FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Kolaborativní filtrování pro portál LearnerOn – případová studie
Student:	Bc. Filip Sikora
Vedoucí:	Ing. Michal Valenta, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2023/2024

Pokyny pro vypracování

Portál LearnerOn používá částečně grafovou databázi Neo4j. Cílem této práce je analyzovat, navrhnout a formou funkčního prototypu ověřit doporučovací modul pro tento portál.
Postupujte dle níže uvedených kroků:
1. Navažte na svou bakalářskou práci a popište stávající datovou strukturu v LearnerOn, která je pro doporučování relevantní.
2. Nastudujte problematiku kolaborativního filtrování, zaměřte se na metody, které jsou vhodně aplikovatelné na grafové databáze, okrajově zmiňte alternativy.
3. Na základě studie z předchozího bodu a na základě specifičnosti domény LearnerOn (online vzdělávání) diskutujte a zvolte vhodné metody.
4. Návrh implementujte formou funkčního prototypu.
5. Na vhodných datech ověřte funkčnost. Diskutujte výsledky a navrhněte vhodné postupy pro produkční řešení.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 13. prosince 2022 v Praze.

Diplomová práce

KOLABORATIVNÍ FILTROVÁNÍ PRO PORTÁL LEARNERON – PŘÍPADOVÁ STUDIE

Filip Sikora

Fakulta informačních technologií ČVUT v Praze
Katedra softwarového inženýrství
Vedoucí: Ing. Michal Valenta Ph.D.
4. května 2023

Obsah

Seznam obrázků

Touto práci chci poděkovat za spolupráci firmě LearnerOn a všem jejim členům ať už současným nebo bývalým, se kterým jsem spolupracoval během vytváření této práce, především pak vedoucímu projektu Vladislavovi Severovi a vedoucímu této diplomové práce Ing. Michalu Valentovi, Ph.D. za jeho pomoc na projektu, kdo se semnou podílel i na předchozí práci na tomto projektu, přinesl velice ceněný vhled do datové části praktické části a pomoc při editaci výsledné práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 20. dubna 2023 .....

Abstrakt

Tato diplomová práce se zabývá analýzou, návrhem a tvorbou více-vrstvého doporučovacího modulu pro vzdělávací portál, jejíž hlavní část tvoří kolaborativní filtrování. To je navrženo a použito specificky pro potřeby portálu. Modul je založen na efektivním využití implementace grafové databáze Neo4J. Modul také používá vrstvy, které se opírají o datovou základnu relační databáze MySQL, která je základem databáze Neo4J.

Klíčová slova doporučovací systém, kolaborativní filtrování, více-vrstvé doporučování, grafová databáze, vzdělávácí sektor, Neo4J

Abstract

This master's thesis deals with the analysis, design and creation of a multi-layer recommendation module for an educational portal, the main part of which is collaborative filtering. It is designed and used specifically for the needs of the portal. The module is based on the efficient use of Neo4J graph database implementation. The module also uses layers that rely on the MySQL relational database that underlies the Neo4J database.

Keywords recommender system, collaborative filtering, multi-layer recommendation, graph database, education sector, Neo4J

Seznam zkratek

BE	Backend
CF	kolaborativní filtrování
EA	Enterprise Architect
NACE	Klasifikace ekonomických činností vydávanou Evropskou komisí

Kapitola 1
Úvod

Doporučování je dnes již běžnou funkcionalitou u většiny služeb. Ať už je to individuální doporučení nebo ne. Velcí poskytovatelé služeb jako Netflix, Youtube atd. dnes poskytují tuto možnost, která umožňuje najít zákazníkovi rychleji relevantní obsah, a tak i benefitovat poskytovateli obsahu. Je naprosto jedno zda-li se jedná o doporučování přátel na sociálních sítích, nebo další video připravené do fronty sledování. Fantazie je limit.

Tato práce navazuje na mou bakalářskou práci, která se zaměřovala na vybudování datové báze pro takové doporučování spolu s kontinuální údržbou aktuálních dat v takovéto databázi. Nebude to proto výjimečné, když budu referovat na svou starou práci. Naopak sekce teoretické části se bude věnovat pouze ji.

Datová základna byla vybudována pro společnost LearnerOn, v době psaní start-up společnost, která má za cíl zpřístupnit profesní rozvoj, jak pro jednotlivce, tak pro firmy vyžadující takovou službu.

Mou motivací pro pokračování na tomto projektu byla, nejen má trvající spolupráce s firmou LearnerOn, ale také osobní víra ve veřejný přínos tohoto projektu, který může lidem pomoci s potřebou a chutí se naučit něco nového.

Ne méně teď než kdykoliv jindy lze vnímat ze strany veřejnosti jistou obavu o budoucnost, ať už by se referovalo na proběhlou pandemii, její hospodářské následky, nebo válku. Tento projekt má potenciál pomoct světu, alespoň v jedné stránce věci a to se vzděláním.

