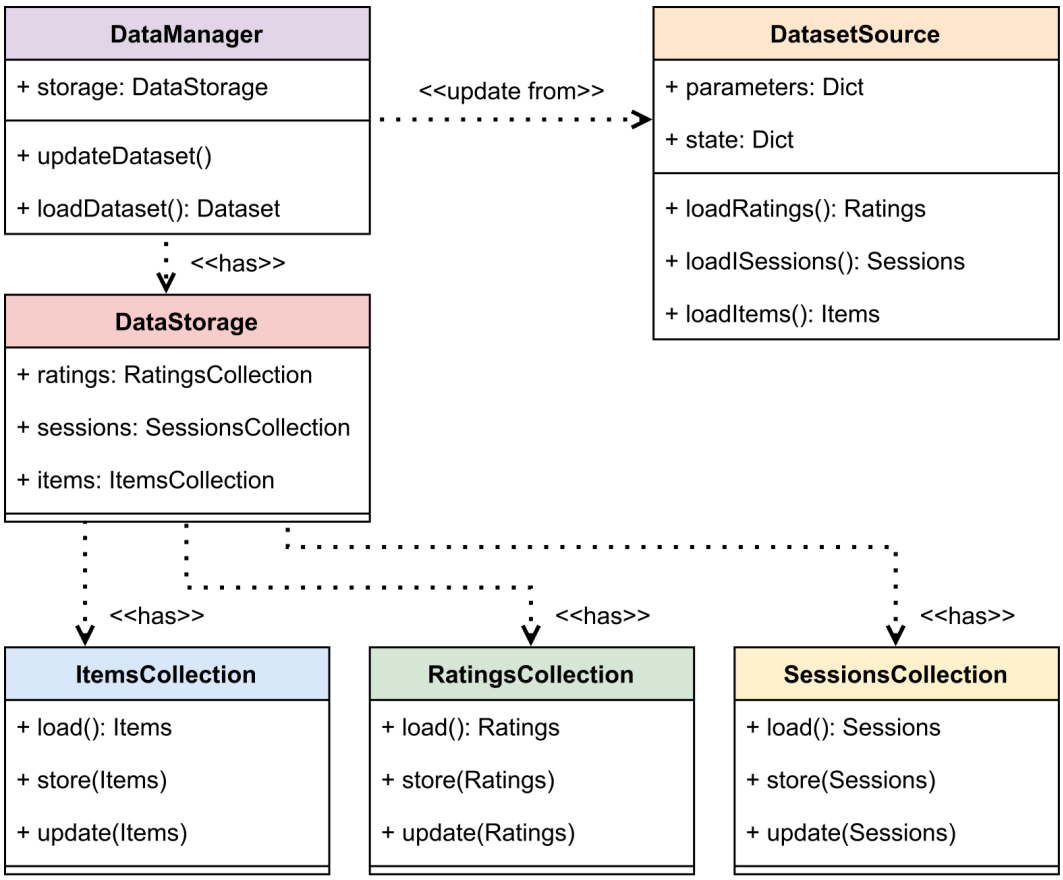
3.5.2 Registr modelů

Registr je úložiště doporučovacího systému, které ukládá natrénované stavy doporučovacích modelů. Stav modelu nemá pevnou strukturu a jedná se spíše o slovník přiřazující hodnoty atributů modelu k textovým klíčům. Vrstva přístupu k datům definuje datovou strukturu **ModelState**, které reprezentuje stav modelu. Tato struktura má dva atributy, jejichž hodnoty obsahují dříve zmíněný slovník a data spojena s trénovacím procesem modelu. Například čas strávený trénováním. Vrstva dále definuje abstraktní třídu **ModelsRegister**, která definuje metody pro ukládání, načítání a mazání zmíněné datové struktury z registru.

Registr modelů je v doporučovacím systému realizovaný adresářem souborového systému. Registr vytváří v daném adresáři vlastní adresářovou strukturu, kam ukládá serializované datové struktury stavů modelů. Aplikace využívá pro serializaci objektů knihovnu pickle [12], která převádí objekty Pythonu do sekvence bitů, jenž se ukládají jako binární soubor do zvoleného adresáře. Dále je využita

[11] `https://www.mongodb.com/`

[12] `https://docs.python.org/3/library/pickle.html`



Obrázek 3.6: Schéma implementace datového úložiště

i knihovna lzma[13] pro kompresi dat pomocí stejnojmenného algoritmu.

3.5.3 Zdroj datové sady

Zdroj datové sady označuje úložiště konkrétního projektu, ve kterém jsou uložena jeho data. Každý projekt registruje v systému svůj zdroj datové sady, ze kterého systém data projektu načítá. Vrstva přístupu k datům reprezentuje zdroj datové sady pomocí abstraktní třídy **DatasetSource**, která definuje základní metody pro načítání dat a jejich transformaci na kolekce datové sady. Každý zdroj datové sady, implementovaný ve vrstvě, je potomkem této třídy a definuje sadu parametrů, které ho inicializují. Implementované zdroje datových sad jsou následující.

- **GoodBooks**: Zdroj datové sady knížek z repositáře GoodBooks[14]. Tento zdroj nemá žádné inicializační parametry a slouží jako příklad.
- **Generic**: Obecný zdroj, který načítá kolekce datové sady ze souborových adresářů definovaných pomocí jeho parametrů. Parametry zdroje určují cestu a datový typ souborů obsahujících jednotlivé kolekce. Tyto kolekce musejí mít strukturu odpovídající 3.4.1. Jednotlivé názvy atributů se dají nastavit pomocí parametrů.

[13] `https://docs.python.org/3/library/lzma.html`
[14] `https://github.com/malcolmosh/goodbooks-10k-extended`

3.6 Databáze

Databáze v systému slouží k zajištění persistence dat doporučovací aplikace. Schéma databáze se řídí datovým modelem, který definuje vrstva databáze v aplikaci. Tady pro jistotu jen uvedeme, že se nejedná o doporučovací model. Datový model definuje entity, které databáze ukládá, a jaké vztahy mezi sebou mají. Pro snaží práci s databází používáme nástroj Object Relational Mapping (ORM)[15], který mapuje záznamy tabulek na objekty daných tříd. Díky tomu můžeme s daty manipulovat čistě pomocí objektově orientovaného návrhu, jenž vrstva databáze implementuje. Pro tyto účely využíváme v aplikaci knihovnu SqlAlchemy[16]. Ta zjednodušuje vytváření dotazů pro manipulaci s daty určité tabulky, na volání metod třídy, která tuto tabulku reprezentuje.

Vrstva databáze implementuje návrhový vzor repositářů, který vytváří abstrakci pro manipulaci s entitami datového modelu. Repositář je třída konkrétní entity, která definuje metody pro CRUD operace. Například repositář projektů, který má metody pro získání projektu podle jména nebo vytvoření nového projektu. Databáze v systému je realizovaná pomocí PostgreSQL[17]. Doporučovací aplikace používá adaptér psycopg2[18], který implementuje rozhraní PostgreSQL databáze pro Python.

3.6.1 Datový model databáze

Datový model definuje entity a vztahy mezi nimi, které databáze ukládá. Jinak řečeno, popisuje konceptuálně datový prostor doporučovací aplikace. Ještě než začneme popisovat jednotlivé entity, uvedeme pár speciálních vlastností datového modelu.

Každá entita datového modelu má číselný identifikátor, který je přiřazován automaticky. Tento identifikátor slouží pro interní identifikaci. Entity, které je potřeba identifikovat mimo doporučovací aplikaci, například v úložištích nebo během komunikace pomocí API, mají pro tento účel textový identifikátor. Hodnota textového identifikátoru je pevně daná nebo určená uživatelem systému.

Některé entity, přímo reprezentují konkrétní implementace v doporučovací aplikaci. Příkladem jsou doporučovací modely, které jsou reprezentovány entitou **Model**. Entita je spojena s konkrétní implementací pomocí atributu cesty v adresářové struktuře aplikace. Cesta odkazuje na třídu, implementující daný doporučovací model. Hodnoty atributů cest jsou omezený pevně definovaným formátem a zároveň implementace, na které odkazují musí být potomky určitých abstraktních tříd v aplikaci. Konkrétně pro implementace doporučovacích modelů třída **ModelBase**.

Entity datového modelu a vztahy mezi nimi můžeme vidět v Entity Relationship (ER) diagramech 3.7 a 3.8. My si teď stručně popíšeme každou z nich.

- **Project**: entita reprezentují externí aplikaci, která využívá doporučovací systém. Projekt má základní atributy název, popis a stav. Stav projektu

[15] `https://en.wikipedia.org/wiki/Object-relational_mapping`
[16] `https://www.sqlalchemy.org/`
[17] `https://www.postgresql.org/`
[18] `https://www.psycopg.org/`

udává zda je projekt aktivní či nikoli. Pokud je projekt neaktivní, nemůže využívat systém. Dalšími atributy projektu jsou textový identifikátor a heslo. Tyto dva atributy slouží pro identifikaci a autentizaci projektu. Každý projekt musí mít uvedený kontext datové sady. To reprezentuje vztah s entitou **DatasetContext**.

- **RecommendationMethod**: entita reprezentující metody rozhraní pro doporučování 3.4.2. Vazbu mezi entitou a konkrétní metodou udává atribut cesty v adresářové struktuře aplikace. Dalšími atributy metody jsou název, popis, textový identifikátor a stav. Stav určuje zda je metoda aktivní. Neaktivní metodu nemůžeme používat pro vytvoření doporučení.

- **Model**: entita reprezentující doporučovací modely 3.4.3. Vazbu mezi entitou a konkrétním modelem udává atribut cesty v adresářové struktuře aplikace. Entita má také podobnou vazbu na schéma hyper-parametrů daného modelu. Dalšími atributy jsou název, popis, textový identifikátor a stav. Stav udává zda je model aktivní a pokud není, tak ho nemůžeme použít k doporučování, a ani nelze natrénovat jeho stav. Doporučovací model implementuje metody z rozhraní pro doporučování. To reprezentuje vztah s entitou **RecommendationMethod**.

- **TrainingContext**: entita reprezentující trénovací kontext. Trénovací kontext patří konkrétnímu projektu a udává, jaký model se má trénovat. To reprezentují vztahy s entitami **Project** a **Model**. Dalšími atributy kontextu jsou seznam hyper-parametrů, použitých pro trénování, a časový interval, kdy se má trénovací proces pro daný model opakovat. Projekt může mít pro každý model pouze jeden trénovací kontext. Kontext trénování slouží pro uložení nastavení trénovacího procesu konkrétního modelu a zajištění automatizace jeho trénování.

- **ModelArtifact**: entita reprezentující doporučovací modely, které mají natrénovaný svůj stav. Jinak řečeno, entita reprezentuje různé verze doporučovacích modelů po skončení jejich procesu trénování. Artefakt je termín strojového učení, který se používá k popisu výstupu vytvořeného v procesu učení. V našem případě se jedná o natrénovaný stav modelu a metadata spojená s trénovacím procesem. Artefakty jsou často spojovány se správou životního cyklu strojového učení, jak popisují (Schlegel a Sattler, 2023). Artefakt modelu patří konkrétnímu projektu, který zahájil jeho proces trénování a zároveň je spojen s doporučovacím modelem, jehož natrénovaný stav reprezentuje. Toto udávají vztahy s entitami **Project** a **Model**. Každému artefaktu je vygenerována unikátní značka, která slouží pro jeho externí identifikaci. Dalšími atributy artefaktu jsou metadata trénovacího procesu modelu, časová známka zahájení trénovacího procesu a status. Metadata obsahují celkový čas strávený trénováním a sadu hyper-parametrů použitých pro inicializaci trénovaného modelu. Status artefaktu může mít jednu z hodnot trénování, připraven a smazán. Status trénování značí, že stav modelu se právě trénuje a není možné ho použít pro vytvoření doporučení. Status připraven označuje artefakt, který ukončil proces trénování a můžeme ho použít k doporučování. Status smazán mají artefakty, které již

není možné použít pro doporučování z důvodu odstranění natrénovaného stavu z registru modelů.

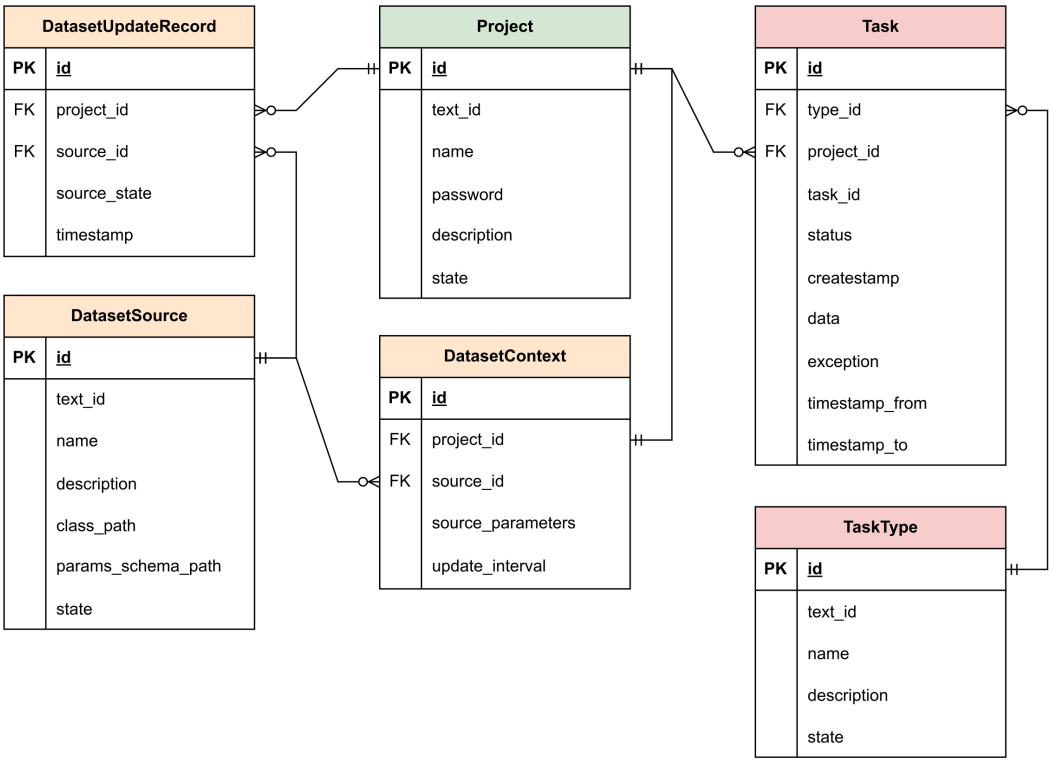
- **DatasetSource**: entita reprezentující zdroje datových sad 3.5.3. Vazbu mezi entitou a konkrétním zdrojem datové sady udává atribut cesty v adresářové struktuře aplikace. Entita má také podobnou vazbu na schéma parametrů daného zdroje. Dalšími atributy jsou název, popis, textový identifikátor a stav. Stav udává zda je zdroj datové sady aktivní. Z neaktivního zdroje nelze načítat datovou sadu projektu.

- **DatasetContext**: entita reprezentující kontext datové sady. Každý projekt musí mít právě jeden kontext datové sady, který udává, z kterého zdroje se má načítat jeho datová sada. Tyto vazby reprezentují vztahy s entitami **Project** a **DatasetSource**. Dalšími atributy kontextu jsou seznam parametrů, použitých při načítání datové sady ze zdroje, a časový interval, kdy se má proces aktualizace datové sady opakovat.

- **DatasetUpdateRecord**: entita reprezentující záznamy aktualizací datové sady. Záznamy vznikají po skončení procesu aktualizace datové sady ze zdroje. Každý záznam je spojen s konkrétním projektem, jehož datová sada byla aktualizována, a se zdrojem odkud byla získána. To reprezentují vztahy s entitami **Project** a **DatasetSource**. Entita má atribut udávající stav zdroje datové sady a využívá pro zajištění jeho persistence po skončení procesu aktualizace. Stav například může obsahovat datum posledního záznamu hodnocení a při další aktualizaci datové sady, bude zdroj načítat pouze nové záznamy. Dalším atributem je časová známka udávající, kdy skončil proces aktualizace.

- **HybridApproach**: entita reprezentující hybridní přístupy 3.4.4. Vazbu mezi entitou a konkrétním přístupem udává atribut cesty v adresářové struktuře aplikace. Entita má také podobnou vazbu na schéma parametrů daného hybridního přístupu. Dalšími atributy jsou název, popis, textový identifikátor a stav. Stav udává zda je hybridní přístup aktivní. Neaktivní hybridní přístup nemůžeme použít pro vytvoření doporučení.

- **RecommendationRecord**: entita reprezentující záznamy o doporučení. Záznamy patří k projektu, jehož položky byly doporučovány, a obsahují informace spojené s jeho vytvořením. Záznam obsahuje identifikátor doporučení, jenž reprezentuje, a je spojen s doporučovací metodou a konkrétním artefaktem modelu nebo s hybridním přístupem v případě hybridního doporučení. Dalšími atributy záznamu jsou časová známka, kdy bylo doporučení vytvořeno, a identifikátor skupiny. Ten se používá jako označení záznamů o doporučení pro danou skupinu. Záznamy se ukládají za účelem rekonstrukce a vyhodnocení poskytnutého doporučení. Můžeme tedy zpětně vyhodnotit výkonost doporučování konkretního modelu, jeho artefaktu nebo již zmíněné skupiny. Záznamy neobsahují konkrétní data z datové sady projektu. Proto jsou podrobné informace o položkách a vstupních argumentech doporučení uloženy v datovém úložišti systému pod uvedeným identifikátorem doporučení.

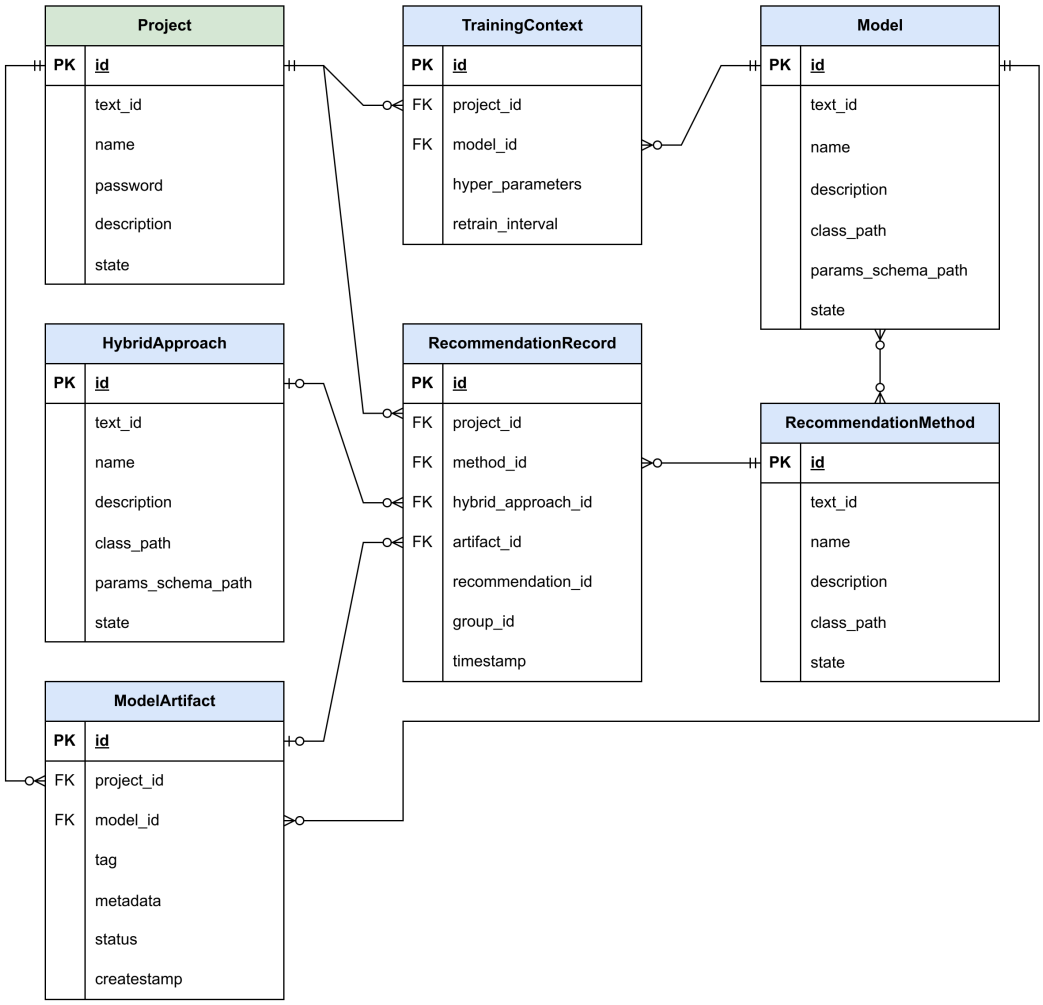


Obrázek 3.7: ER diagram databáze 1.část

- **Task**: entita reprezentující úlohy, které se zpracovávají na pozadí. V systém tyto úlohy reprezentují dlouhotrvající procesy, které se zpracovávají vedlejšími běhy doporučovací aplikace. Konkrétně se jedná o procesy trénování doporučovacího modelu a aktualizace datové sady projektu ze zdroje. Tyto konkrétní procesy určují typ úlohy a jsou reprezentovány vlastní entitou **TaskType**. Každá úloha je spojena s projektem, který ji vytvořil, a jejím typem. Entita má textový identifikátor a atribut data, který udává vstupní argumenty procesu, jenž úloha reprezentuje. Dále má entita časové známky, které udávají, kdy byla úloha vytvořena, kdy se začala zpracovávat a kdy byl proces zpracování úlohy ukončen. Dalšími atributy jsou výjimka, v případě že zpracování úlohy skončilo neúspěchem, a status úlohy. Status úlohy může mít jednu z hodnot čekání na zpracování, zpracovávání, úspěch a neúspěch.
- **TaskType**: entita reprezentující typ úlohy. Doporučovací aplikace definuje dva typy úloh, kterými jsou trénování doporučovacího modelu a aktualizace datové sady ze zdroje projektu. Entita má základní atributy název, popis, textový identifikátor a stav. Stav udává zda je úloha aktivní. Neaktivní úlohu nejde vytvořit ani zpracovat vedlejšími běhy aplikace.