Hlavní část implementace kolaborativního filtrování je provedena na straně grafové databáze Neo4J. Tato databáze byla v minulosti zvolena pro vyzrálost, vhodnost pro doporučovací úlohy a strukturu vyhovující ukládaným datům. Ostatní úrovně doporučování jsou pak popsány v praktické části, kde se vysvětluje, jak fungují a důvod jejich použití.

Je důležité hned na úvod poznamenat, že od mé předešlé práce se struktura dat jemně pozměnila, aby vyhovovala firemním představám. Protože tato grafová databáze nemá pevnou definic schéma, změny nebude možné jednoduše představit ve formě definice schéma, místo toho tyto změny uvedu slovně a to v teoretické části.

V první části popíšu teorii k pochopení praktické části. Bude v ní popsána firma LearnerOn, předešlá práce na kterou tato navazuje, změny provedeny po vypracování předešlé práce a na konec důvod vybrání a princip fungování kolaborativního filtrování.

V druhé části popíšu realizované více vrstvé řešení doporučovacího systému pro firmu LearnerOn.

Kapitola 2
Cíl práce

Cílemi této práce jsou analyzovat, navrhnout a vytvořit funkční prototyp doporučovacího modulu pro vzdělávací portál LearnerOn. Je také potřeba uvést problematiku na teoretické úrovni k pochopení vytvořeného doporučovacího mechanismu. Analytická část se také zabývá pochopením konkrétních potřeb firmy LearnerOn a jak chce doporučování využívat. Návrh doporučovacího mechanizmu diskutuje nad alternativami typů doporučování a dopodrobna popisuje ty, které byly vybrány portálem k implementaci. Část, která je o tvorbě prototypu řešení se pak zabývá implementací v konkrétním systému.

Kapitola 3

Teoretická část

V této velice důležité kapitole přiblížím použitou teorii potřebnou k pochopení výstupu z praktické části mé diplomové práce.

První sekce popisuje firmu LearnerOn a projekt, na kterém spolupracovala se ČVUT, do jehož řešení jsem byl zapojen. Při něm vznikla má bakalářská práce, která dala za základ této diplomové práci. Poté teoreticky popíšu hlavní část této práce, a to kolaborativní filtrování, spolu s ostatními typy filtrování, výhodami i nevýhodami tohoto typu doporučování. Další sekce představuje pojem grafové databáze a jednu z jejich konkrétní implementaci - Neo4J, která byla použita pro implementaci kolaborativního filtrování.

3.1 O projektu LearnerOn

Přestože jsem tuto firmu již představil ve své bakalářské práci [1], rád bych z ní vybral a zopakoval pár nejdůležitějších bodů okolo této firmy a společného projektu s ČVUT.

„Firma LearnerOn, SE podniká dle NACE kategorizace v podpůrné činnosti ve vzdělávání. Byla založena v roce 2017. Hlavní podnikatelskou aktivitou společnosti je poskytovat zájemcům o celoživotní vzdělávání návodné a personalizované vedení při volbě cesty vzdělání (tzv. Learning Path) a v pozdější fázi i cílů vzdělání (Learning Goal), a to v online prostředí.“[1]

NACE stojí pro klasifikaci ekonomických činností vydávanou Evropskou komisí. V podstatě je to systém pro charakterizaci, v jakém odvětví firma pracuje. Kromě doplňujícího popisu k předchozímu odstavci, je dalším důvodem, proč to zmiňuji, ten, že tato charakterizace hrála velkou roli v mé bakalářské práci [1] jako prostředek pro shlukování firem/ zákazníku se stejným typem produkce.

S touto firmou jsem začal spolupracovat při společném projektu firmy LearnerOn, SE a Fakulty informačních technologií ČVUT, kde financování spolupráce bylo podpořeno Pražským voucherem na inovační projekty ve výzvě s číslem tři. Předmětem řešení byl *Pokročilé inovativní datové struktury v podpoře celoživotního vzdělávání.* Tato žádost byla vytvořena podnikatelským záměrem předaném hlavnímu městu Praha. Kde je mimojiné popisováná konkurenceschopnost, tržní potenciál i jeho příspěvek společnosti. [1]

Jedním z cílů této spolupráce byl i datový model a konverze částečných datových modelů do požadovaných, na kterém jsem se do velké míry podílel. To byl mimojiné předmět mé bakalářské práce[1] a je základem pro tuto práci. Vytvoření datové řešení totiž mělo sloužit jako základ pro doporučovací systém vzdělávacích materiálů pro uživatele.