3.6.2 Migrace schémat

Migrace se využívají pro řízení a správu verzí schématu relační databáze. Konkrétní migrace reprezentuje změnu, zapsanou jako posloupnost jednotlivých operací upravující schéma databáze. Migrace uvádí operace, které vedou k výsledné úpravě, a také operace, které tuto úpravu vracejí do původního stavu. Migrace



Obrázek 3.8: ER diagram databáze 2.část

má také odkaz na předchozí a následující úpravy. Jejich kolekce tedy tvoří oboustranně spojitý seznam, jenž se prochází v obou směrech za účelem aktualizace nebo vrácení předchozích změn schématu databáze. Jednotlivé migrace můžeme chápat jako verze a databáze si ukládá konkrétní verzi, ve které se právě nachází.

Díky migracím můžeme schéma databáze verzovat společně se změnami v doporučovací aplikaci. Migrace umožňují krom změny schématu i manipulaci s daty databáze. V systému migrace používáme pro iniciální vytvoření schématu databáze, jeho úpravu a vložení iniciálních dat. Pro vytvoření migrací a jejich spouštění využíváme knihovnu Alembic[19], která umí automaticky generovat migrace z implementovaného datového modelu pomocí ORM.

3.7 Background modul

V doporučovací aplikaci se setkáváme s úlohami, které jsou časově náročné na zpracování. Takové úlohy bychom chtěli zpracovávat asynchronně mimo hlavní proces aplikace, aby nedocházelo k jeho blokování. Background modul implementuje distribuovanou frontu úloh, která umožňuje přenést náročnou práci na

[19] `https://alembic.sqlalchemy.org/`

jiný proces a zpracovávat úlohy asynchronně a paralelně s ostatními. Fronta nám tedy dává možnost zpracovávat náročné úlohy na pozadí, zatímco hlavní proces aplikace pokračuje ve zpracování ostatních úloh.

Modul definuje dva typy úloh, které je možné zpracovávat pomocí fronty. Prvním je trénování doporučovacího modelu a druhým je aktualizace datové sady projektu. Úlohy mají unikátní název, který je identifikuje ve frontě, a definují sadu vstupních parametrů. Úloha trénování modelu přijímá parametry identifikátor modelu a sadu jeho hyper-parametrů. Úloha aktualizace datové sady přijímá identifikátor zdroje datové sady a jeho parametry.

Modul implementuje úlohy pomocí knihovny Celery[20], která se stará o jejich zpracování. Diagram zpracování úloh lze vidět na obrázku 3.9 a můžeme ho logicky rozdělit na dvě části. V první části se úloha vytvoří a zařadí do fronty, a ve druhé části je zpracována vedlejšími procesy. Přesný čas začátku a délky zpracování úloh není známý a záleží na zátěži vedlejší procesů.

Background modul na základě příchozích požadavků vytváří úlohy, které jsou reprezentovaný tak zvanými zprávami. Zpráva úlohy obsahuje její název, vygenerovaný unikátní identifikátor a hodnoty vstupních parametrů. Tyto zprávy se zasílají zprostředkovateli zpráv, který je zařadí do příslušných front. Zprostředkovatel může mít obecně více front s různými prioritami pro zpracování. Po zařazení úloh do front, modul vrací jejich identifikátory jako odpovědi na příchozí požadavky. Fronty zprostředkovatele jsou sledovány spotřebiteli, kteří reprezentují vedlejšími procesy aplikace a zpracovávají příchozí úlohy.

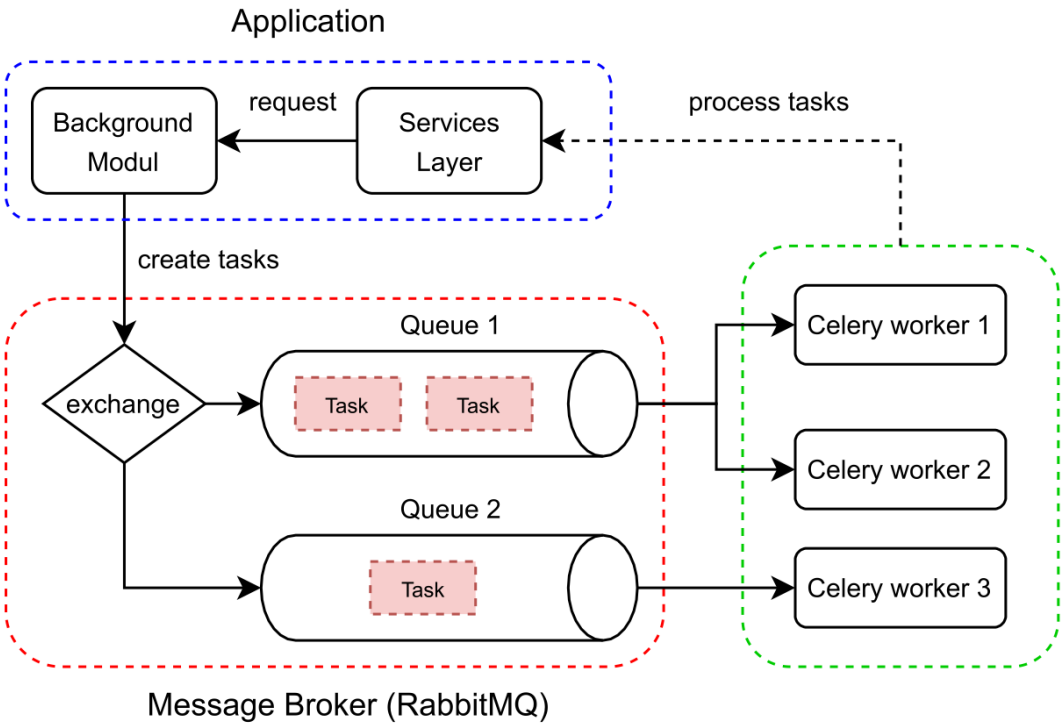
Během zpracování konkrétní úlohy je sledován její stav, který se ukládá do databáze systému. Společně s úlohou se vytvoří záznam, který obsahuje podobné informace jako zpráva, jenž se zasílá zprostředkovateli. Záznam obsahuje vstupní argumenty úlohy, její unikátní identifikátor a název, který určuje typ úlohy. Záznamu je přiřazen status čekání, který udává, že úloha je zařazena do fronty a čeká na zpracování. Poté co spotřebitel začne úlohu zpracovávat, záznamu se změní status na zpracovávání. Když je proces zpracování úlohy ukončen, status záznamu úlohy v databázi je změněn na úspěšný nebo neúspěšný podle výsledku zpracování. Společně se statusem úlohy se zaznamenávají časové známky, kdy byla úloha vytvořena, kdy se začala zpracovávat a kdy byl proces zpracování úlohy dokončen.

3.8 Servisní vrstva

Servisní vrstva poskytuje ucelené funkcionality systému modulům implementující komunikační rozhraní. Například poskytuje metody pro vytvoření Json Web Token (JWT)[21], jenž využívá Api modul k ověření a autorizaci projektu během komunikace. Vrstva dále implementuje správce pro přístup do datových úložišť nebo procesy pro trénování doporučovacích modelů a vytvoření výsledného doporučení. Vrstva zapouzdří funkce dříve zmíněných komponent a vytváří z nich ucelené procesy, které tyto funkce kombinují k dosažení určitých cílů. V této sekci si popíšeme dva nejdůležitější procesy, kterými jsou právě trénování doporučovacího modelu a vytvoření doporučení pro konkrétního uživatele.

[20] `https://docs.celeryq.dev/`

[21] https://datatracker.ietf.org/doc/html/rfc7519



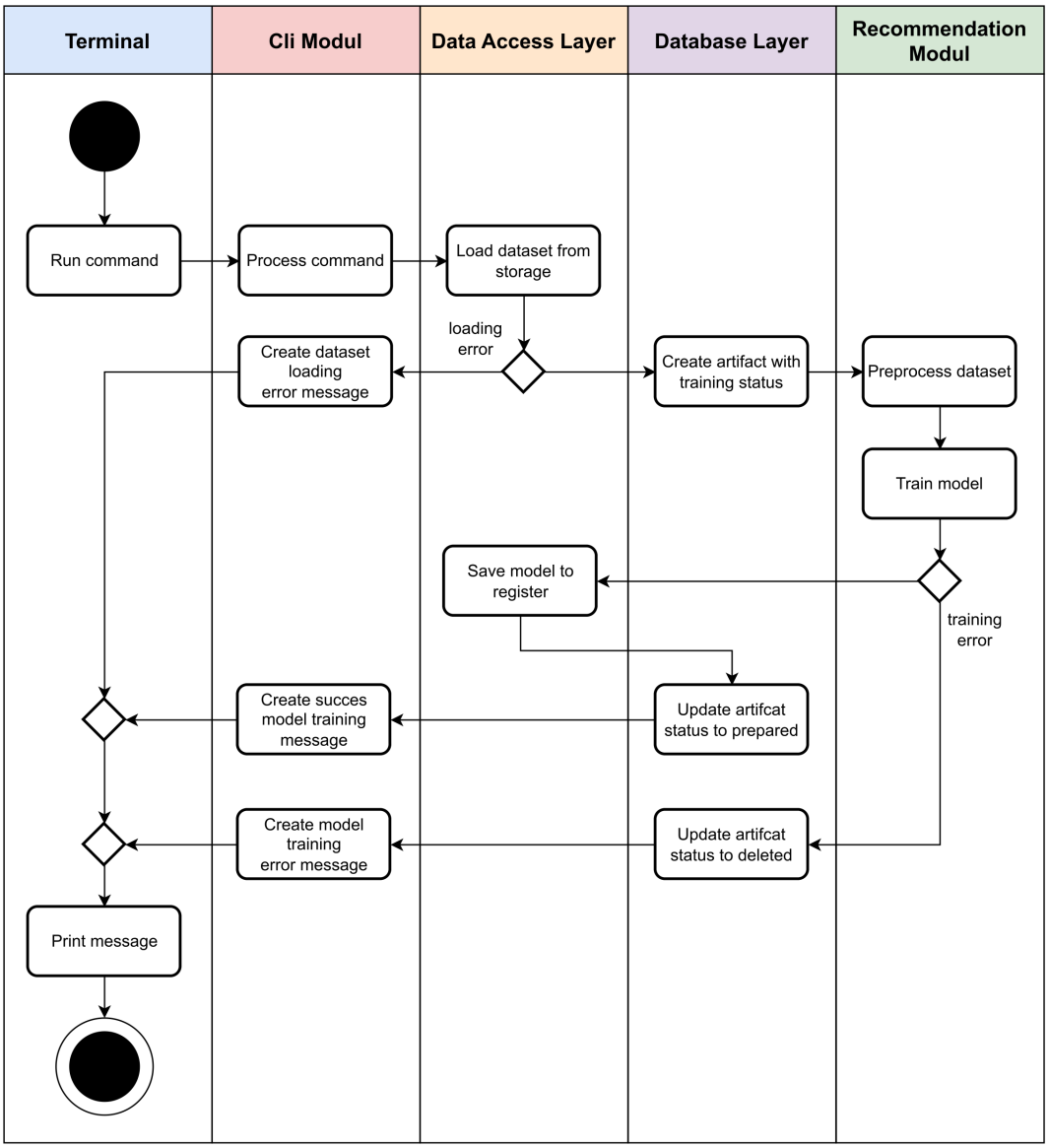
Obrázek 3.9: Diagram zpracování úloh

3.8.1 Proces trénování modelů

Proces trénování modelu si popíšeme z pohledu spuštění příkazu v terminálu. Aktivity diagram procesu můžeme vidět na obrázku 3.10. Diagram a následný popis neobsahují veškeré detaily, kterými jsou například některé kontroly správnosti dat nebo určení konkrétních hodnot parametrů trénování. Proces je rozdělen do několika částí podle komponent, které se na jeho zpracování podílejí.

Proces trénování začíná spuštěním příkazu v terminálu. Příkaz dostane na vstupu argumenty určující projekt a doporučovací model, který se má natrénovat. Příkaz je následně zpracován Cli modulem, jenž validuje vstupní data a inicializuje interní proces trénování modelu. Dalším krokem trénovacího procesu je načítání datové sady určeného projektu z datového úložiště. Tento krok může selhat pokud projekt nemá uloženou datovou sadu v úložišti, nebo pokud nastane problém s jejím načítáním. Nepovede-li se načít datovou sadu, proces končí s chybovou zprávou.

Je-li datová sada načtena, vytvoří se artefakt, který reprezentuje instanci modelu během trénovacího procesu. Artefakt obsahuje informace o trénovaném modelu a projektu na jehož datové sadě se trénuje. Status artefaktu má v tomto okamžiku nastavenou hodnotu trénování. Před samotným trénováním, se datová sada ještě upraví do formátu, který vyhovuje danému trénovanému modelu. Trénování následně probíhá podle konkrétního algoritmu, jenž doporučovací model integruje. Je-li trénování modelu úspěšně dokončené, stav modelu se uloží do registru modelů a status vytvořeného artefaktu se změní na připraven. Model je možné od této chvíle použít k vytvoření doporučení. Nastane-li při trénování chyba, artefaktu je přiřazen status smazán a proces skončí s chybovou zprávou.



Obrázek 3.10: Aktivity diagram trénovacího procesu

Artefakt s hodnotou statusu smazán označuje, že model nemá uložený natrénovaný stav v registru modelů a nelze ho tudíž použít k vytvoření doporučení.

Proces trénování můžeme monitorovat pomocí logů, které se vypisují do terminálu, ve kterém jsme spustili daný příkaz. Samotné trénování modelů navíc můžeme analyzovat pomocí knihovny TensorBoard[22]. Většina modelů se trénuje iterativním přístupem. To znamená, že každou trénovací iterací se model snaží vylepšit svůj stav zlepšováním trénovacích metrik. Tyto metriky jsou registrované v TensorBoadu a lze je sledovat přímo během zpracování procesu.

3.8.2 Proces vytvoření doporučení

Proces si popíšeme z pohledu, kdy byl ve webovém rozhraní vytvořen požadavek na vytvoření doporučení pro konkrétního uživatele. Proces má podobný

[22] `https://www.tensorflow.org/tensorboard`

průběh i pro jiný cílový subjekt doporučení. Aktivity diagram procesu můžeme vidět na obrázku 3.11. Diagram a následný popis neobsahují všechny detaily, kterými jsou například některé kontroly správnosti dat nebo určování konkrétních hodnot parametrů doporučování. Proces je rozdělen do několika částí podle komponent, které se na jeho zpracování podílejí.

Proces doporučení položek uživateli začíná vytvořením požadavku ve webové rozhraní. Požadavek obsahuje informace o uživateli, který je cílovým subjektem doporučení, a identifikaci modelu, který má doporučení vytvořit. Identifikace modelu je složena z textového identifikátoru modelu a případné značky konkretního artefaktu modelu. Není-li uveden artefakt, pro vytvoření doporučení se použije artefakt modelu, který byl natrénován nejpozději. Požadavek je zaslán doporučovací aplikaci pomocí HTTP, kde ho Api modul zpracuje. Modul validuje vstupní data a inicializuje vnitřní proces vytvoření doporučení.

Z databáze se vybere požadovaný artefakt natrénovaného modelu, který se následně načte z registru modelů. Pokud artefakt neexistuje, nemá status připraven nebo se nepodaří načíst stav modelu z registru, proces se ukončí a Api modul vytvoří odpověď s adekvátní chybou. Je-li natrénovaný model úspěšně načten z registru, z datového úložiště se načtou aktuální preference daného uživatele. Preference uživatele jsou reprezentovány uživatelským hodnocením položek. Tyto preference jsou následně použity pro vytvoření doporučení natrénovaným modelem.

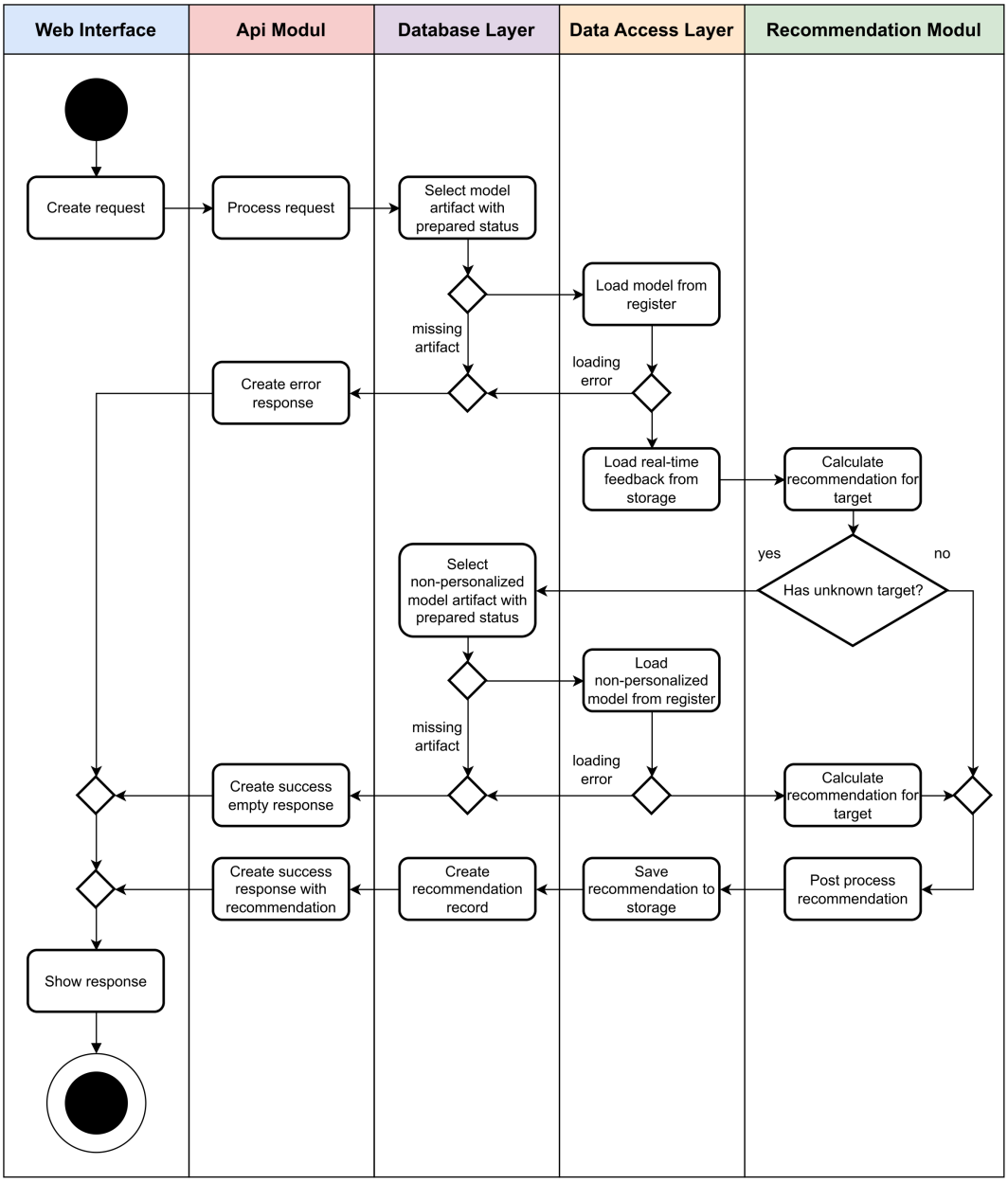
V některých případech se může stát, že model nezná uživatele pro kterého vytváří doporučení. Například v momentě, kdy uživatel neměl žádné preference v době trénování modelu a nemá žádné preference ani během doporučování. V tomto případě se proces pokusí takovému uživateli doporučit položky nepersonalizovaně. To znamená, že se pokusí načíst adekvátní model, který poskytuje takové doporučení. Pokud takový model není natrénovaný, proces končí s úspěšnou odpovědí a prázdným doporučením. V opačném případě se vytvoří doporučení a proces pokračuje v jeho úpravě.