Dalšími cíli této spolupráce byly:

1. Datová ontologie pro definici a modelování souvislostí a propojení datových entit
2. Konkrétní vzorová implementace výsledného datového modelu
3. Testování algoritmické funkcionality
4. Vzorová / testovací implementace. Vzorové end-to-end propojení a implementace v rámci GRANDstack
5. Průběžné konzultace, manuály

GRANDstack je kombinace technologií pro vytvoření full-stack datově orietovaných aplikací, jejíž klíčovými komponenty jsou následující technologie: GraphQL, React, Apollo a Neo4j. Bližší popis je možné se dozvědět ze sekce *GRANDstack* v mé bakalářské práci [1]. Tyto body byly řešeny také mnou nebo ostatními spolupracovníky zastupující ČVUT v projektu, ale nejsou klíčové pro pochopení výsledků této práce, proto je zde nebudu příliš rozvádět.

3.1.1 O čem byla má bakalářská práce

Hlavním cílem mé bakalářské práce byla analýza, návrh a implementace datové části pro doporučovací systém. Součástí návrhu datové části byl také ontologický OntoUML model pro zachycení modelu a jeho jednoznačnou dokumentaci. Pro implementaci bylo nutné zpracovat již existující data s novými do potřebné formy vyhovující cílům dějícího se projektu. [1]

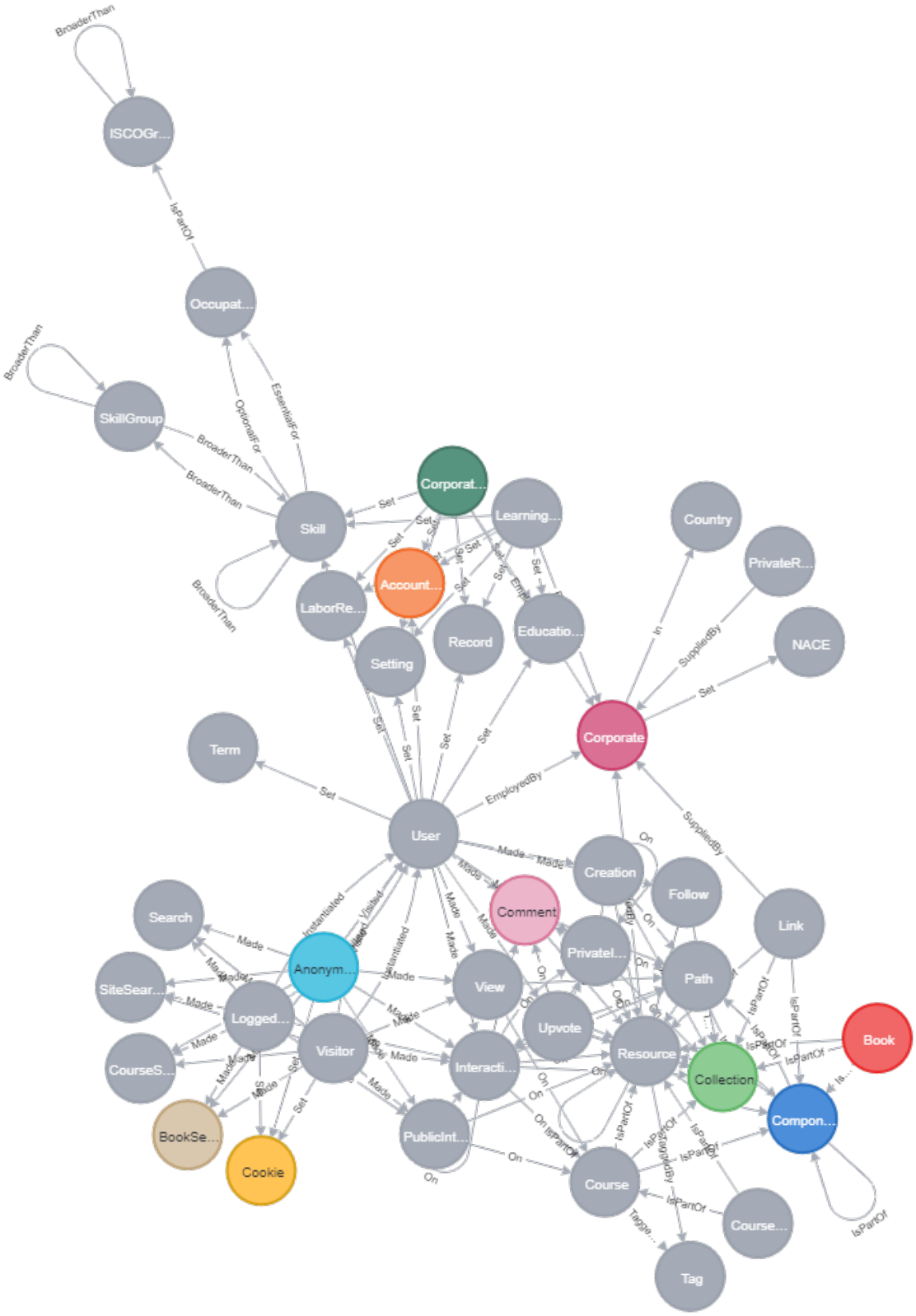
3.1.2 Výsledný datový model po dokončení spolupráce na projektu

Implementací navržené ontologie do grafové databáze Neo4j transformací dat ze stávající MySQL databáze do Neo4j, která se řídila sadou pravidel popsanou v mé bakalářské práci, jsme dostali databázový model, jak je ukázán na obrázku 3.1.