Je-li doporučení vytvořeno a není prázdné, proces toto doporučení upraví do výsledné podoby. Úpravu doporučení je možné vyžádat pomocí konkrétních atributů v požadavku pro doporučení. Úpravou je myšleno například vyfiltrování již hodnocených položek nebo zvýšení vzájemné diverzity doporučených položek na základě jejich obsahové podobnosti.

Poté co je doporučení upravené do výsledné podoby, uloží se doporučené položky s informacemi o cílovém subjektu doporučení do datového úložiště a do databáze se uloží záznam o doporučení obsahující konkrétní model a metodu doporučení. Tyto informace se následně využívají pro vyhodnocení poskytnutého doporučení. Nakonec Api modul vrátí odpověď s výsledným doporučením webovému rozhraní, který odpověď zobrazí.

3.9 Komunikační rozhraní

Doporučovací aplikace poskytuje dvě rozhraní pro komunikaci. Prvním z nich je Command Line Interface (CLI), které je implementováno v modulu Cli. Rozhraní definuje příkazy, které po zavolání vykonávají procedury nebo procesy doporučovací aplikace. Toto rozhraní je využíváno zejména pro zajištění automatizace doporučovacího systému.



Obrázek 3.11: Aktivity diagram vytvoření doporučení

Druhým je Representational State Transfer (REST) aplikační rozhraní, které je implementováno v modulu Api. Rozhraní definuje koncové body, které umožňují externí aplikaci nebo webovému rozhraní přistupovat k prostředkům v rámci doporučovací aplikace.

3.9.1 Cli Modul

Modul, který umožňuje komunikaci s doporučovací aplikací pomocí rozhraní příkazové řádky. Implementuje sadu příkazů, které spouštějí procesy doporučovací aplikace. Příkazy mají argumenty a možnosti, které parametrizují procesy. Argumenty příkazů mají pevný datový typ a jsou povinné. Na příkazové řádce mají určenou pozici. Možnosti příkazů mají také pevný datový typ a jsou vetšinou nepovinné. Na příkazové řádce je uvádíme pomocí pomlček a názvu.

Rozhraní příkazové řádky je implementované pomocí knihovny Typer[23]. Jednotlivé příkazy rozhraní jsou implementované jako potomci třídy **CommandBase**, která definuje jedinou metodu, jenž spouští konkrétní proces. Tyto příkazy jsou inicializovány v metodách, které se volaní přímo z příkazové řádky. Knihovna Typer umožňuje právě toto volání pomocí anotací metod a jejich parametrů.

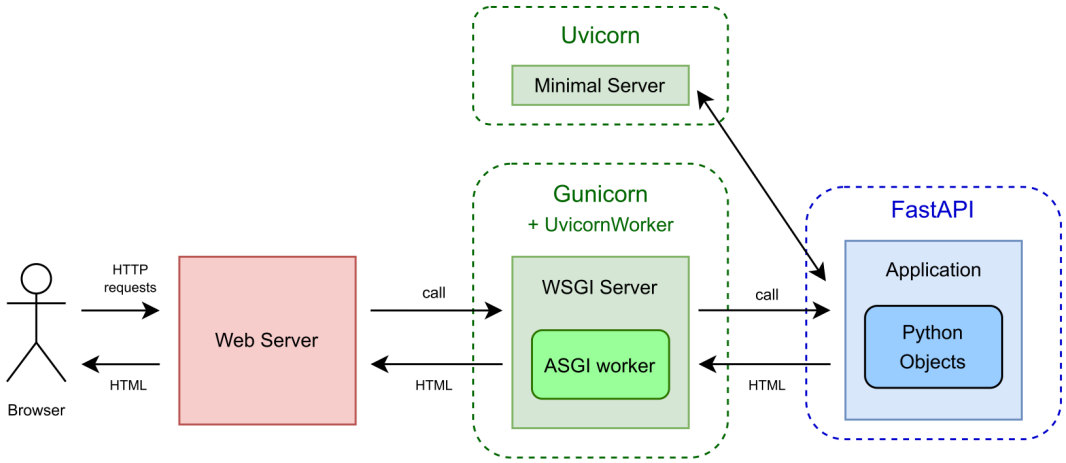
3.9.2 Api Modul

Modul, který implementuje aplikační REST rozhraní. REST je architektonický styl pro návrh webových služeb a API. REST API je způsob komunikace mezi klientem a serverem pomocí HTTP protokolu. Rozhraní definuje koncové body, které umožňují přístup ke zdrojům aplikace. Každý koncový bod je reprezentován unikátní URL adresou. Rozhraní využívá metody HTTP protokolu, jenž říkají, jaké operace se mají provést se zdroji. Metodami jsou GET, POST, PUT a DELETE reprezentující operace získání, vytvoření, úprava a smazání zdroje.

Modul implementuje rozhraní s využitím webového frameworku FastApi[24], jehož hlavními výhodami jsou vysoká rychlost díky podpoře asynchronního zpracování požadavků, automatická dokumentace a snadné použití. Modul pomocí frameworku vytváří objekt webové aplikace. K aplikaci jsou přiřazeny konkrétní koncové body, jenž jsou reprezentovány anotovanými metodami. Každá metoda má definované schémata vstupních a výstupních dat, jenž se vyskytují v těle příchozího požadavku, v URL adrese nebo v odpovědi na požadavek.

Mezi klientem a aplikací jsou webový server a Web Server Gataway Interface (WSGI)[25]. Webový server zpracovává příchozí HTTP požadavky a vrací klientům nejčastěji HTML dokumenty a HTTP odpovědi se stavovými kódy. WSGI definuje jednoduché a univerzální rozhraní mezi webovým serverem a webovou aplikací napsanou v Pythonu. WSGI umožňuje volání konkrétního kódu v Pythonu, jako reakci na příchozí HTTP požadavek. Přímím nástupcem WSGI je poté Asynchronous Server Gateway interface (ASGI)[26], které dokáže zpracovávat příchozí požadavky asynchronně.

[23] `https://typer.tiangolo.com/`
[24] `https://fastapi.tiangolo.com/`
[25] Web Server Gataway Interface - `https://peps.python.org/pep-3333/`
[26] Asynchronous Server Gateway interface - `https://asgi.readthedocs.io/`



Obrázek 3.12: Schéma komunikace klienta s webovou aplikací

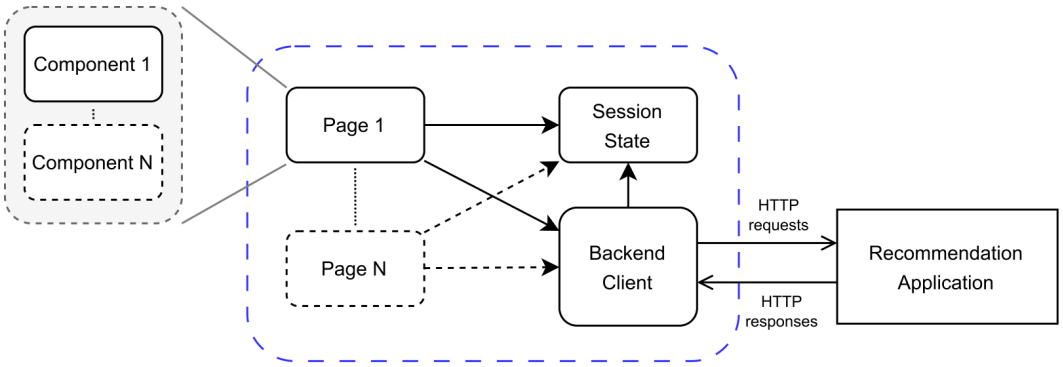
Schéma komunikace klienta (webový prohlížeč) s webovou aplikací napsanou v Pythonu můžeme vidět na obrázku 3.12. Pro vývoj webové aplikace používáme minimalistickou verzi WSGI serveru Uvicorn[27] s jedním procesem zpracovávající příchozí požadavky. Pro produkční verzi aplikace využíváme Gunicorn[28], které spouští více ASGI procesů.

3.10 Webové rozhraní

Webové rozhraní je část doporučovacího systému napsané v Pythonu, která umožňuje uživateli snadnou interakci s doporučovací aplikací. Struktura webového rozhraní se skládá z několika stránek, které jsou složeny z jednotlivých komponent. Komponenty jsou nezávislé části uživatelského rozhraní, které je možné opakovaně používat napříč celým rozhraním. Hlavní částí webového rozhraní je klient, který implementuje volání REST API doporučovací aplikace. Klient vytváří HTTP požadavky a zpracovává příchozí odpovědi. Pro tyto účely se využívá standardní Python knihovna requests[29]. Další částí rozhraní je stav relace webového rozhraní. Stav ukládá data, která jsou přístupná napříč všemi stránkami a komponenty. Schéma struktury webového rozhraní můžeme vidět na obrázku 3.13.

Rozhraní je implementované pomocí knihovny Streamlit[30], která umožňuje snadnou a rychlou implementaci interaktivních webových aplikací. Knihovna obsahuje sadu ovládacích prvků a widgetů, které umožňují interakci s uživatelem. Dále také poskytuje funkce pro efektivní a rychlou vizualizaci dat. Komponenty webového rozhraní jsou konceptuálně Python funkce, které využívají prvky a funkce Streamlit knihovny.

[27] `https://www.uvicorn.org/`
[28] `https://gunicorn.org/`
[29] `https://requests.readthedocs.io/`
[30] `https://streamlit.io/`



Obrázek 3.13: Schéma webového rozhraní

3.10.1 Dynamické generování formulářů

Doporučovací aplikace implementuje sadu doporučovacích modelů a zdrojů datových sad, jenž definují jim specifické parametry, které ve webovém rozhraní umožňujeme uživateli nastavovat. Parametry jsou různých datových typů a platí pro ně různá omezení. Ve webovém rozhraní bychom museli implementovat pro každou sadu parametrů vlastní formulář, který by uživateli umožňoval hodnoty parametrů nastavit a následně je správně validovat. Navíc je možné doporučovací modely a zdroje datových sad libovolně přidávat pomocí principu zásuvných modulů, což by vedlo vždy k potřebné úpravě kódu ve webovém rozhraní.

Doporučovací aplikace poskytuje spolu s informacemi o konkrétním modelu nebo zdroji také datové schéma jeho parametrů. Toto schéma využijeme pro dynamické generování formulářů. Schéma obsahuje informace o parametrech a omezení jejich hodnot. Schéma rekurzivně procházíme a pro každý parametr vytvoříme odpovídající formulářový prvek. Tímto způsobem dokážeme vytvořit formulář, který umožňuje uživateli vložit hodnoty správných datových typů. Parametry navíc můžou být i komplexní, skládající se z několika vnořených parametrů. Během následné validace hodnot použijeme podobný princip. Dynamické generování formulářů řeší problémy s nastavováním hodnot parametrů doporučovacích modelů a zdrojů datových sad popsané výše.

3.11 Zásuvné moduly

Aplikace podporuje mechanizmus zásuvných modulů, které rozšiřují funkcionalitu doporučovacího systému. Konkrétněji je možné do aplikace přidat zásuvné moduly pro doporučovací modely a zdroje datových sad. Zásuvné moduly jsou konceptuálně třídy, které implementují určité rozhraní. V případě zdrojů datových sad třídy dědí od abstraktní třídy **DatasetSourceBase** a implementují její abstraktní metody. Zásuvné moduly doporučovacích modelů dědí od abstraktní třídy **ModelBase**. Zásuvné moduly vyžadují taktéž implementaci datového schéma přijímaných parametrů. Schéma je reprezentované datovým modelem z knihovny Pydantic[31]. Soubory s třídami zásuvných modulů je následně nutné vložit do souborového adresáře, odkud je doporučovací aplikace dynamicky načítá. Zásuvné moduly je nakonec potřeba registrovat v aplikaci pomocí REST API rozhraní,

[31] `https://docs.pydantic.dev/latest/`

aby se přidali do databáze záznamy nově přidaných doporučovacích modelů a zdrojů datové sady.

4. Uživatelská dokumentace

V této kapitole si popíšeme uživatelskou dokumentaci doporučovacího systému. Uvedeme návod jak systém používat pomocí různých komunikačních rozhraní, které systém a aplikace poskytují. Nakonec si také popíšeme možnosti nastavení systému a jeho spuštění.

4.1 REST API

Základní rozhraní pro komunikaci s doporučovací aplikací. Rozhraní umožňuje interakci mezi externí aplikací nebo službou a doporučovací aplikací pomocí HTTP protokolu. Komunikace probíhá posláním HTTP požadavku a získáním odpovědi. REST API poskytuje sadu koncových bodů, které přijímají HTTP požadavky s určitými metodami a vracejí odpovědi s daty ve formátu JSON[1].

Koncové body rozhraní poskytují přístup ke zdrojům doporučovací aplikace a mají Uniform Resource Identifier (URI)[2]. Rozlišujeme navíc URI kolekce a URI konkrétního prvku kolekce. Například `'/models'` identifikuje zdroj kolekce doporučovacích modelů a `'/models/elsa'` identifikuje zdroj konkrétního doporučovacího modelu. Zdroje definují metody HTTP požadavků, které reprezentují požadovanou operaci, jenž se má se zdrojem provést.

1. **GET**: metoda, která se používá pro získání kolekce nebo jejího záznamu. Požadavek s touto metodou by neměl mít jiný vliv na data kromě jejich získání. Použití s URI kolekce i konkretního prvku.

2. **POST**: metoda, která se používá pro vytvoření nového záznamu kolekce. S požadavkem se na server odešlou i data, pomocí nich se záznam vytvoří. Použití s URI kolekce, ve které chceme záznam vytvořit. Identifikátor záznamu se automaticky přidělí během vytváření a zašle se zpátky jako odpověď.

3. **PUT**: metoda pro úpravu existujícího záznamu. S požadavkem se na server odešlou data upravující konkrétní záznam. Použití s URI záznamu kolekce, který chceme upravit.

4. **DELETE**: metoda, jenž se používá k smazání konkrétního záznamu kolekce. Použití s URI záznamu kolekce, který chceme smazat.

Požadavek HTTP se skládá z URL adresy koncového bodu, HTTP metody, verze HTTP protokolu, hlaviček udávají doplňující informace požadavku a těla požadavku ve formátu uvedeném v hlavičce požadavku. Nejdůležitějšími informacemi v hlavičce požadavku jsou identifikace klienta, autentizace, formát přijímaných a odeslaných dat v těle požadavku a preferovaný jazyk. Příklady požadavků budeme uvádět ve formátu volání nástroje příkazové řádky cURL[3]. URL adresu požadavku budeme uvádět s lokální adresou serveru `localhost` a aplikačním portem 8000. Adresu a port je nutné při reálném použití změnit na skutečné hodnoty.

[1] https://www.json.org/json-en.html

[2] `https://datatracker.ietf.org/doc/html/rfc3986`

[3] `https://curl.se/docs/manpage.html`

Příklad požadavku pro získání kolekce doporučovacích modelů z aplikace může například vypadat takto.

```
$ curl -X GET \
  -H "Accept: application/json" \
  -H "Content-Type: application/json" \
  "http://localhost:8000/api/v1/models"
```

Struktura odpovědi je podobná struktuře HTTP požadavku. Odpověď navíc obsahuje stavový kód HTTP, který je součástí hlavičky a udává informaci o stavu zpracování požadavku serverem.

- **2****: značí úspěch. Požadavek byl na serveru úspěšně zpracován a jeho odpověď bude záviset na použité metodě požadavku.
- **4****: značí chybu na straně klienta. Například špatná struktura požadavku.
- **5****: značí chybu na straně serveru.

Příklad úspěšné odpovědi se stavovým kódem `200` a formátem dat v těle odpovědi `application/json`.

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 256

{
  "models": [...]
}
```

Kompletní přehled implementovaných koncových bodů REST API rozhraní doporučovací aplikace lze nalézt v **openapi**[4] specifikaci vyskytující se v příloze A.1 této práce. Specifikace obsahuje také ukázkové příklady požadavků a odpovědí.

4.1.1 Autentizace požadavků

Rozhraní doporučovací aplikace poskytuje tak zvané chráněné zdroje, k jejichž přístupu je zapotřebí autentizace a autorizace projektu, využívající doporučovací systém. Rozhraní používá pro autentizace a autorizaci projektů protokol OAuth 2.0[5].

Během registrace projektu v doporučovacím systému se uvádí textový identifikátor projektu a jeho heslo. Tyto informace se následně využívají pro ověření projektu. Projekt se ověří volání koncového bodu `/auth/token` rozhraní doporučovací s odpovídajícím textovým identifikátorem a heslem. Odpovědí tohoto požadavku je vygenerovaný přístupový token, pomocí kterého se projekt autorizuje u chráněných zdrojů. Přístupový token obsahuje informace o ověřeném projektu a vypršení jeho platnosti. Nyní si uvedeme příklady ověření projektu a HTTP požadavek chráněného zdroje.

[4] `https://www.openapis.org/`
[5] `https://oauth.net/2/`

Příklad HTTP požadavku pro ověření projektu.

```
$ curl -X POST \
  -H "Content-Type: application/x-www-form-urlencoded" \
  -d "grant_type=password&username=<project-id>&password=<password> \
  "http://localhost/api/v1/auth/token"
```

Odpověď s vygenerovaným přístupovým tokenem.

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 512

{
"access_token": "<access_token>",
"token_type": "bearer"
}
```

Příklad HTTP požadavku chráněného zdroje.

```
$ curl -X GET \
  -H "Accept: application/json" \
  -H "Authorization: Bearer <access_token>" \
  "http://localhost/api/v1/tasks"
```

4.2 CLI

Vedlejší rozhraní pro komunikaci s doporučovací aplikací pomocí příkazové řádky. Rozhraní se využívá zejména při vývoji a pro automatizaci doporučovacího systému. Příkazy rozhraní spouštíme pomocí příkazové řádky v souborovém adresáři doporučovací aplikaci a klíčového slova `recs`. Rozhraní poskytuje tři příkazy, které si teď popíšeme.

4.2.1 Trénování modelu

Příkaz `train-model` umožňuje spuštění trénovacího procesu doporučovacího modelu pro konkrétní projekt.

```
$ recs train-model [OPTIONS] MODEL
```

Příkaz má jeden argument `MODEL`, který udává textový identifikátor modelu, jenž chceme natrénovat. Dále má příkaz jednu povinnou možnost `--project, -p`, která udává textový identifikátor projektu, pro který chceme model natrénovat a na jehož datové sadě se proces trénování spustí. Zde je uveden příklad spuštění trénovacího procesu doporučovacího modelu **AlsMatrixFactorization** pro příkladový projekt.

```
$ recs train-model -p sample_project als_matrix_factorization
```

Doporučovací modely se na začátku trénovacího procesu inicializují pomocí sady hyper-parametrů. Pro nastavení hyper-parametrů, rozhraní využívá uložené trénovací kontexty modelů. To znamená, že musíme mít vytvořený trénovacího kontext modelu, pro který chceme spustit proces trénování pomocí příkazové řádky.