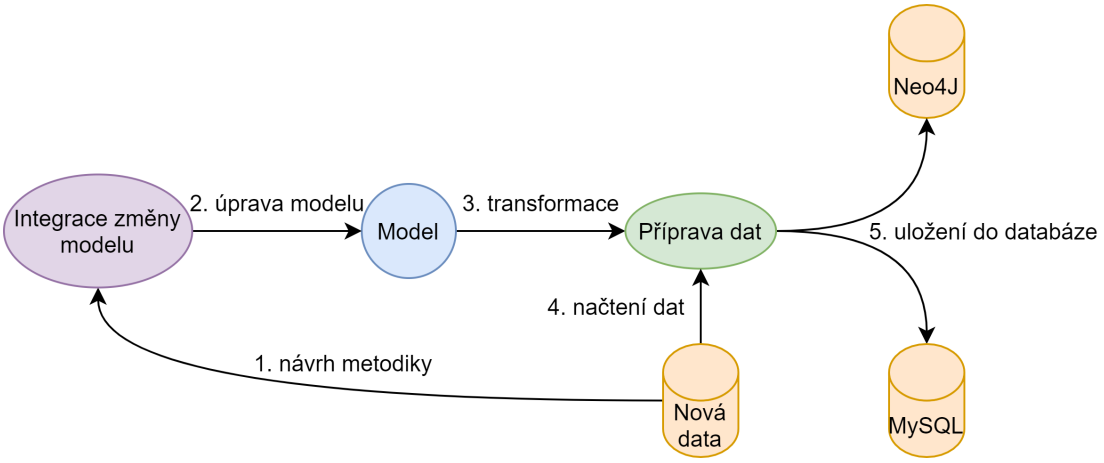
„Je nutné poznamenat, že pokud má uzel více štítků pak jsou vizualizovány všechny jeho štítky jako jiný uzel. Ten sdílí všechny společné atributy i vazby s jinými vizualizacemi stejného uzlu. Proto lze vidět na obrázku například uzly se štítkem Resource. V databázi se ovšem nevyskytují data pouze se štítkem Resource, ale ve skutečnosti konkrétní data jako knihy, nebo odkazy, které mají takový štítek pro označení učebního materiálu. Pro přehlednost pak databázový stroj při vizualizaci seskupuje všechny uzly se stejnými štítky.“ [1]

3.1.2.1 Metodika pro rozvoj databáze

Protože padlo rozhodnutí, že některá data budou (z bezpečnostních a jiných důvodů) udržována jak v novém úložišti, tak také v původní relační databázi. Vznikla potřeba pro údržbu aktuálnosti dat mezi oběma datovými zdroji. Návrh pro postup udržení aktuálnosti dat v obou databázích jsem vytvořil ve své bakalářské práci a jeho symbolické znázornění je viditelné na obrázku 3.2.



Obrázek 3.1 Celý Neo4J graf po seskupení hran se stejnými štítky



Obrázek 3.2 Integrace nových dat

Jak je tedy viditelné na obrázku 3.2 v případě, že je potřeba upravit ontologický model, nebo se přizpůsobit změně relačního modelu změnou vyvolanou vývojem, návrh je, že se bude postupovat následujícím způsobem.

1. „Prvně je potřeba dle charakteru dat navrhnout metodiku pro definování a zpracování dat,
2. poté aplikovat tyto metody do úpravy ontologického modelu.
3. V rámci transformace se připraví data dle skutečnosti, zda půjdou pouze do Neo4J databáze, nebo je potřeba je zálohovat i v relační MySQL databázi.
4. Nový zdroj dat se pak zpracuje transformací modelu
5. a data zbývá uložit.“ [1]

Samotnou implementaci popíšu až v praktické části této práce, i to totiž bylo potřeba dovyřešit k praktickému zprovoznění doporučování na webu LearnerOnu.