4.2.2 Aktualizace datové sady

Příkaz `update-dataset` umožňuje spuštění procesu aktualizace datové sady projektu.

```
$ recs update-dataset [OPTIONS]
```

Příkaz má jednu povinnou možnost `--project, -p`, která udává textový identifikátor projektu, jehož datovou sadu chceme aktualizovat. Datová sada se aktualizuje ze zdroje nastaveného v kontextu datové sady určeného projektu. Zde je uveden příklad aktualizace datové sady příkladového projektu.

```
$ recs update-dataset -p sample_project
```

4.2.3 Spuštění experimentu

Příkaz `run-experiment` umožňuje spuštění experimentu. Pomocí experimentů zkoumáme výkonnost doporučovacích modelů a hybridních přístupů na datové sadě konkrétního projektu. Experimenty jsou definovány pomocí konfiguračních souborů ve formátu JSON.

```
$ recs run-experiment [OPTIONS] EXPERIMENT
```

Příkaz má jeden argument, kterým je **EXPERIMENT**, jenž udává cestu v adresářové struktuře ke konfiguračnímu souboru experimentu. Dále má příkaz jednu povinnou možnost `--project, -p`, která udává projekt na jehož datové sadě se experiment vykoná. Specifikaci konfiguračních parametrů experimentu, použití a příklady konkrétních experimentů můžeme nalézt ve vývojové dokumentaci doporučovací aplikace.

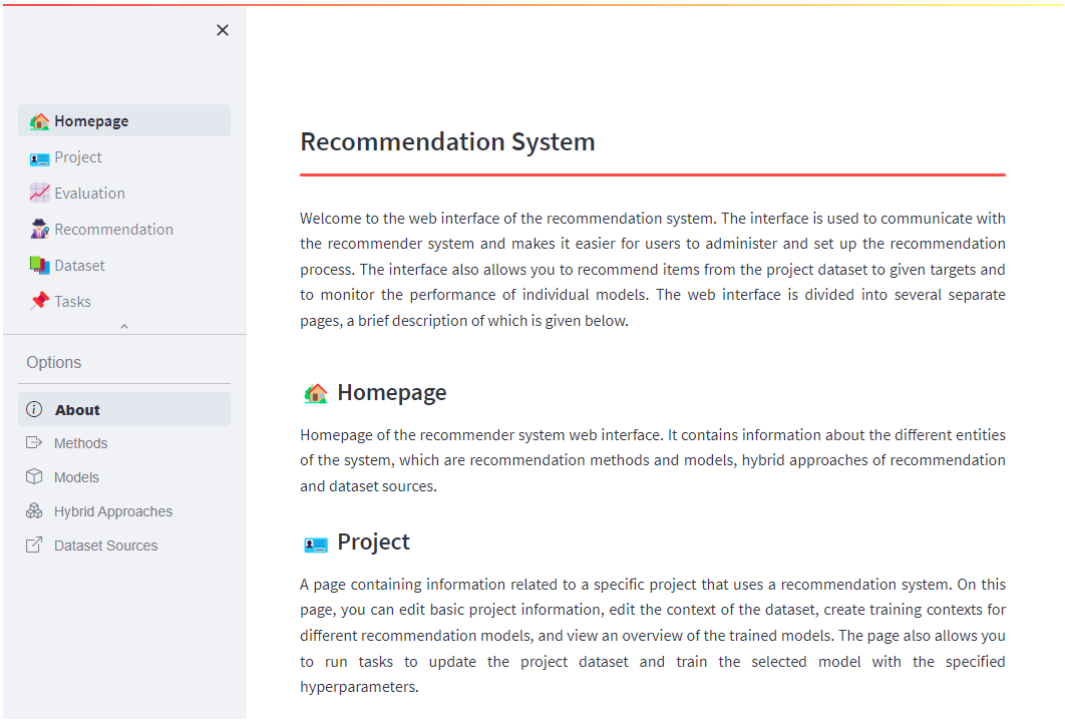
4.3 Webové rozhraní

Webové rozhraní umožňuje uživateli snadno komunikovat s doporučovací aplikací pomocí interaktivního prostředí.

4.3.1 Rozložení

Rozložení webového rozhraní si popíšeme na obrázku 4.1. Webové rozhraní je složeno z několika stránek, které zobrazují svůj obsah v závislosti na vybrané možnosti stránky. Stránka webového rozhraní má dvě části. První částí je postranní panel, který můžeme vidět vlevo na obrázku. Panel obsahuje menu, které se nachází v horní části, a možnosti stránky, jenž se nacházení v dolní části panelu. Menu umožňuje uživateli přecházet mezi různými stránkami webového rozhraní a možnosti stránky určují zobrazený obsah na stránce. Druhou částí stránky je panel zobrazující její obsah a nachází se napravo od postranního panelu.

Postranní panel je možné skrýt pomocí křížku nacházející se v pravém horním rohu postranního panelu a opět zobrazit pomocí šipky, která se po skrytí panelu zobrazí v levém horním rohu stránky. Obsah každé stránky začíná nadpisem a krátkým popisem jejího kontextu. Obsah stránky se zobrazuje v závislosti na vybrané možnosti stránky v postranním panelu.



Obrázek 4.1: Domovská stránka webového rozhraní.

4.3.2 Domovská stránka

Domovská stránka obsahuje základní informace o doporučovacím systému. Na stránce lze zvolit jednu z pěti možnosti zobrazení obsahu. První možnost **About** zobrazí základní popisy všech stránek webového rozhraní. Další možnosti zobrazují přehled doporučovacích metod, doporučovacích modelů, hybridních přístupů a registrovaných datových zdrojů. Příklad přehledu můžeme vidět na obrázku 4.2. Zde se zobrazuje přehled implementovaných doporučovacích modelů. Modely lze filtrovat podle implementovaných doporučovacích metod. Každý záznam modelu obsahuje jeho krátký popis, implementované doporučovací metody a informaci o tom zda je model aktivní či nikoli. Zbylé přehledy vypadají podobně.

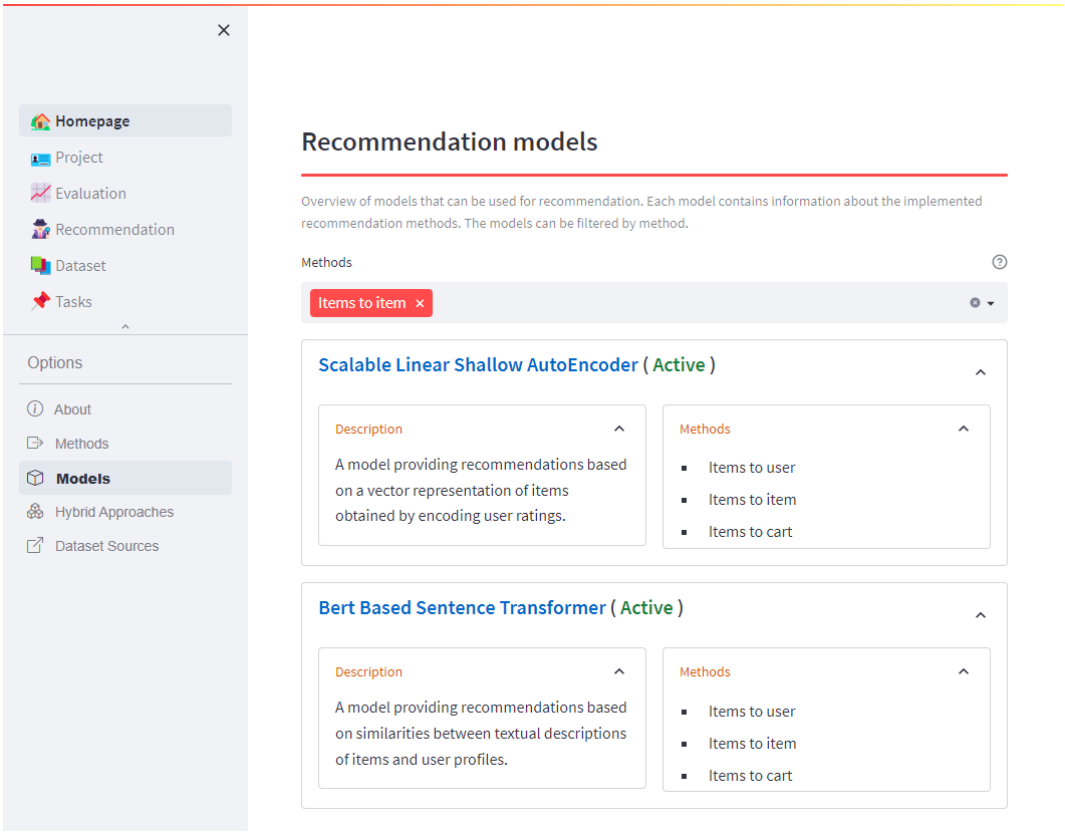
4.3.3 Registrace a přihlášení

Domácí stránka je jediná, která zobrazuje obsah bez nutnosti přihlášení nebo registrace. Obsah na domácí stránce je obecný a ostatní stránky zobrazují obsah, který patří konkrétnímu projektu. Pokud se chceme pomocí menu přesunout na jakoukoliv jinou stránku a nejsme ještě přihlášený pod nějakým projektem, ukáže se nám stránka s možností přihlášení nebo registrace projektu.

Formulář pro přihlášení k projektu můžeme vidět na obrázku 4.3. Přihlášení vyžaduje textový identifikátor projektu, ke kterému se přihlašujeme, a odpovídající heslo.

Pokud chceme projekt registrovat do systému, zvolíme v postranním panelu možnost **Signup**. Stránka zobrazí formulář pro registraci projektu do doporučovacího systému, který lze vidět na obrázku 4.4. Registrační formulář se skládá ze dvou částí, které obsahují povinné parametry, jenž je potřeba vyplnit.

V první části se vyžadují základní informace o registrovaném projektu. Povin-

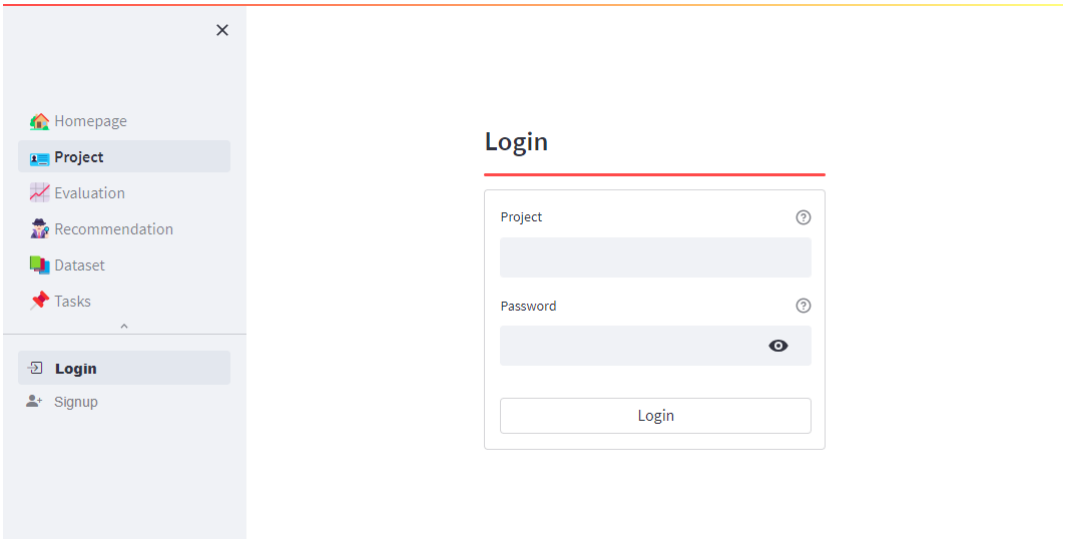


Obrázek 4.2: Přehled doporučovacích modelů.

nými parametry jsou název projektu, textový identifikátor a heslo. Poslední dva zmíněný se používají pro přihlášení k projektu a nelze je po ukončení registrace měnit. Nepovinným parametrem je popis projektu. Název projektu může libovolný, ale nesmí být prázdný nebo obsahovat více než 64 znaků. Maximální délka popisu projektu je z hora omezena 1024 znaky. Textový identifikátor projektu se může skládat pouze z velkých a malých písmen anglické abecedy, čísel a znaku podtržítka. Zároveň nemůže být delší než 64 znaků. Nakonec heslo může být také libovolné, ale nesmí být delší než 64 znaků.

Ve druhé části registračního formuláře projektu se nastavuje kontext jeho datové sady. Kontext datové sady udává, ze kterého zdroje se bude načítat datová sada projektu a případně může také udávat, v jakém časovém intervalu se bude datová sada automaticky aktualizovat. V nastavení kontextu datové sady uživatel zvolí jeden z dostupných zdrojů. Podle zvoleného zdroje se v nastavení zobrazí formulář pro vyplnění jeho parametrů. Na obrázku 4.4 můžeme vidět zvolený zdroj datové sady **GoodBooks**3.5.3. Tento zdroj nemá žádné parametry, proto je formulář prázdný. Parametry se liší pro každý zdroj a jejich podrobný popis je uveden ve vývojové dokumentaci doporučovací aplikace.

Kontextu datové sady dále můžeme nastavit časový interval, ve kterém se bude datová sada automaticky aktualizovat. Nastavení intervalu je nepovinné a pokud ho chceme nastavit, musíme zaškrtnou políčko **Use** pod jeho nastavením. Nastavení intervalu se skládá z hodnoty a jednotky času, přičemž hodnota musí být nezáporná a jednotka je týden, den, hodina nebo minuta. Po úspěšném přihlášení nebo registraci projektu se nám zobrazí obsah stránky na které se nacházíme.



Obrázek 4.3: Stránka s přihlášením projektu.

Pro odhlášení projektu použijeme tlačítko **Logout**, které se zobrazuje ve spodní části postranního panelu stránky **Project** pod výběrem možností.

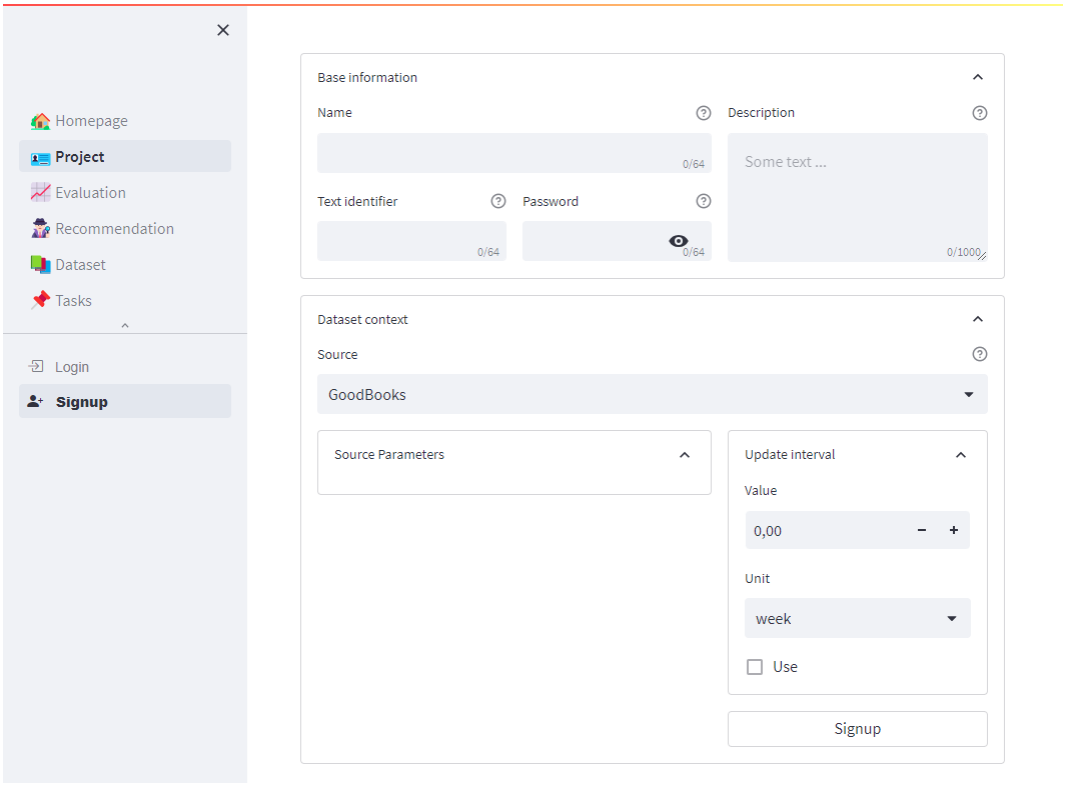
4.3.4 Administrace projektu

Stránka **Project** zobrazuje data, která souvisí s projektem, pod kterým jsme přihlášeni. Stránka nabízí čtyři možnosti obsahu, které si nyní popíšeme. První možností jsou základní informace projektu, obrázek 4.5. Zde se nachází formulář, který uživateli umožňuje změnit název a popis projektu. Pro prvky formuláře platí stejná omezení jako při registraci, kterou jsem si popsali výše.

Další možností je **Dataset**, jenž můžeme vidět na obrázku 4.6. Zde můžeme upravit kontext datové sady projektu, který jsme nastavovali při jeho registraci. Formuláře jsou identické a platí pro ně stejná omezení. Změny potvrdíme kliknutím na tlačítko **Update context**. Pokud jsem nedělali žádné změny v kontextu nebo máme všechny změny potvrzené, můžeme datovou sadu projektu manuálně aktualizovat, kliknutím na tlačítko **Update dataset**. Webové rozhraní odešle požadavek doporučovací aplikaci o vytvoření úlohy, která na pozadí systému aktualizuje datovou sadu podle nastaveného kontextu projektu. Informace o konkrétní úloze můžeme poté nalézt na stránce **Tasks**, kterou popíšeme později.

Možnost **Training** zobrazuje trénovací kontexty doporučovacích modelů a můžeme ji vidět na obrázku 4.7. Trénovací kontext sdružuje doporučovací model, který se má natrénovat s jeho nastavenými hyper-parametry. V horní části stránky vybereme z nabídky doporučovací model jehož trénovací kontext chceme spravovat. Ve formuláři pod ním pak vyplníme povinné hyper-parametry a časový interval, který určuje po jaké době se má model znovu natrénovat. Hyper-parametry se liší podle zvoleného modelu a jejich podrobný popis můžeme nalézt ve vývojové dokumentaci doporučovací aplikace. Časový interval je nepovinný a pokud bychom ho chtěli nastavit , musíme zaškrtnout políčko **Use**, které se nachází pod ním. Nastavení intervalu se skládá z hodnoty a jednotky času, přičemž hodnota musí být nezáporná a jednotka je týden, den, hodina nebo minuta.