3.1.3 Použité technologie v aplikaci

V průběhu spolupráce taky došlo k architektonickému rozhodnutí a pokynu vedoucích projektu LearnerOn, že primární data zůstanou nadále v relačním databázovém stroji MySQL a pouze část bude převedena do Neo4j, takže některé datové entity jsou z výkonových a jiných důvodů replikovány v obou úložištích. Toto rozhodnutí padlo především kvůli některým z nevýhod Neo4J, které popíšu níže. Je tedy pochopitelné, že zvolená grafová databáze se používá k přínosu, pro který byla zvolena, místo toho aby nutně nahrazovala existující relační databázi, se kterou měl tým větší zkušenost.

Další důležitou technologií pro tuto práci je vyhledávač OpenSearch. OpenSearch, který je odnoží potenciálně známějších technologií jménem Elasticsearch a Kibana, je otevřená technologie skládající se především z datového úložiště a efektivního vyhledávače nad těmito daty. Kromě toho, že se používá tato technologie k podpoře vyhledávacích dotazů na stránce, tak se tento vyhledávač také používá k typu doporučení jménem sémantické doporučování, protože udržuje ve své databázi odlehčený seznam všech položek s jejich popisnými vektory a poskytuje možnost vyhledávání k nejbližších sousedů. Jak tento typ doporučování funguje, popíšu v praktické části podrobněji.

Obrázek 3.3 Proč potřebujeme doporučovací systémy

3.2 Proč potřebujeme doporučovací systémy

Na tuto otázku by sice lze šlo odpovědět jednoduchou odpovědí a bylo by tím ta hlavní motivace všeho v komerčním sektoru…peníze. Ale pokud se odprostíme od tohoto přímočarého pohledu, lze najít spoustu věcí s čím takový doporučovací systém může pomoct. Jak lze vidět na obrázku 3.3 mezi motivace k implementaci tohoto typu systému patří:

1. zlepšení uživatelského zážitku,
2. zlepšení udržení si zákazníků / uživatelovy věrnosti
3. prodej rozmanitějších položek
4. ušetření (uživatelova) času
5. lepší pochopení uživatetův potřeb
6. zvýšení pravděpodobnosti, že si uživatelé koupí doporučené zboží.
7. shlukovaní podobných uživatelů
8. a zlepšení příjmů společnosti.

Produkty lze navrhovat podle nejoblíbenějších položek dostupných na webových stránkách, demografického profilu zákazníka nebo jako odhad budoucích nákupních akcí zákazníka na základě jeho předchozích nákupů. [2] Způsobů, jak a co doporučit zákazníkovi je mnoho, v této práci popíšu některé z nich, jaké jsou jejich výhody, nevýhody a proč jsme zvolili, co jsme zvolili.

Funkce doporučení také pomáhá objevit jedinečné a zajímavé položky, které by jinak zůstaly nepovšimnuty, což přispívá ke spokojenosti uživatelů a poskytuje společnostem příležitost prodávat širší škálu produktů. [3]

Jedním z motivátorů, nebo možná vynucovačů implementace doporučování, je také skutečnost, že postupem času došlo k výraznému nárůstu množství údajů o chování uživatelů, které

zahrnují historické údaje o aktivitách na webu, jako jsou kliknutí, vyhledávání, zobrazení stránek a zobrazení položek, a také o aktivitách mimo web, jako je sledování kliknutí v e-mailech, mobilních aplikacích a push notifikací. Další údaje zahrnují historii nákupů zákazníků, hodnocení a lajky zboží a služeb, podrobnosti o jednotlivých položkách a připojení uživatelů na různých sociálních sítích. [2] To ve spojení s velkým množstvím doporučitelných položek vytváří perfektní kombinaci ke spárování uživatele se vhodným nabízeným zbožím.

3.3 Obecně k typům doporučovacích systémů

Algoritmy pro doporučovací systémy se využívají k tomu, aby zákazníkům poskytovaly obsah nebo produkty, které odpovídají jejich vkusu nebo výběru v minulosti.

Pokud jde o základní doporučovací systémy, existují čtyři hlavní kategorie:

1. filtrování na základě popularity,
2. filtrování založené na obsahu,
3. kolaborativní filtrování
4. a hybridní přístup.

Doporučovací systém založený na popularitě navrhuje uživateli nejoblíbenější položky, což je snadné implementovat a slouží jako dobrý základ pro ostatní modely. Tento přístup je vhodný pro uživatele bez historických dat, čehož v budoucnu využijeme. Algoritmus navrhuje seznam populárních položek, které byly např. hodnoceny nejvíce krát. Tato metoda může být někdy efektivnější než složité techniky kolaborativního filtrování. [4] Je nutné zmínit, že k dispozici je více základních algoritmů, o kterých se můžete dozvědět více v této publikaci [4], ty ale pro tuto práci nebudou podstatné.

Filtrování založené na obsahu se spoléhá na algoritmy strojového učení, které navrhují nové položky, jež jsou podobné těm, s nimiž uživatel již dříve komunikoval. Systém seskupuje podobné položky na základě jejich atributů, což je potenciální nevýhoda, kterou prozkoumám později, jelikož vymáhá důsledné porozumění těmto atributům. [5] Cílem technik založených na obsahu je rozpoznat společné rysy položek, které uživatel ocenil, jako jsou pozitivní hodnocení, nákupy nebo kliknutí, a navrhnout mu podobné položky. Tyto metody však vyžadují shromažďování informací, které nemusí být snadno dostupné, jednoduché na shromáždění nebo přímo relevantní. [3]

Hybridní přístup zahrnuje kombinaci více filtračních technik, které zvyšují přesnost a výkonnost. Nelze jej přesně definovat, bohužel se jedná o trochu vágnější definici.

Nakonec jsem si nechal pro nás ten nejdůležitější přístup…kolaborativní filtrování se při navrhování nových položek spoléhá výhradně na předchozí interakce mezi spotřebiteli a produkty, které využívali. Vlastnosti zboží nejsou relevantní, protože interakce mezi uživatelem a položkou se ukládají v matici interakcí mezi uživatelem a položkou. [5]

Protože na našem webu je k dispozici pestrá škála položek jako knih, linků, kurzů a jejich kolekcí (vzdělávacích cest), informace uvedené u jednotlivých položek není možné mít vždy pečlivě vyplněné. Především pokud mluvíme o obsahu vytvořeném uživateli, který nelze snadno kurátorovat. Protože některé položky mají pouze název a stručný, nebo i neužitečný popis, tak se váhy jednoznačně klaní na stranu kolaborativního filtrování jako prostředku k doporučování.

V tomto případě ani není možné vhodně použít přístup založený na obsahu, který byl zmíněn výše.

Ve spojení s relativní jednoduchostí fungování je kolaborativní filtrování také dobré řešení pro začátek, který je možné v budoucnu vylepšit, nebo nahradit na míru šitým modelem vytvořeným strojovým učením.

Takovéto byly tedy pohnutky k implementaci kolaborativního filtrování ve firmě LearnerOn a zvolení téma mé diplomové práce.

Hodnocení knih	Kniha 1	Kniha 2	Kniha 3	Kniha 4
Uživatel A	9			
Uživatel B	10		9	
Uživatel C		2		
Uživatel D		1		2

Obrázek 3.4 Matice interakcí mezi uživateli a položkami

3.4 Co je tedy kolaborativní filtrování

Kolaborativní filtrování bere v úvahu všechny uživatele a identifikuje ty, kteří sdílejí podobné záliby a zápory, aby bylo možné doporučit hlavnímu zákazníkovi nové a cílené produkty. Tato technika umožňuje podnikům i spotřebitelům sledovat nejnovější trendy. [5]

Rozeznáváme dva typy interakcí mezi uživateli a produkty:

1. První způsob zahrnuje implicitní zpětnou vazbu, jako je klikání na odkazy, procházení historie objednávek nebo přehrávání určitého obsahu.

2. Druhý způsob je založen na explicitní zpětné vazbě poskytnuté uživatelem, jako je hodnocení filmu na stupnici od 1 do 5 hvězdiček, lajkování nebo nelajkování videa nebo označení alba či playlistu jako oblíbeného.

My budeme pracovat především s prvním typem interakce, jelikož je na webu LearnerOnu plně implementován a existuje plno dat tohoto charakteru v existující databázi. Bude se od toho pak odvíjet použitá formule pro výpočet míry doporučení materiálu.

3.4.1 Co si představit pod maticí interakcí mezi uživateli a položkami

Protože maticí interakcí mezi uživateli a položkami (nadále matici interakcí) budu vícekrát zmiňovat, představím zde jednoduchým způsobem, co je jí myšleno, a jaké jsou myšlenkové postupy za nalezením podobné smýšlejících uživatelů.

Na ilustrativním obrázku 3.4, kde čísla od jedné do deseti značí oblíbenost dané položky od nejméně až nejvíce, lze vyčíst následující skutečnosti:

1. Uživatel A i uživatel B mají rádi knihu 1, mají možná podobný vkus a uživateli A by se mohla líbit i kniha 3.

2. Uživatel C i uživatel D nemají rádi knihu 2, mají možná podobný vkus, a proto se uživateli C nebude líbit ani kniha 4.

3. Nemáme tušení, jaké je propojení mezi výše zmíněnými skupinami uživatelů.

Jak lze vidět, z takovéto matice jdou vyčíst podstatné informace, které nám pomohou při doporučování kolaborativním filtrováním. A to nepotřebujeme vědět nic ani o uživatelích ani o knihách.