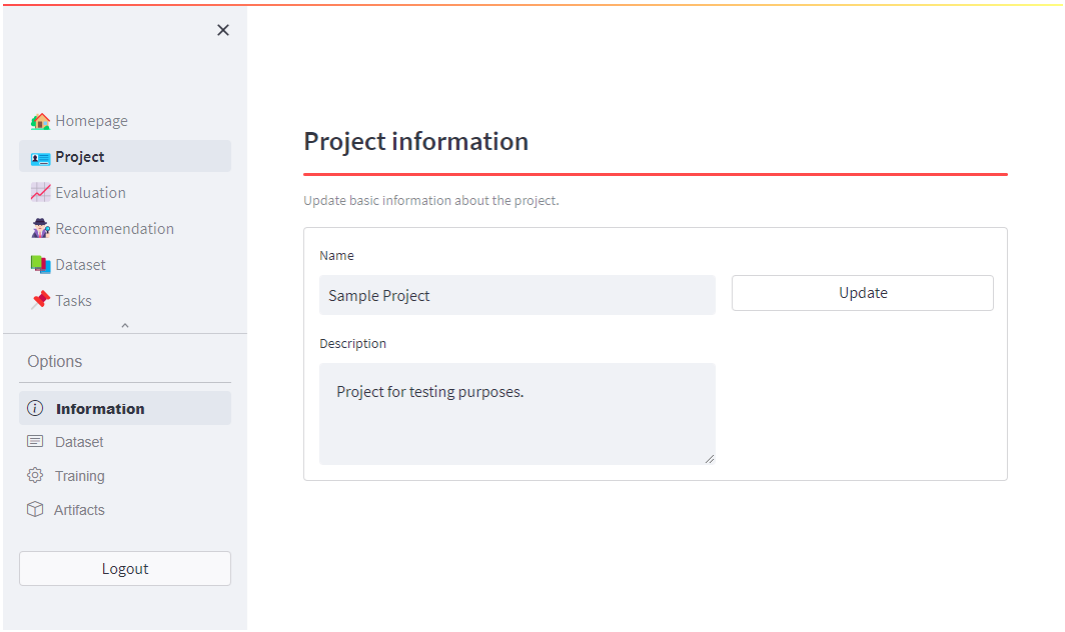
Pokud máme nastavený kontext, můžeme ho vytvořit nebo upravit, jestliže



Obrázek 4.4: Stránka s registrací projektu.

byl již nastavený. Formulář nám také umožňuje model s nastavenými hyper-parametry manuálně natrénovat, kliknutím na tlačítko **Train model**. Webové rozhraní potom odešle požadavek doporučovací aplikaci o vytvoření úlohy, která natrénuje daný model na pozadí doporučovacího systému. Pro vytvoření takové úlohy není nutné trénovací kontext modelu ukládat. Trénovací kontext slouží pro uložení nastavených hyper-parametrů modelu a pro jeho automatické znovu natrénování v případě nastaveného časového intervalu. Úlohu trénování modelu můžeme následně vidět na stránce **Tasks**.

Poslední možností této stránky je přehled artefaktů doporučovacích modelů. Artefakt reprezentuje výsledek trénovacího procesu modelu. Pokud byl model úspěšně natrénovaný a jeho stav je uložený v registru modelů, artefakt označuje natrénovaný model, který je možné použít pro vytvoření doporučení. V opačném případě artefakt označuje pouze záznam o trénování modelu. Přehled artefaktů můžeme vidět na obrázku 4.8. Přehled zobrazuje prvních 10 artefaktů zvoleného modelu, seřazené podle datum zahájení trénovacího procesu, které můžeme filtrovat podle jejich statusu. Pro zobrazení dalších části přehledu můžeme využít šipky ve spodní části postranního panelu. Artefakt obsahuje svoji unikátní značku, datum zahájení trénovacího procesu, v závorce dobu trénování a sadu hyper-parametrů, které byly použity pro inicializaci trénovaného modelu. Pokud má artefakt status přípraven (natrénovaný model, jenž artefakt reprezentuje, je uložen v registru modelů), můžeme artefakt smazat kliknutím na tlačítko **Delete**. Natrénovaný model se vymaže z registru modelů a artefaktu se přiřadí status smazán. Konkrétní artefakt modelu můžeme vyhledat podle jeho značky pomocí vyhledávacího formuláře, jenž se zobrazí po kliknutí na tabulátor **Search**.



Obrázek 4.5: Základní informace projektu.

4.3.5 Přehled vyhodnocení

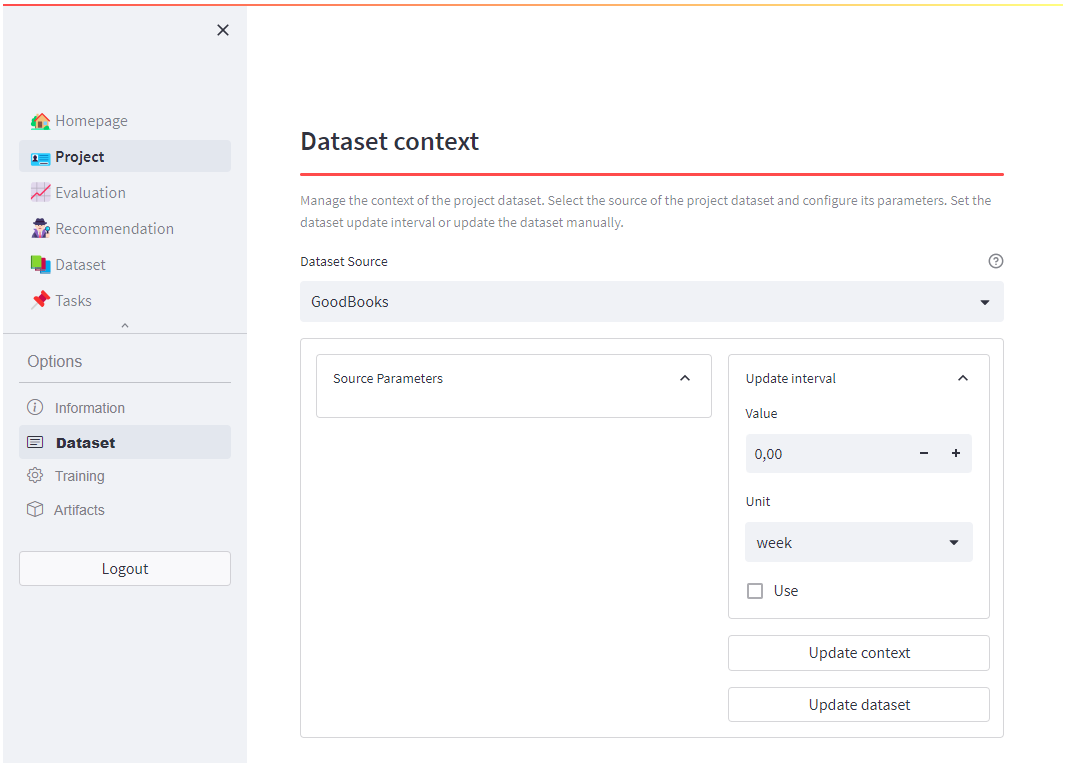
Přehled vyhodnocení výkonosti doporučování můžeme nalézt na stránce **Evaluation**. Stránka zobrazuje vyhodnocení pomocí grafů, které obsahují hodnoty vyhodnocovacích metrik v určeném časovém období. Navíc zobrazuje také sloupcový graf, jenž obsahuje četnosti doporučených a zpětnou vazbou hodnocených položek v tomto časovém období. Příklad vyhodnocení výkonosti doporučení můžeme vidět na obrázku 4.9.

Pomocí parametrů můžeme určit časové období, ve kterém se výkonost doporučení bude vyhodnocovat. Vyhodnocení probíhá po dnech a časové období musí být validní interval. Dále můžeme zvolit velikost doporučeného seznamu položek od počátku, jenž se má vyhodnotit. Velikost musí být nezáporné číslo vetší než 0. Nakonec také můžeme vybrat agregační funkci, která se používá k výpočtu hodnot vyhodnocovacích metrik, jenž se zobrazují nad jejich grafy. Výchozí agregační funkce zobrazuje poslední hodnoty metrik z určeného časového období.

Stránka umožňuje zobrazit vyhodnocení pro doporučovací modely a skupiny doporučení. V prvním případě se vyhodnocení výkonosti doporučení zobrazuje pro zvolený doporučovací model. Krom toho můžeme také zvolit konkrétní artefakt daného modelu a případně doporučovací metodu, pokud jich model implementuje více. Po kliknutí na tabulátor **Comparison** se zobrazí výběr dvou modelů a jejich artefaktů čí doporučovacích metod. Vyhodnocení výkonosti zvolených modelů se následně zobrazí vedle sebe tak, aby se daly vizuálně porovnat. Vyhodnocení výkonosti skupiny doporučení můžeme zobrazit v možnosti **Group** této stránky.

4.3.6 Vytvoření doporučení

Základní vlastností doporučovacího systému je vytváření doporučení položek z datové sady projektu. Doporučení můžeme ve webovém rozhraní vytvořit na



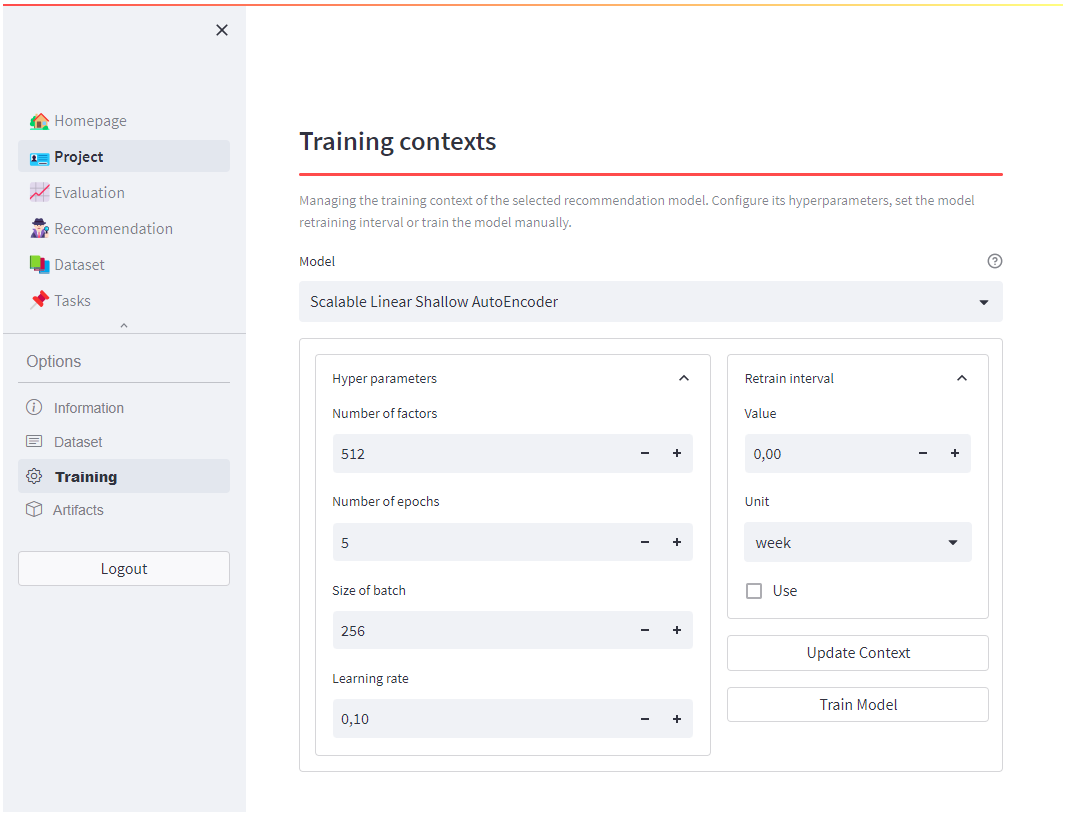
Obrázek 4.6: Nastavení kontextu datové sady.

stránce z názvem **Recommendation**. Stránku můžeme vidět na obrázku 4.10. Stránka nemá klasické možnosti zobrazení obsahu jako ostatní stránky. Možnosti stránky jsou různé doporučovací metody, které můžeme použít k vytvoření doporučení.

Všechny možnosti stránky vypadají podobně. V horní části stránky je výběr modelu, jenž se použije pro vytvoření doporučení a nalevo od něj je zvolení konkrétního artefaktu. Ve výběru se zobrazí pouze modely, které mají artefakt se statusem připraven. Pokud takové modely neexistují, stránka vypíše adekvátní zprávu a vytvoření doporučení není možné.

Pod výběrem modelu se nachází formulář s parametry, které jsou specifické pro každou metodu doporučování. Metody mají některé parametry společné. Jsou jimi počet doporučených položek a seznam kandidátů, ze kterých je výsledné doporučení složeno. Oba tyto parametry jsou nepovinné, výchozí hodnota počtu doporučených položek je 10 a výchozí hodnota kandidátů je prázdný seznam. Pokud formulář požaduje položku nebo seznam položek, myslí se tím identifikátory daných položek, tak jak jsou uvedeny v datové sadě. Seznam identifikátorů se do formuláře vyplňuje pomocí textového řetězce a jednotlivé identifikátory jsou oddělené čárkou. Další parametry jednotlivých metod jsou následující.

- **Items to user**: má povinný parametr určující uživatele pro kterého chceme doporučení vytvořit. Uživatel se určuje pomocí jeho identifikátoru. Dalším nepovinným parametrem je identifikátor skupiny, do které chceme doporučení zařadit. Skupina se využívá pro následné vyhodnocení doporučování.
- **Items to item**: má povinný parametr určující položku, ke které chceme vytvořit doporučení.

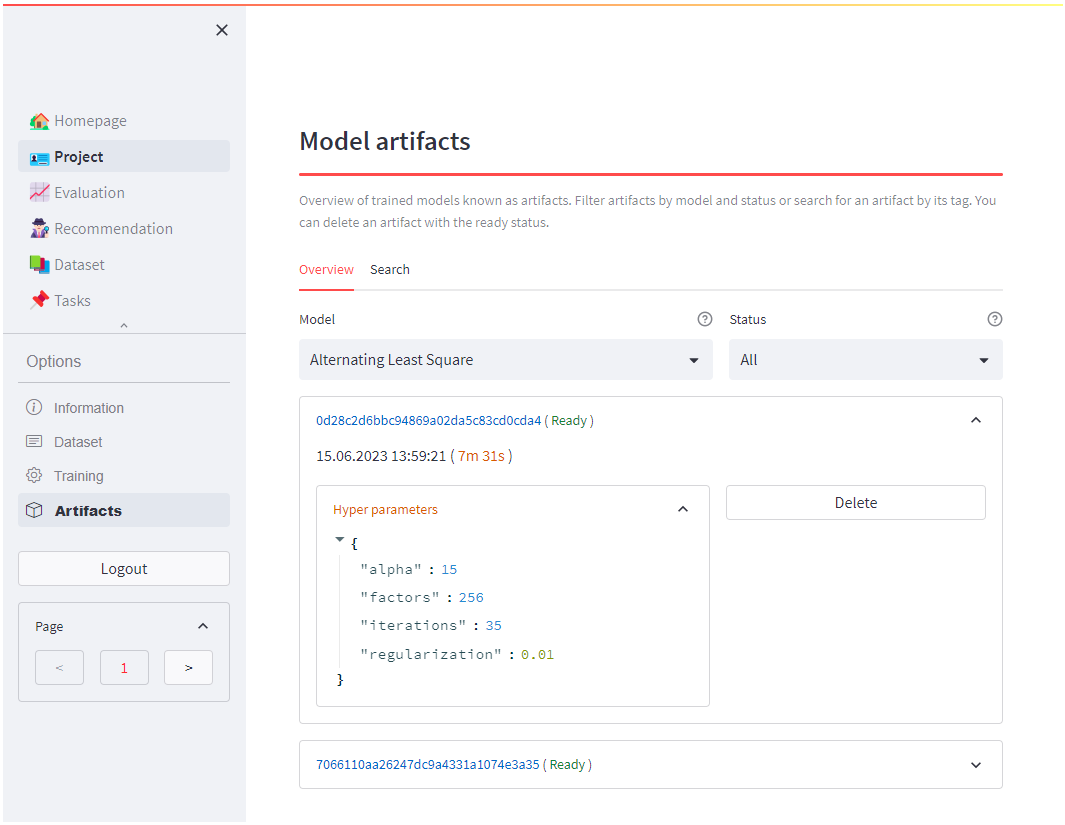


Obrázek 4.7: Nastavení kontextu trénování modelu.

- **Items to cart**: má povinný parametr určující seznam položek v nákupním košíku, pro který chceme vytvořit doporučení. Dalším nepovinným parametrem je uživatel, jenž je vlastníkem nákupního košíku. Pokud je uživatel uveden, doporučení může být personalizované.

Ve formuláři lze dále nastavit, jakým způsobem bude výsledné doporučení upravené. Toto nastavení je nalevo od parametrů metod a liší se opět podle zvolené metody doporučení. U metod, které mají parametr uživatele, můžeme zvolit možnost filtrování. Filtrování zajistí, že ve výsledném doporučení se nebudou nacházet položky, které již daný uživatel hodnotil. U metody **Items to user** můžeme využít možnost doplnění doporučení pro neznámého uživatele pomocí nepersonalizované doporučení. Pokud je tedy určený uživatel pro zvolený doporučovací model neznámí, doporučení se vytvoří například na základě popularity položek v datové sadě. Má-li projekt natrénovaný model **BertBased** 3.4.3, můžeme nastavit hodnotu koeficientu lambda. Pomocí snížení hodnoty lambda koeficientu můžeme zvýšit diverzitu položek ve výsledném doporučení. Tato úprava se provádí pomocí metody MMR, jenž je popsaná v článku (Carbonell a Goldstein, 1998). Hodnota koeficientu 1 znamená maximalizaci relevance doporučených položek. Hodnota 0 znamená maximalizaci diverzity položek doporučení.

Pokud máme správně vyplněné všechny povinné parametry, můžeme kliknout na tlačítko **Recommend**, které se nachází vpravo dole ve formuláři. Webové rozhraní odešle požadavek doporučovací aplikaci na vytvoření doporučení pro určený cílový subjekt podle zvolené doporučovací metody. Pokud je doporučení neúspěšné, zobrazí se pod formulářem adekvátní chybová zpráva. V opačné případě proběhne doporučení položek úspěšně a pod formulářem se zobrazí doporučené

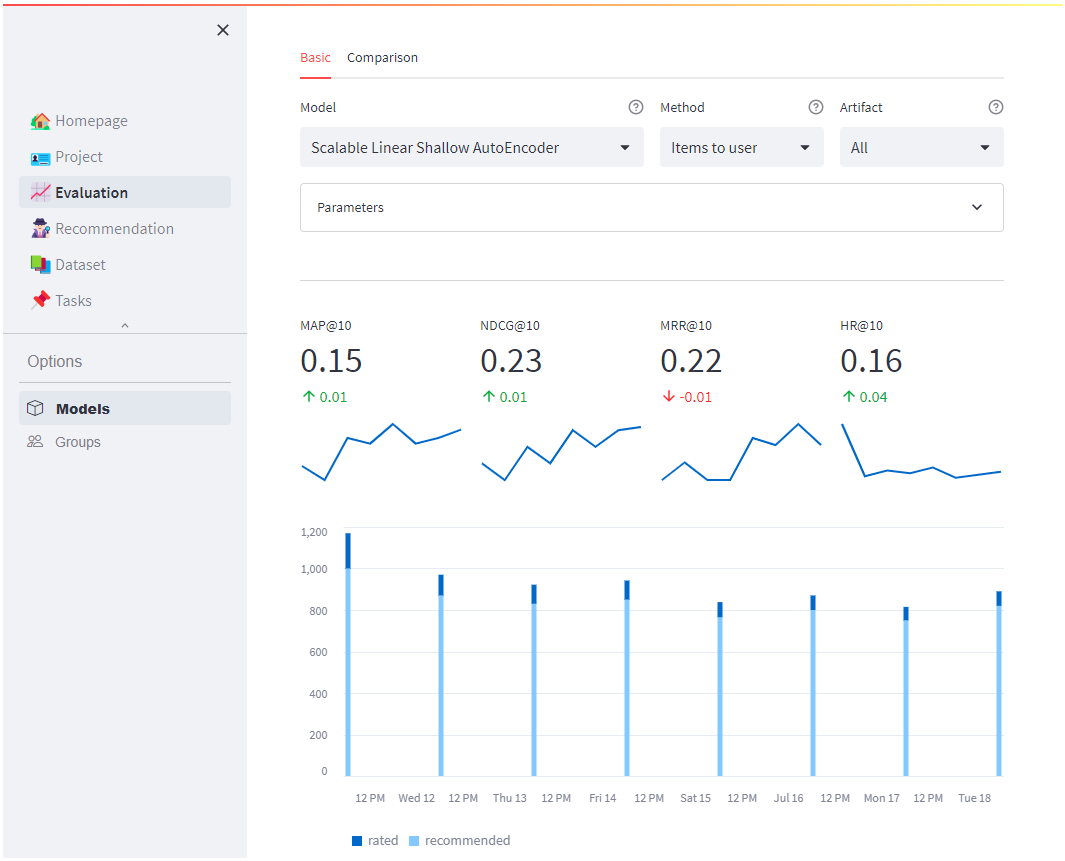


Obrázek 4.8: Přehled natrénovaných modelů.

položky, jenž můžeme vidět na obrázku 4.11. Položky se na stránce zobrazují ve dvou pohledech. Prvním je **Grid**, který zobrazí obrázky položek s jejich názvy v mřížce. Pokud položka nemá dostupný obrázek, zobrazí se místo něj zástupný statický obrázek. Druhým pohledem je **List**, který zobrazuje položky v seznamu pod sebou a u každé položky navíc zobrazí její ostatní vlastnosti. Tento pohled můžeme vidět na obrázku 4.14, jenž pohled ukazuje na stránce datové sady. U každé doporučené položky můžeme rovnou vytvořit zpětnou vazbu doporučení. Vyplněním formuláře pod vlastnostmi položky vytvoříme její hodnocení určeným uživatelem. Pokud se jedná o metody, které doporučují položky uživateli (identifikátor uživatele je vyplněn v parametrech doporučení), je políčko **User** pevně předepsané. Hodnocení vytvořené jako zpětná vazba konkrétního doporučení se využívá pro vyhodnocení výkonosti doporučování. Doporučovací metody **Items to item** a **Items to cart** zobrazují spolu s přehledem doporučených položek také referenční položky pro které bylo doporučení vytvořené.