3.5 Typy kolaborativního filtrování

Rozeznáváme dva typy přístupů kolaborativního filtrování a těmi jsou:

1. přístup založený na paměti
2. a přístup založený na modelu

3.5.1 Přístup založený na paměti

Kolaborativní filtrování založené na paměti se při generování doporučení pro uživatele spoléhá výhradně na matici interakcí mezi uživatelem a položkou. Tento přístup je zcela závislý na minulých hodnoceních a interakcích uživatelů. [5]

Filtrování založené na paměti se skládá ze dvou metod:

1. kolaborativního filtrování založeného na uživatelích
2. a kolaborativního filtrování založeného na položkách.

Tyto metody využívají matici interakcí uživatelů a položek k předpovědím hodnocení položek. Běžně se jim říká *metody sousedství*, protože se zaměřují na určení vazeb mezi uživateli a položkami. [3]

Při rozhodování, zda použít přístup založený na uživatelích nebo na položkách, je důležité zvážit, jak často se uživatelé a položky v systému mění. Pokud je uživatelů více než dostupných položek, může být metoda založená na položkách lepší, protože systém může doporučovat položky novým uživatelům, i když se váhy podobnosti položek počítají jen občas. V online mediálních systémech, kde přibývá uživatelů obvykle rychleji než položek, jsou pak algoritmy pro doporučování založené na položkách pro komerční nebo online aplikace praktičtější volbou. [6]

3.5.1.1 Filtrování podle uživatele

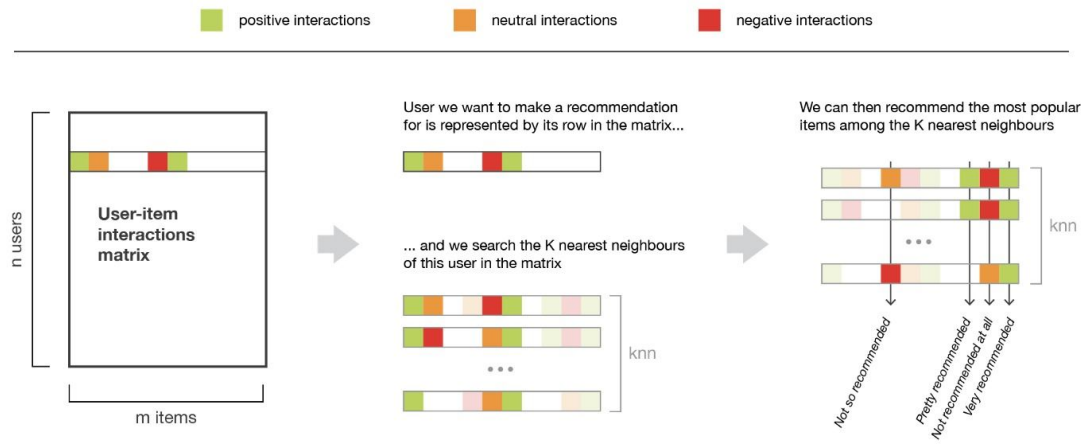
Aby bylo možné poskytovat nová doporučení konkrétnímu uživateli, vytvoří se na základě analýzy akcí provedených daným uživatelem soubor srovnatelných uživatelů (tzv. nejbližších sousedů). Doporučení jsou založena na nejoblíbenějších položkách v této skupině, které jsou pro cílového uživatele nové. [5]

Nyní přiblížím postup více pomocí obrázku 3.5 z publikace [5], která popisuje, jak kolaborativní filtrování funguje. Obrázek pro jednoduchost rozděluje postup do tří následujících kroků.

1. Identifikace matice interakcí uživatele a položek, která je datovým zdrojem pro nalezení doporučených položek. Kde proměnná n je počet uživatelů a zároveň velikost jedné z dimenzí matice a proměnná m je počet všech položek a zároveň velikost druhé z dimenzí matice.

2. Extrakce řádku matice korespondujícího cílovému uživateli, kterému budou položky doporučovány. Tento řádek pak daného uživatele reprezentuje při dalších operacích. V tomto kroku se také vyhledá k nejbližších sousedů cílového uživatele pomocí vektorové podobnosti. Jaké metriky se dají použít a jakou jsem zvolil já, popíšu v praktické části této práce.

3. Posledním krokem je doporučení položek, které jsou nejpopulárnější v okruhu nejbližších sousedů a se kterými cílový uživatel ještě neinteragoval. V tomto případě doporučujeme akorát položky, které byly nejbližšími sousedy hodnoceny kladně. Pro negativně hodnocené položky nemáme praktické využití. Na obrázku jsou kladně hodnocené položky označeny zeleně, neutrálně hodnocené jsou oranžové a negativně hodnocené jsou červené.