4.3.7 Vytvoření hybridního doporučení

Doporučovací aplikace nabízí dva způsoby vytvoření hybridního doporučení. První způsob je využití natrénovaného doporučovacího modelu Monolith 3.4.3, jenž je zástupcem modelů poskytující hybridní doporučení. Model již v procesu trénování kombinuje principy kolaborativního filtrování a doporučení založeném na obsahu. Hybridní doporučení poté vytvoříme na stránce **Recommendation**, kde vybereme tento model z nabídky dostupných modelů pro vytvoření doporučení.

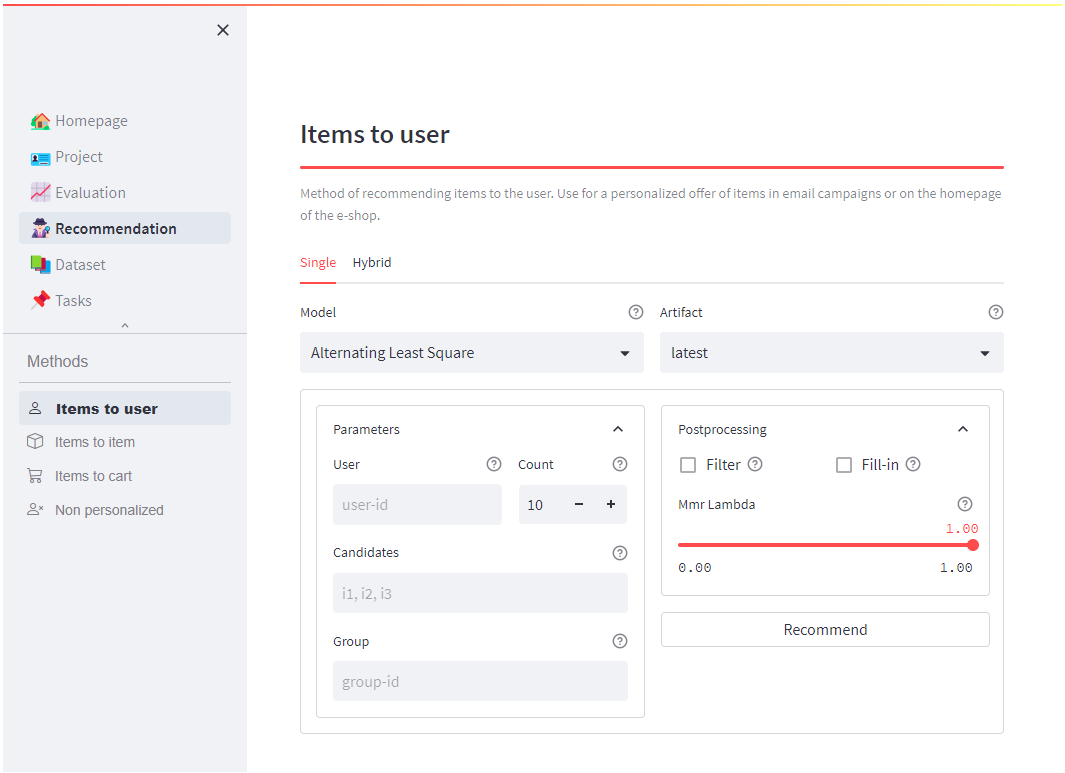


Obrázek 4.9: Přehled vyhodnocení výkonosti doporučování.

Druhý způsob vytvoření hybridního doporučení je kombinací různých doporučovacích modelů v průběhu procesu doporučení položek. Doporučovací aplikace tento způsob implementuje pouze pro metodu doporučení **Items to user**. Ve webovém rozhraní můžeme vytvořit hybridní doporučení druhým způsobem na stránce **Recommendation** s odpovídající metodou po kliknutí na tabulátor **Hybrid**. Formulář pro vytvoření hybridního doporučení můžeme vidět na obrázku 4.12. Na stránce se zobrazí výběr hybridního přístupu 3.4.4, který udává, jakým způsobem se dílčí doporučení různých modelů kombinují. Vedle výběru přístupů je výběr modelů, jimiž vytvořené doporučení chceme zkombinovat do jednoho výsledného. Je povinné vybrat minimálně dva doporučovací modely. Hybridní přístup **Cascade** požaduje výběr právě dvou modelů. Výběr obsahuje pouze natrénované modely, které mají artefakt se statusem připraven. Pokud projekt nemá natrénovaný dostatečný počet různých modelů, není možné tento způsob hybridního doporučení použít. Pod výběrem hybridního přístupu je opět nastavení parametrů doporučování určené metody a nastavení způsobu upravení doporučení. Rozdílem oproti normálnímu doporučení je nastavení parametrů zvoleného hybridního přístupu. Parametry jsou specifické pro zvolený přístup3.4.4.

4.3.8 Přehled datové sady projektu

Datová sada se skládá z kolekcí 3.4.1, jejichž přehledy se zobrazují na stránce **Dataset**. Stránka zobrazuje pouze kolekce položek a hodnocení mezi jejichž přehledy se můžeme přesouvat pomocí možností stránky v postranním panelu.



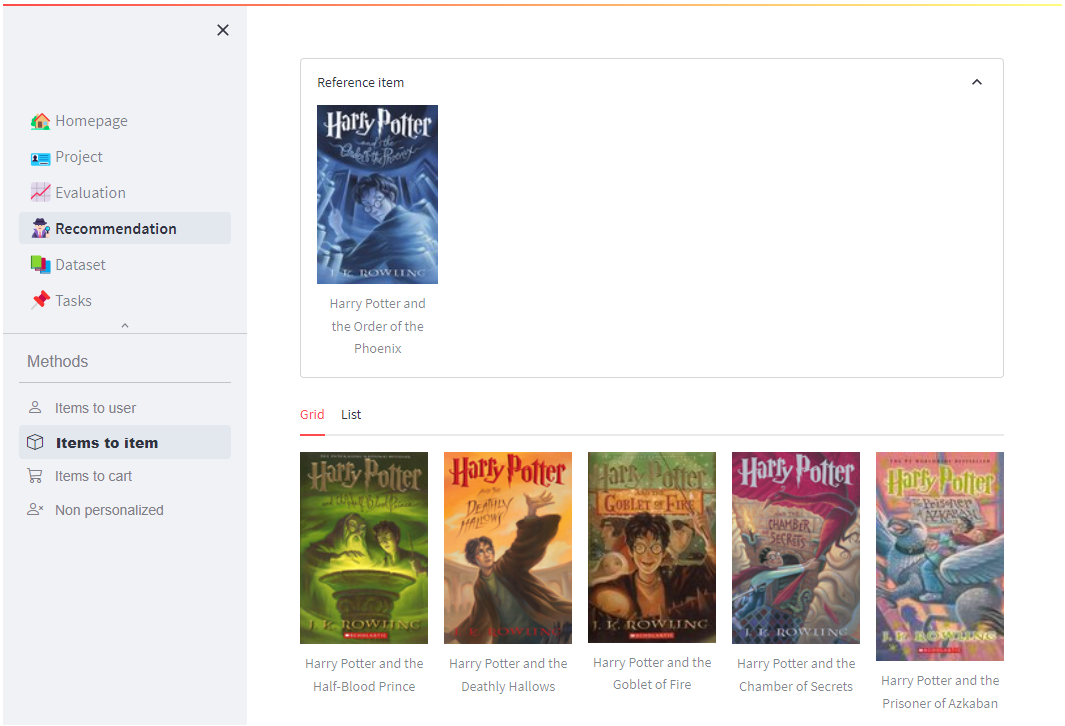
Obrázek 4.10: Vytvoření doporučení.

Přehled kolekce položek se zobrazuje, podobně jako u přehledu doporučených položek, ve dvou pohledech **Grid** a **List**, které můžeme vidět na obrázcích 4.13 a 4.14. Položky můžeme vyhledávat pomocí jejich identifikátorů nebo názvů. Pokud vyhledáváme konkrétní položku podle identifikátoru, hodnota názvu ve vyhledávacím formuláři bude ignorována. Jestliže vyhledáváme položky podle názvu, přehled zobrazí ty, jejichž název je nejpodobnější k požadovanému. Přehled zobrazuje naráz pouze 20 položek a pokud bychom chtěli zobrazit další, můžeme využít šipky v dolní části postranního panelu stránky. Je-li přehled v pohledu **List**, můžeme pomocí vyplnění formuláře ve vlastnostech položky vytvořit její hodnocení pro konkrétního uživatele.

Přehled kolekce hodnocení se zobrazuje pro určeného uživatele a můžeme jej vidět na obrázku 4.15. Stejně jako kolekce položek se kolekce hodnocení zobrazuje ve dvou pohledech s omezením po 20 záznamech. Záznam v pohledu **List** obsahuje hodnocenou položku, hodnotu hodnocení, datum hodnocení a možnosti pro úpravu nebo smazání daného hodnocení položky uživatelem.

4.3.9 Přehled úloh

Úlohy se v doporučovacím systému využívají pro trénování doporučovacích modelů nebo aktualizaci datové sady projektu na pozadí mimo hlavní proces doporučovací aplikace. Přehled úloh, které patří přihlášenému projektu, můžeme vidět na stránce **Tasks**. Stránka zobrazuje přehled úloh zvoleného typu, které můžeme filtrovat podle hodnoty jejich statusů. Přehled zobrazuje prvních 20 úloh, seřazených podle data vytvoření. Pro zobrazení dalších úloh můžeme využít šipky ve spodní části postranního panelu stránky. Úloha obsahuje identifikátor, status,



Obrázek 4.11: Přehled doporučených položek.

datum vytvoření úlohy, v závorce dobu zpracování úlohy a data, která byla použita pro její vytvoření. Pokud se jedná o úlohu typu trénování modelu, můžeme ji znovu zpracovat kliknutím na tlačítko **Rerun**. Webové rozhraní zašle požadavek doporučovací aplikaci o vytvoření nové úlohy se stejnými daty. Nová úloha se vytvoří a zařadí se do fronty pro zpracování. Konkrétní úlohu můžeme vyhledat podle identifikátoru pomocí vyhledávacího formuláře, jenž se zobrazí po kliknutí na tabulátor **Search**. Přehled úloh můžeme vidět na obrázcích 4.16 a 4.17.

4.4 Spuštění systému

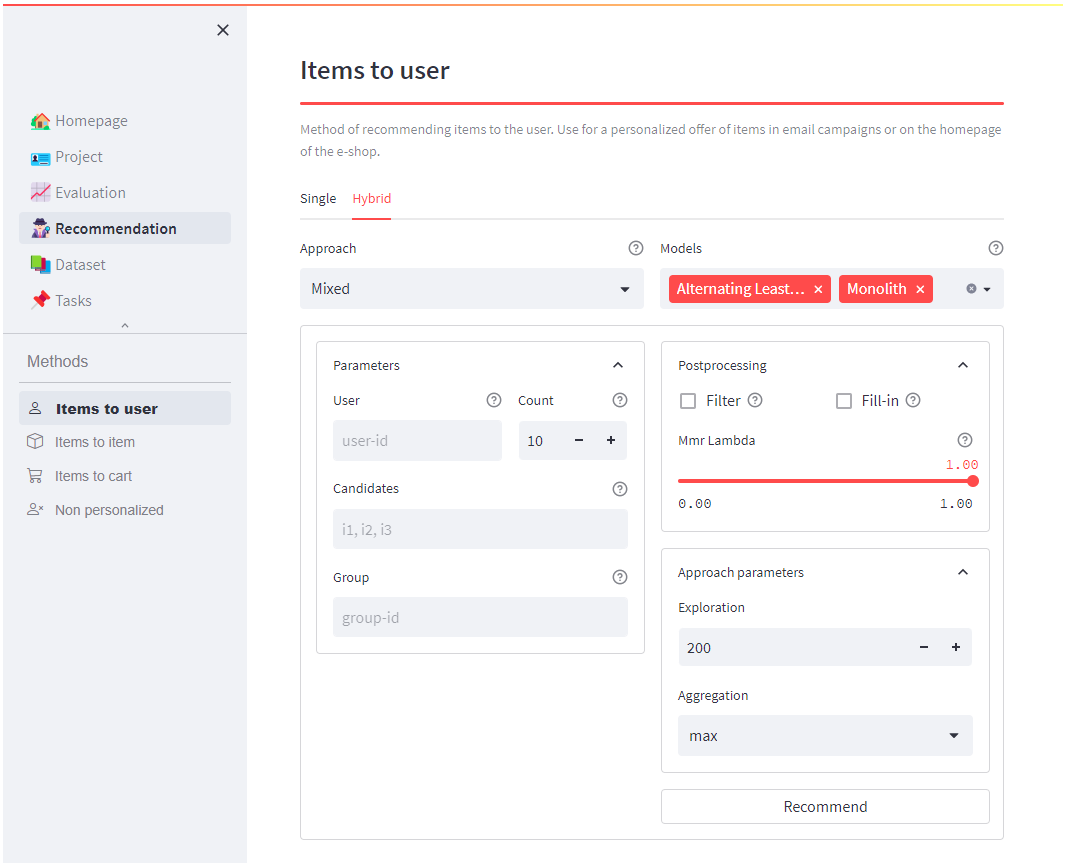
V této sekci si popíšeme jak spustit doporučovací systém. Jako první uvedeme softwarové a hardwarové požadavky, poté popíšeme adresářovou strukturu systému a instalaci závislostí, a nakonec ukážeme jak doporučovací systém spustit.

4.4.1 Požadavky

Doporučovací systém je napsaný v jazyce Python verze 3.10 a využívá nástroj Poetry[6] 1.4.2 pro správu aplikačních závislostí. Systém budeme spouště pomocí nástroje Docker[7] 4.20.1, a jestliže budeme zdrojový kód získávat z veřejného repositáře, budeme také potřebovat nástroj Git[8].

Doporučovací systém je náročný na hardwarové zdroje. Proto zde uvádíme minimální požadavky pro jeho spuštění, které jsem odvodili při testování.

[6] `https://python-poetry.org/`
[7] `https://www.docker.com/`
[8] `https://git-scm.com/`



Obrázek 4.12: Vytvoření hybridního doporučení.

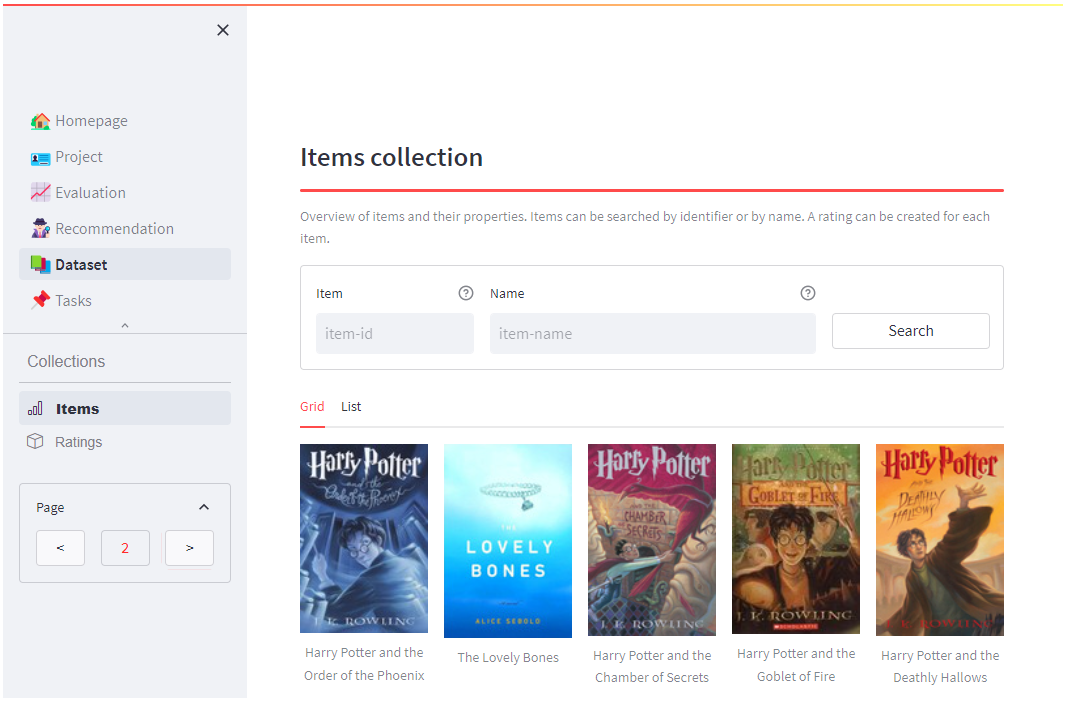
- **CPU**: více-jádrový procesor s možností virtualizace. Virtualizace je nutná pro spuštění pomocí Dockeru.
- **RAM**: minimální velikost 10GB pro načtení datové sady a natrénovaného doporučovacího modelu.
- **HDD/SDD**: minimální velikost místa na disku 15GB pro uložení dat systému. Všechna data se ukládají lokálně na disk.

4.4.2 Adresářová struktura projektu

Jak jsme již popsali v kapitole 3.2, doporučovací systém se skládá z několika částí, které si předávají data a vykonávají s nimi určité operace. Každá část systému reprezentuje samostatnou aplikaci s vlastním běhovým prostředím.

V následujícím textu se zaměříme na část doporučovací aplikace a webového rozhraní. Obě tyto části systému jsou aplikace napsané v jazyce Python a jsou uložené v jednom projektu ve veřejném repositáři, odkud je můžeme stáhnout. Projekt reprezentuje doporučovací systém a je rozdělen na dvě části. **Backend** označuje doporučovací aplikaci a **frontend** webové rozhraní systému. Projekt stáhneme z repositáře pomocí následujícího příkazu.

```
$ git clone project-url recommendation-system
```



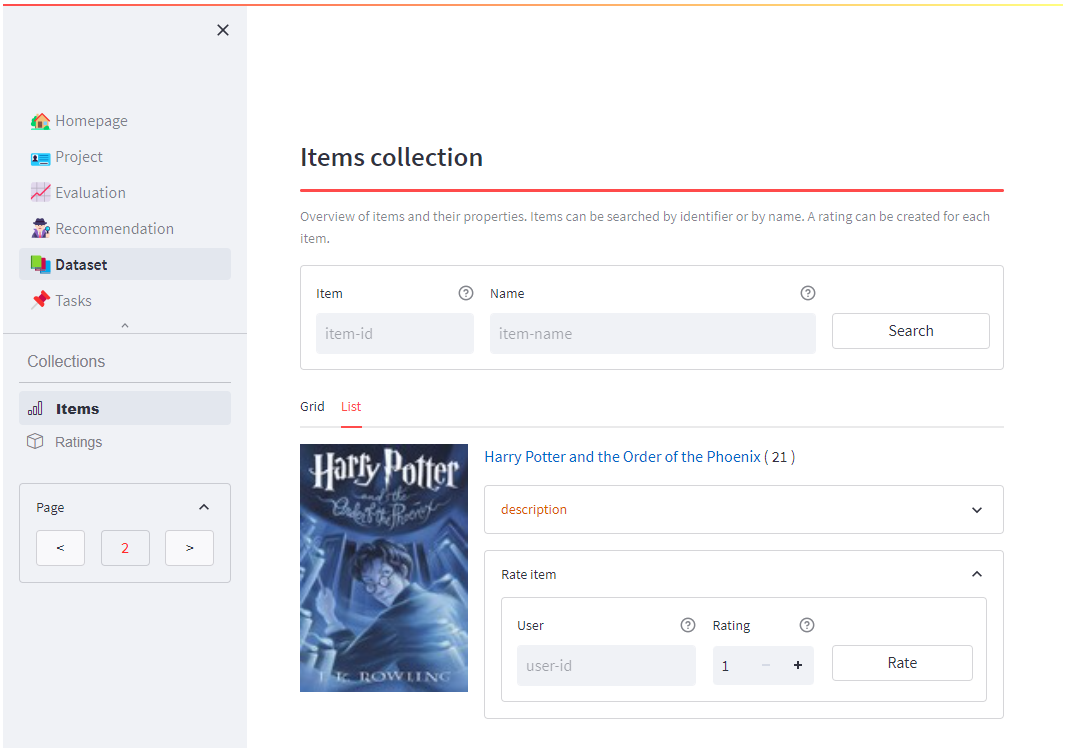
Obrázek 4.13: Přehled položek v datové sadě.

Příkaz stáhne do adresáře, ve kterém se právě nacházíme projekt doporučovacího systému s názvem `recommendation-system`. Kořenový adresář doporučovací aplikace má umístění `{pwd}/recommendation-system/backend` a kořenový adresář webového rozhraní má umístění `{pwd}/recommendation-system/frontend`. Kořenové adresáře obsahují soubory spojené s instalací aplikačních závislostí a se spuštěním aplikací. Adresář doporučovací aplikace je dále rozdělen do následujících podadresářů.

- **/src**: obsahuje zdrojový kód doporučovací aplikace.
- **/tests**: obsahuje jednotkové testy, které testují některé funkce aplikace, zejména datové struktury.
- **/alembic**: obsahuje soubory spojené s migrací schémat databáze.
- **/docs**: obsahuje soubory spojené s generováním vývojové dokumentace doporučovací aplikace.
- **/examples**: obsahuje příkladové soubory nastavení aplikace a příklady nastavení spustitelných experimentů.

Adresář webového rozhraní je rozdělen do podadresářů.

- **/src**: obsahuje zdrojový kód webového rozhraní.
- **/static**: obsahuje statické data webového rozhraní. Například použité obrázky. item **/examples**: obsahuje příkladové soubory nastavení webového rozhraní.



Obrázek 4.14: Vytvoření hodnocení položky.

4.4.3 Instalace závislostí

Doporučovací aplikace a webové rozhraní jsou aplikace napsané v jazyce Python. Aplikace využívají ke svému běhu volně dostupné externí knihovny, jejichž verze a závislosti spravujeme pomocí nástroje Poetry. Během instalace Poetry vytváří pro každou aplikaci vlastní virtuální prostředí, ve kterém jsou dostupné všechny potřebné knihovny požadovaných verzí spolu s danou aplikací a jejím nastavením. Pokud jsme v kořenovém adresáři aplikace, můžeme její závislosti instalovat následujícím příkazem.

```
$ poetry config virtualenvs.in-project true && poetry install
```

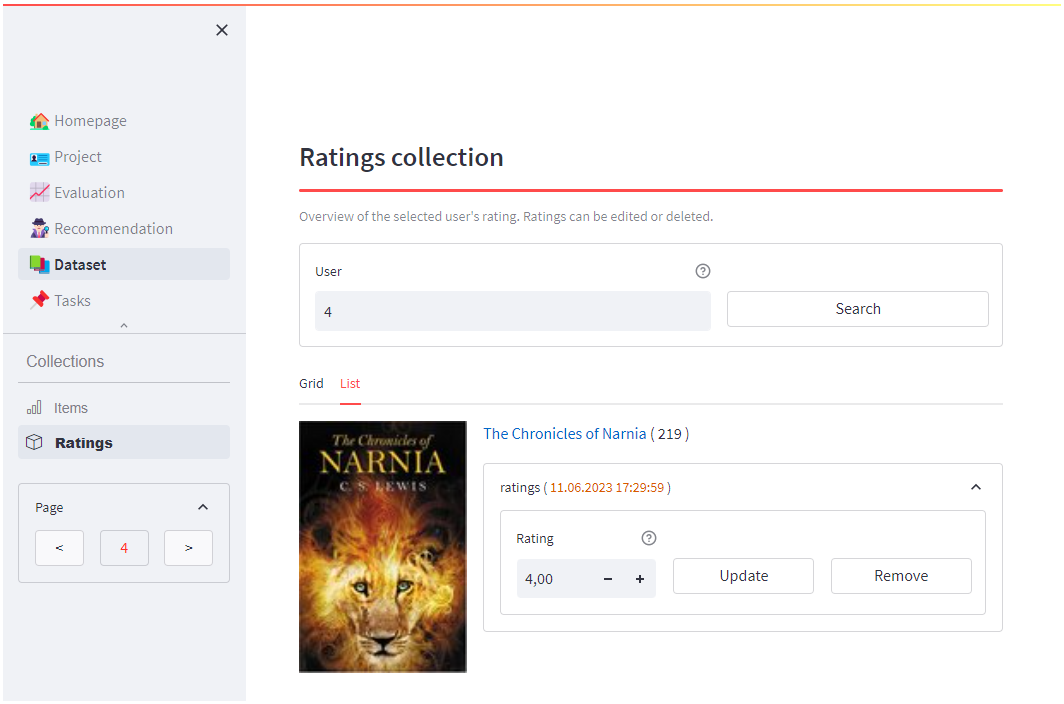
Krok instalace závislostí není nutný provádět manuálně, pokud bychom chtěli systém pouze spustit. Pro spuštění systému použijeme později v této sekci nástroj Docker, který instalaci provede za nás.

4.4.4 Generování vývojové dokumentace

Vývojová dokumentace je vytvořená pouze pro doporučovací aplikaci. Pokud máme nainstalované závislosti aplikace, můžeme dokumentaci vygenerovat spuštěním následujícího příkazu v kořenovém adresáři doporučovací aplikace.

```
$ make docs
```

Příkaz vygeneruje vývojovou dokumentaci pomocí nástroje [sphinx] ve formátu HTML do adresáře `./docs/build` v kořenovém adresáři aplikace.



Obrázek 4.15: Přehled hodnocení položek uživatelem.

4.4.5 Spuštění jednotkových testů

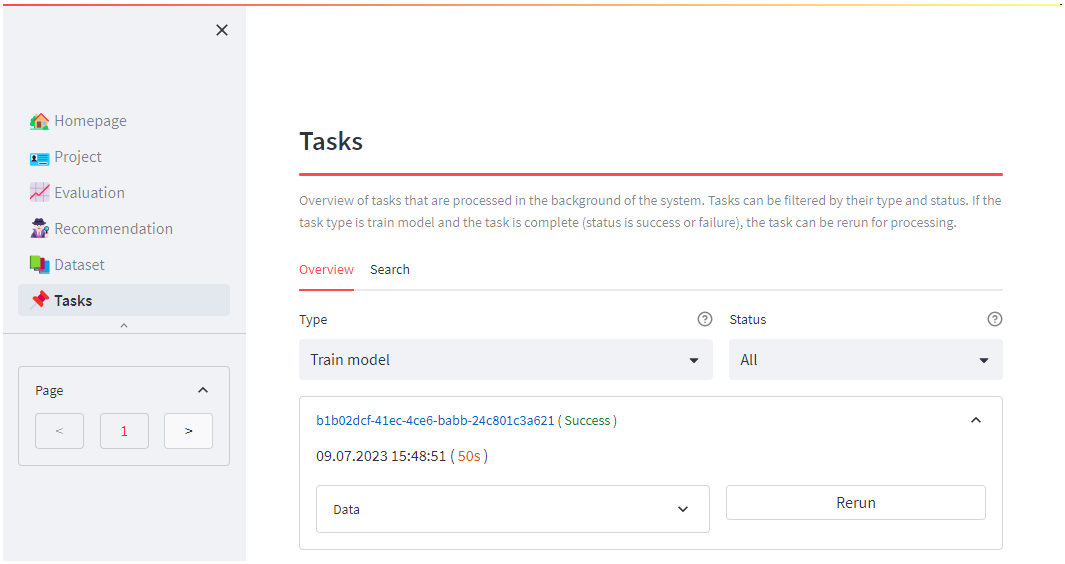
Testy jsou stejně jako vývojová dokumentace vytvořeny pouze pro doporučovací aplikaci. Jestliže máme pro doporučovací aplikaci nainstalované potřebné závislosti, můžeme jednotkové testy spustit následujícím příkazem v kořenovém adresáři doporučovací aplikace.

```
$ make tests
```

4.4.6 Nastavení systému

Nastavení doporučovací aplikace a webového rozhraní je realizované pomocí `.env` konfiguračních souborů, které jsou uloženy v kořenových adresářích obou aplikací. Konkrétně umístění konfiguračního souboru doporučovací aplikace je `{pwd}/backend/.env` a webového rozhraní je `{pwd}/frontend/.env`, kde `{pwd}` označuje souborový adresář ve kterém je projekt doporučovacího systému umístěn. Základní parametry nastavení doporučovací aplikace jsou následující.

- **SECRET_KEY**: parametr, který určuje tajný klíč doporučovací aplikace. Využívá se při generování přístupových tokenů pro autorizovány přístup k chráněným koncovým bodům REST API.
- **LOG_LEVEL**: parametr, který určuje úroveň logování. Výchozí hodnotou je informační úroveň.
- **ACCESS_TOKEN_EXPIRE_SECONDS**: parametr, jenž určuje počet vteřin od vygenerování přístupového tokenu, po které vyprší jeho platnost.



Obrázek 4.16: Přehled úloh, které se zpracovávají na pozadí.

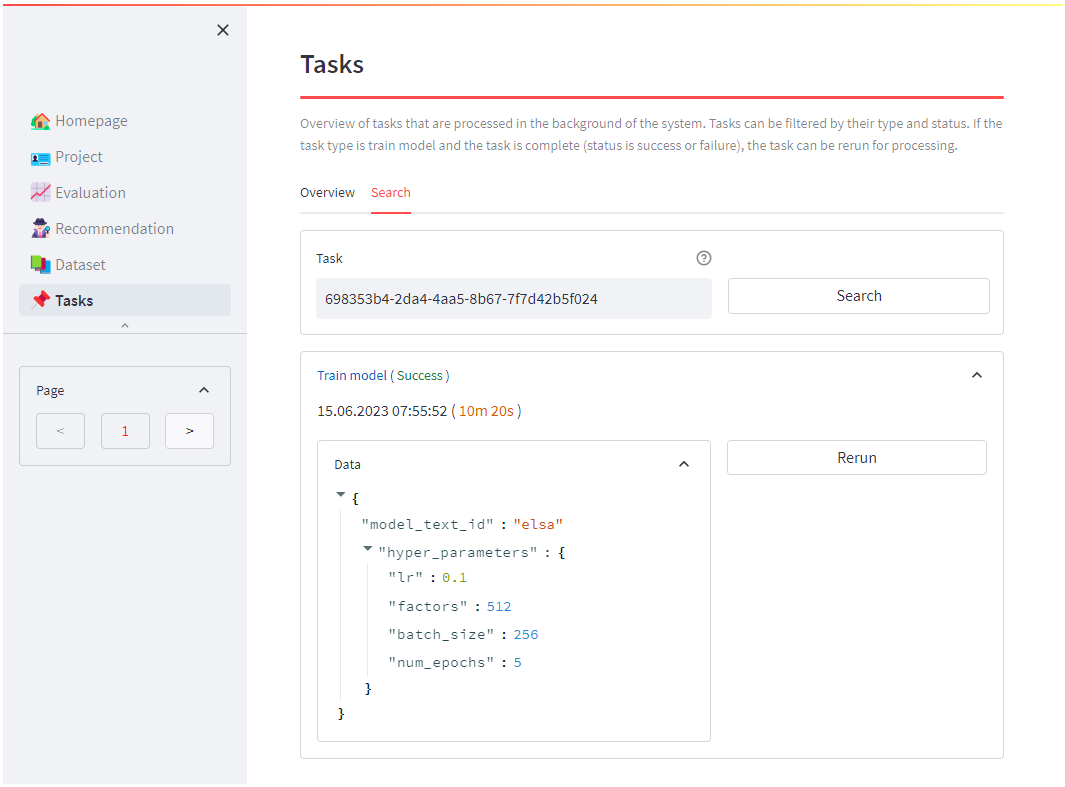
- **POSTGRES_HOST**: parametr adresy Postgres databáze.
- **POSTGRES_USER**: parametr uživatele databáze, pod kterým se doporučovací aplikace k databázi přihlašuje.
- **POSTGRES_PASSWORD**: parametr uživatelského hesla do databáze.
- **POSTGRES_DATABASE**: parametr databáze, do které doporučovací aplikace ukládá data.
- **MONGO_HOST**: parametr adresy Mongo databáze.
- **MONGO_USERNAME**: parametr jména uživatele, pod kterým doporučovací aplikace přistupuje do databáze.
- **MONGO_PASSWORD**: parametr uživatelského hesla.

V adresáři `./examples` doporučovací aplikace se nachází příkladový konfigurační soubor `.env.example`, který obsahuje nastavení, jenž koresponduje s výchozím nastavením zbylých částí systému. Například nastavení přístupových údajů obou databází. Pro rychlé spuštěni doporučovacího sytému je tudíž možné tento soubor adekvátně přejmenovat a použít v kořenovém adresáři doporučovací aplikace. Zbylé konfigurační parametry můžeme nalézt ve vývojové dokumentaci doporučovací aplikace.

Nastavení webového rozhraní má pouze jeden konfigurační parametr **BACKEND_HOST**, jenž udává URL adresu, na které REST API rozhraní doporučovací aplikace zpracovává příchozí požadavky. V adresáři `./examples` webového rozhraní je opět příkladový konfigurační soubor `.env.example`, jenž je možné použít.

4.4.7 WSGI

WSGI server nastavujeme pomocí souboru `gunicorn.conf.py` v kořenovém adresáři doporučovací aplikace. Základní parametry jsou následující.



Obrázek 4.17: Vyhledání konkrétní úlohy.

- **bind**: adresa a port, na které naslouchá. Číslo portu se odděluje od adresy dvojtečkou.
- **worker_class**: třída procesu, jenž zpracovává požadavky. V našem případě využíváme ASGI třídu `uvicorn.workers.UvicornWorker`.
- **workers**: počet paralelně spuštěných procesů.
- **timeout**: procesy, jenž zpracovávají požadavky déle než určený limit, jsou ukončeny a restartováni.

Podrobný popis všech parametrů můžeme nalézt v dokumentaci Gunicorn[9]. V adresáři `./examples` doporučovací aplikace můžeme opět nalézt příkladový soubor s nastavením WSGI serveru `gunicorn.conf.example`, jenž je možné použít.

4.4.8 Spuštění systému

Pro spuštění doporučovacího systému využijeme nástroj Docker. Docker je kontejnerizační nástroj, který umožňuje izolaci a správu aplikací a jejich závislostí. Nástroj poskytuje dva základních principy obraz a kontejner. Docker obraz je spustitelný balíček, který obsahuje všechny potřebné soubory a závislosti pro spuštění aplikace. Obraz je definován pomocí souboru s názvem **Dockerfile**, který popisuje kroky vytvoření obrazu. Těmi jsou například určení výchozího obrazu, kopírování zdrojových souborů aplikace, instalace závislostí aplikace a její spuštění. Dalším principem nástroje je Docker kontejner. Kontejner je instance

[9] `https://docs.gunicorn.org/en/latest/settings.html`

spuštěného Docker obrazu. Poskytuje izolované prostředí pro běh aplikace, které je oddělené od ostatních kontejnerů a hostitelského operačního systému.

Doporučovací systém se skládá z šesti izolovaných kontejnerů, jenž se nachází ve stejné síti pro zajištění vzájemné komunikace.

- **postgres**: kontejner, ve kterém běží Postgres databáze s verzí 15.2. Obraz kontejneru je volně dostupný z Docker repositáře a databáze naslouchá na výchozím portu 5432. Kontejneru se nastavují parametry určující uživatelské jméno a heslo pro přístup do databáze spolu s jejím názvem.

- **mongodb**: kontejner ve kterém běží Mongo databáze s verzí 5.0.17. Obraz kontejneru je opět volně dostupný z Docker repositáře. Mongo databáze naslouchá na výchozím portu 27017. Kontejneru se nastavují parametry určující uživatelské jméno a heslo pro přístup do databáze.

- **rabbitmq**: kontejner, ve kterém běží zprostředkovatel zpráv pro zpracování úloh na pozadí systému. Zprostředkovatel je realizovaný pomocí RabbitMQ[10], který naslouchá na výchozím portu 5672. Obraz zprostředkovatele je volně dostupný z Docker repositáře. Kontejneru se nastavují základní parametry pro přístup ke zprostředkovateli.

- **backend**: kontejner doporučovací aplikace. Obraz aplikace se vytváří ze souboru `Dockerfile` v jejím kořenovém adresáři. Výchozím obrazem je `python:3.10-bullseye`. Kontejner závisí na předchozích kontejnerech. Závislosti určují pořadí spuštění kontejnerů.

- **frontend**: kontejner webového rozhraní doporučovacího systému. Obraz se vytváří ze souboru `Dockerfile` v kořenovém adresáři webového rozhraní. Výchozím obrazem je `python:3.10-slim`. Kontejner závisí na kontejneru doporučovací aplikace.

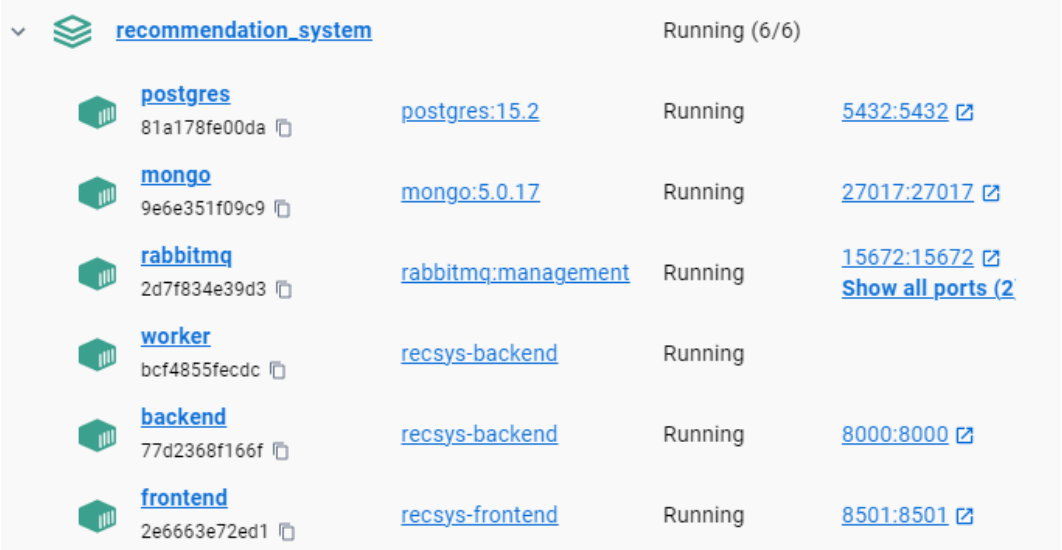
- **worker**: kontejner vedlejšího procesu doporučovací aplikace, jenž zpracovává úlohy na pozadí. Kontejner má stejný obraz a závislosti jako kontejner doporučovací aplikace. Odlišné je pouze jeho spuštění.

Jednotlivé kontejnery jsou definované v souboru `docker-compose.yml` v kořenovém adresáři projektu doporučovacího systému. V souboru jsou také k jednotlivým kontejnerům přiřazeny výchozí parametry nastavení, jenž je možné upravit. Pro spuštění doporučovacího systému použijeme následující příkaz v kořenovém adresáři projektu doporučovacího systému.

```
$ make build && make up
```

Příkaz postupně vytvoří všechny kontejnery. V případě kontejnerů **backend**, **worker** a **frontend** také vytvoří určené obrazy aplikací. Zbylé obrazy databází a zprostředkovatele stáhne z Docker repositáře. Vytvořené kontejnery následně spustí v pořadí určeném podle daných závislostí. Přehled běžících kontejnerů můžeme vidět na obrázku 4.18.

[10] `https://www.rabbitmq.com/`



Obrázek 4.18: Přehled běžících kontejnerů v nástroji Docker.

4.4.9 Použití systému

Máme-li vytvořené a spuštěné všechny kontejnery, můžeme začít používat doporučovací systém pomocí implementovaných komunikačních rozhraní.

- **http://localhost:8000/api/v1**: REST API rozhraní doporučovací aplikace.
- **http://localhost:8000/api/v1/redoc**: dokumentace restového rozhraní, vytvořená z openapi specifikace pomocí nástroje [ReDoc].
- **http://localhost:8000/sphinx**: vývojová dokumentace aplikace, je-li vygenerovaná.
- **http://localhost:8501**: webové rozhraní doporučovacího systému.

Chceme-li použít rozhraní příkazové řádky, musíme se připojit do běžícího kontejneru doporučovací aplikace (backend). Toho můžeme dosáhnout pomocí následujícího příkazu v kořenovém adresáři doporučovací aplikace.

```
$ make app-cli
```

V bežícím kontejneru doporučovací aplikace můžeme následně používat rozhraní příkazové řádky, tak je uvedeno v sekci 4.2. Příkazy rozhraní musíme spouštět v běhovém prostředí doporučovací aplikace, o které se stará nástroj Poetry.

```
$ poetry run recs COMMAND [OPTIONS] ARGUMENT
```

5. Experiment

Jedním z cílů naší práce je vyhodnotit použitelnost moderních metod pro analýzu textu pro doporučování založeném na obsahu, a to jak samostatně, tak i kombinovaně s metodami kolaborativního filtrování. V této kapitole si popíšeme průběh experimentu, jenž vyhodnotí výkonost doporučování daných metod a ze získaných výsledků odvodíme závěr.

5.1 Popis průběhu

V rámci tohoto experimentu budeme vyhodnocovat výkonost doporučování pomocí offline testování různých modelů využívající různé paradigmaty. Modely, které budeme vyhodnocovat jsou **ALS**, a **BPR** varianty faktorizace matic, dále **ELSA**, **BERT** a **Monolith**, jenž reprezentuje hybridní princip doporučování. Každý z modelů byl vhodným způsobem optimalizován pro výběr hyper-parametrů.

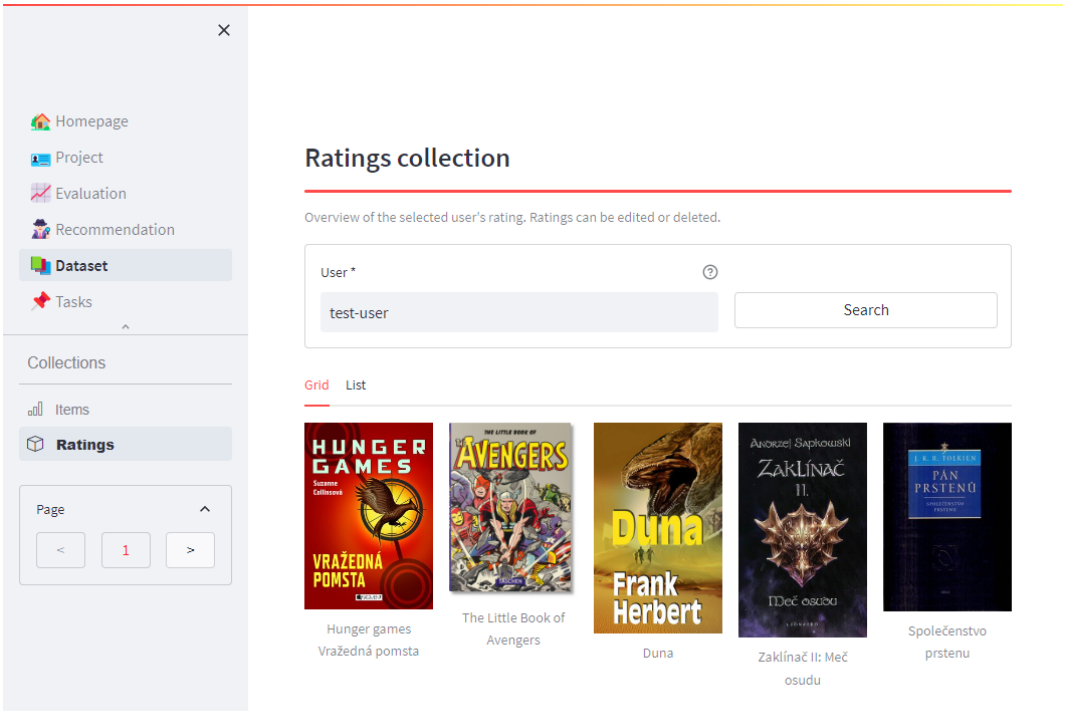
Experiment proběhne na dostupné datové sadě, kterou jsme popsali v kapitole 2.2.2. Datová sada obsahuje kolekce interakcí a položek, a pro účely vyhodnocení modelů bude rozdělena na trénovací a testovací část v poměru 80 : 20. Datovou sadu před jejím rozdělením ještě upravíme, aby platilo, že každý zákazník bude mít nejméně 4 záznamy v kolekci interakcí. Tuto úpravu provádíme z důvodu, adekvátního rozdělení záznamů interakcí zákazníka do trénovací a testovací části datové sady. Testovací část bude obsahovat posledních 20% záznamů každého zákazníka a trénovací část zbytek. Upravená datová sada obsahuje 2293194 záznamů interakcí od 138002 zákazníků. Počet knížek v datové sadě je 78766. Na trénovací části se určené modely natrénují a na testovací části se vyhodnotí výkonost jejich doporučování pomocí zvolených metrik. Zvolené vyhodnocovací metriky jsou následující:

- **Mean Average Precision (MAP)**[1]
- **Normalized Discounted cumulative gain (NDCG)**[2]
- **Mean reciprocal rank (MRR)**[3]
- **Hit rate (HR)**: udává procentuálně množství zákazníků, kterým byl doporučený jimi preferovaný produkt.
- **Novelty**: metrika implementovaná podle definice z článku (Zhang, 2013).
- **Coverage**: udává kolik procent produktů z datové sady bylo doporučeno.

5.2 Výsledky

Výsledky experimentu můžeme nalézt v tabulce 5.1. Na ose `y` jsou názvy modelů, jenž byly vyhodnocovány a na ose `x` jsou zvolené metriky. Metriky byly

[1] `https://en.wikipedia.org/wiki/Evaluation_measures_(information_retrieval)`
[2] `https://en.wikipedia.org/wiki/Discounted_cumulative_gain`
[3] `https://en.wikipedia.org/wiki/Mean_reciprocal_rank`



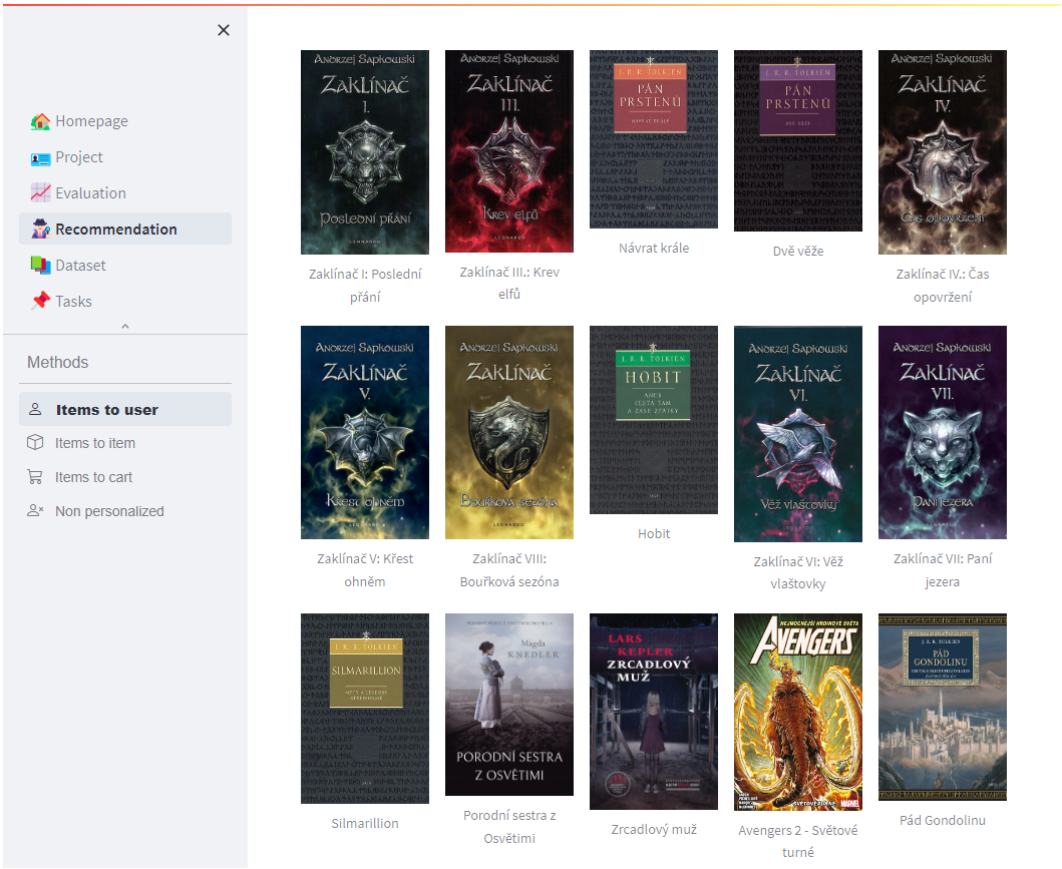
Obrázek 5.1: Hodnocení knížek testovacím uživatelem.

vyhodnoceny na seznamu 20 doporučených produktů. Z výsledků můžeme vidět, že modely kolaborativního filtrování jsou výrazně lepší v metrikách týkající se relevance doporučení. Modelem, který vychází z experimentů nejlépe, je varianta metody faktorizace matic **ALS**. Avšak výměnou za relevantnější doporučení jsou nižší hodnoty metrik **Coverage** a **Novelty**. Ty určitým způsobem ovlivňují schopnost explorace během doporučování. Modelem, jenž v nich vyniká je **BERT** využívající content-based princip.

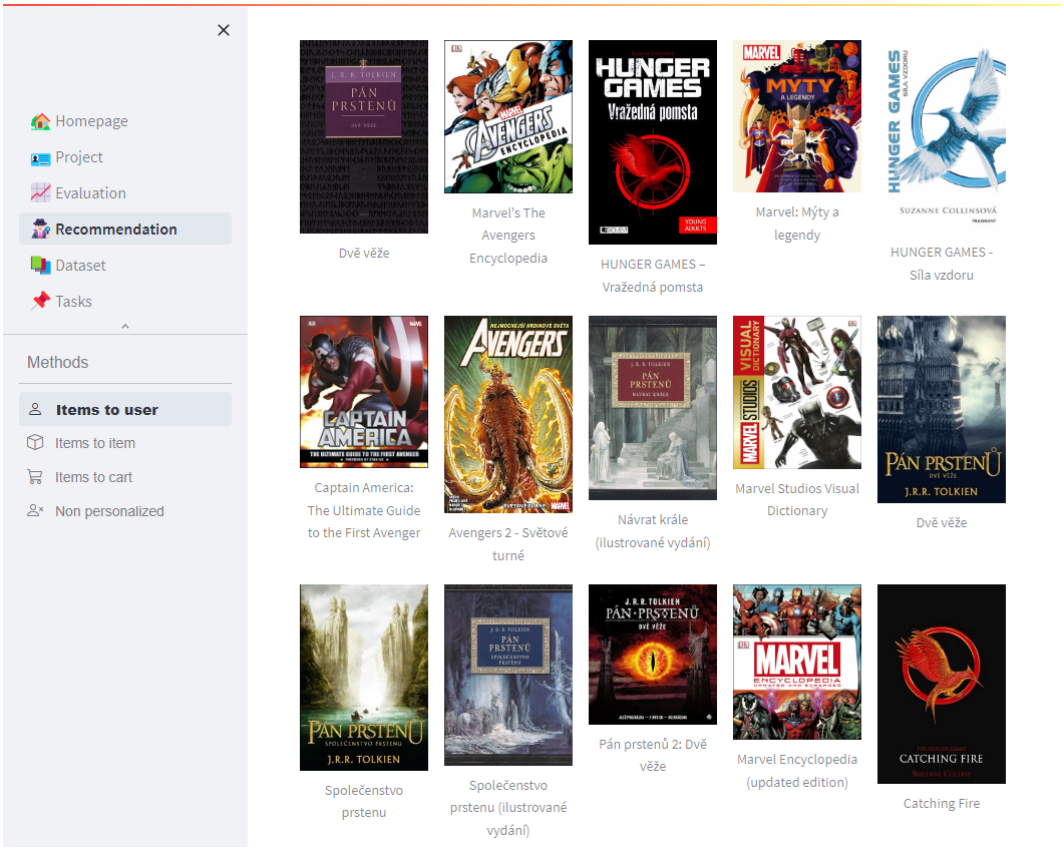
Hybridní a content-based modely se obecně podle offline experimentu zdají být horší, ale podívejme se na konkrétní příklady. Na obrázku 5.1 můžeme vidět hodnocení knížek zákazníkem, pro kterého vytvoříme doporučení pomocí **ALS** a **BERT** modelů. Doporučení produktů 5.2 5.3 se nezdají být příliš odlišné a s jistotou můžeme říci, že jsou pro daného zákazníka relevantní. Model **BERT** je o něco více explorativní a důvodem nepřesného doporučení může být větší množství skoro identických produktů (druhé vydání apod.).

	MAP@20	NDCG@20	HR@20	MRR@20	Coverage	Novelty
ALS	**0.0723**	**0.1142**	**0.3388**	**0.1304**	0.1550	6.4391
BPR	0.0465	0.0747	0.2314	0.0856	0.3414	7.8226
ELSA	0.0659	0.1056	0.3178	0.1192	0.2584	6.6610
BERT	0.0111	0.0204	0.0775	0.0211	**0.5646**	**9.3799**
Monolith	0.0244	0.0463	0.1900	0.0594	0.2982	4.9796

Tabulka 5.1: Výsledky vyhodnocení experimentu.



Obrázek 5.2: Doporučení knížek kolaborativní metodou ALS pro testovacího uživatele.



Obrázek 5.3: Doporučení knížek content-based metodou BERT pro testovacího uživatele.

Závěr

Vytvořili jsme doporučovací systém, který je možné použít jako kompletní řešení pro e-commerce společnosti, jenž chtějí začít personalizovat svoji nabídku produktů. Doporučovací systém podporuje různé doporučovací metody různých paradigmat, které poskytují různorodé doporučení nejen pro doménu knížek.

Systém slouží jako webová služba s architekturou RaaS. Poskytuje rozhraní REST pro vnější komunikaci a také interaktivní webové rozhraní pro snazší administraci doporučování, monitorování výkonosti a demonstraci doporučování. Umožňuje využít metody faktorizace matic a lineární autoencoder pro doporučení založeném na kolaborativním filtrování. Dále také umožňuje využít moderní přístupy ke CB analýze obsahu pomocí modelu BERT. Nakonec umožňuje tyto principy kombinovat pomocí hybridních přístupů. Systém podporuje kompletní životní cyklus doporučovacích modelů od jejich návrhu a integrace pomocí principu zásuvných modulů, až po jejich nasazení do ostrého provozu, verzování a již zmíněné monitorování.

Systém jsme nakonec vyhodnotili na konkrétní datové sadě středně velké e-commerce společnosti, jenž se zabývá prodejem knih. Z experimentu vyšly výrazně lépe metody kolaborativního doporučování, které dokáží přesněji odhadnout preference zákazníků. Na druhou stranu metody content-based a hybridní přístupy dokáží během doporučování lépe explorovat, což zvyšuje jejich schopnost uplatnění v systému. Horší výsledky metrik relevancí content-based metod mohly být způsobené častým výskytem příliš podobných produktů v použité datové sadě. Tomu lze předejít například použitím metod clustrování při zpracování datových sad.

Seznam použité literatury

Aggarwal, C. C. (2016). *Recommender Systems: The Textbook*. Springer, Cham. ISBN 978-3-319-29657-9. doi: 10.1007/978-3-319-29659-3.

Ahuja, R., Solanki, A. a Nayyar, A. (2019). Movie recommender system using k-means clustering and k-nearest neighbor. In *2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, pages 263–268. doi: 10.1109/CONFLUENCE.2019.8776969.

Burke, R. (2002). Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, **12**. doi: 10.1023/A:1021240730564.

Carbonell, J. a Goldstein, J. (1998). The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '98, page 335–336, New York, NY, USA, 1998. Association for Computing Machinery. ISBN 1581130155. doi: 10.1145/290941.291025. URL `https://doi.org/10.1145/290941.291025`.

Chiny, M., Chihab, M., Bencharef, O. a Chihab, Y. (2021). Netflix recommendation system based on TF-IDF and cosine similarity algorithms. In *Proceedings of the 2nd International Conference on Big Data, Modelling and Machine Learning*. SCITEPRESS - Science and Technology Publications.

Devlin, J., Chang, M., Lee, K. a Toutanova, K. (2018). BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, **abs/1810.04805**. URL `http://arxiv.org/abs/1810.04805`.

Fleder, D. M. a Hosanagar, K. (2007). Recommender systems and their impact on sales diversity. In MacKie-Mason, J. K., Parkes, D. C. a Resnick, P., editors, *Proceedings 8th ACM Conference on Electronic Commerce (EC-2007), San Diego, California, USA, June 11-15, 2007*, pages 192–199. ACM. doi: 10.1145/1250910.1250939. URL `https://doi.org/10.1145/1250910.1250939`.

He, R. a McAuley, J. (2015). Vbpr: Visual bayesian personalized ranking from implicit feedback.

Hossain, M., Sattar, A. H. M. S. a Paul, M. K. (2019). Market basket analysis using apriori and fp growth algorithm. In *2019 22nd International Conference on Computer and Information Technology (ICCIT)*, pages 1–6. doi: 10.1109/ICCIT48885.2019.9038197.

Pilászy, I., Zibriczky, D. a Tikk, D. (2010). Fast als-based matrix factorization for explicit and implicit feedback datasets. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, page 71–78, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781605589060. doi: 10.1145/1864708.1864726. URL `https://doi.org/10.1145/1864708.1864726`.

Reimers, N. a Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. URL https://arxiv.org/abs/1908.10084.

Rendle, S., Freudenthaler, C., Gantner, Z. a Schmidt-Thieme, L. (2012). Bpr: Bayesian personalized ranking from implicit feedback.

Schlegel, M. a Sattler, K.-U. (2023). Management of machine learning lifecycle artifacts: A survey. *SIGMOD Rec.*, **51**(4), 18–35. ISSN 0163-5808. doi: 10.1145/3582302.3582306. URL https://doi.org/10.1145/3582302.3582306.

Steck, H. (2019). Embarrassingly shallow autoencoders for sparse data. *CoRR*, **abs/1905.03375**. URL http://arxiv.org/abs/1905.03375.

Vančura, V., Alves, R., Kasalický, P. a Kordík, P. (2022). Scalable linear shallow autoencoder for collaborative filtering. In *Proceedings of the 16th ACM Conference on Recommender Systems*, RecSys '22, page 604–609, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392785. doi: 10.1145/3523227.3551482. URL https://doi.org/10.1145/3523227.3551482.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. a Polosukhin, I. (2017). Attention is all you need. *CoRR*, **abs/1706.03762**. URL http://arxiv.org/abs/1706.03762.

Zhang, L. (2013). The definition of novelty in recommendation system. *Journal of Engineering Science and Technology Review*, **6**, 141–145. doi: 10.25103/jestr.063.25.

Seznam obrázků

A. Přílohy

A.1 Elektronická příloha práce

V rámci přílohy odevzdáváme primárně zdrojový kód aplikace, openapi specifikaci REST rozhraní, webový scrapper a konfigurační soubory pro použitý experiment. Adresářová struktura zdrojového kódu je následující.

- **/recommendation-system**
  - **/backend**: obsahuje zdrojový kód, příklady nastavení a sadu testů doporučovací aplikace systému.
  - **/frontend**: obsahuje zdrojový kód a příklad nastavení webového rozhraní systému.
- **/experiments**: obsahuje konfigurační soubory experimentů pro vyhodnocení doporučovacích modelů.

V rámci každého adresáře existuje soubor **README**, který poté umožní další práci. Mimo jiné zdrojový kód aplikace lze také najít v repositáři `https://gitlab.mff.cuni.cz/vargaon/recommendation-system`.