Obrázek 3.5 Ilustrativní popis kolaborativního filtrování na základě uživatele z publikace [5]

3.5.1.2 Filtrování podle položek

Proces filtrování podle položek zahrnuje výběr doporučení pro uživatele na základě jeho předchozích interakcí. To se provádí tak, že se zohlední položky, o které uživatel již projevil zájem, a pomocí nalezení podobnosti se identifikují podobné produkty. Z těchto shluků podobných položek jsou uživateli navrženy nové položky jako potenciální doporučení. [5]

Nyní přiblížím postup více pomocí obrázku 3.6 z publikace [5], která popisuje, jak kolaborativní filtrování funguje. Obrázek pro jednoduchost rozděluje postup do tří následujících kroků.

1. Identifikace položek, které cílový uživatel kladně ohodnotil.

   Kolik položek se zde vybere, ovlivňuje rozsah prohledávaných položek. Pro každou z vybraných položek se následující postup opakuje. Všechny výsledky se pak srovnají a vrátí se v pořadí nejlepšího skóre. Konkrétní postup popíšu v praktické části této práce.

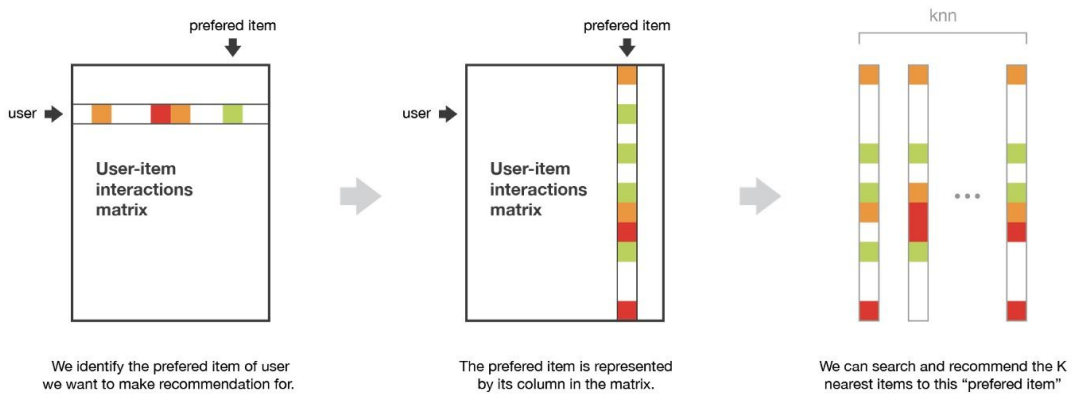
2. Extrakce sloupce matice korespondujícího vybrané položce. Tento sloupec pak danou položku reprezentuje při dalších operacích.

3. V tomto kroku se vyhledá k nejbližších sousedů vybrané položky pomocí vektorové podobnosti. Jaké metriky se dají použít a jakou jsem zvolil já, taky opět popíšu v praktické části této práce. Poslední částí tohoto kroku je doporučení položek, které jsou nejpodobnější položkám, které cílový uživatel ohodnotil kladně a se kterými cílový uživatel ještě neinteragoval. Na obrázku jsou kladně hodnocené položky označeny zeleně, neutrálně hodnocené jsou oranžové a negativně hodnocené jsou červené.

3.5.2 Přístup založený na modelu

Přístup založený na modelu zahrnuje využití modelů strojového učení k předvídání a upřednostňování interakcí mezi uživateli a položkami, se kterými ještě neprovedli interakci. K trénování těchto modelů jsou implementovány různé algoritmy, jako je maticová faktorizace, hluboké učení a shlukování, které využívají informace o interakcích, jež jsou již přítomny v matici interakcí. [5]

Tyto algoritmy vytvářejí modely, které reprezentují chování uživatelů a položek prostřednictvím skupiny faktorů nebo rysů a jejich příslušných vah pro každého uživatele a položku. [3]



Obrázek 3.6 Ilustrativní popis kolaborativního filtrování na základě položek z publikace [5]

Pro vytvoření prediktivního modelu pro doporučování položek uživatelům je potřeba data (soubor dat o uživatelích a položkách) předzpracovat s využitím informací o hodnoceních nebo předchozích nákupech. Během vlastního procesu doporučování je k předpovědím zapotřebí pouze předem vypočtený nebo „naučený“ model. Nejpoužívanějším přístupem v této kategorii jsou modely latentních faktorů, jejichž cílem je vytvořit hodnocení definicí položek a uživatelů na základě 20 až 100 faktorů, které jsou odvozeny ze vzorců hodnocení. [3]

Ačkoli jsem našel práce, kde výzkum naznačuje, že techniky založené na modelech jsou pro předpovídání hodnocení lepší než *sousedské přístupy*. Zdá se, že samotná „přesnost“ doporučení nestačí k tomu, aby uživatelé získali uspokojivý a efektivní zážitek. Je důležité vzít v úvahu i další faktory. [3]

3.5.2.1 Maticová faktorizace

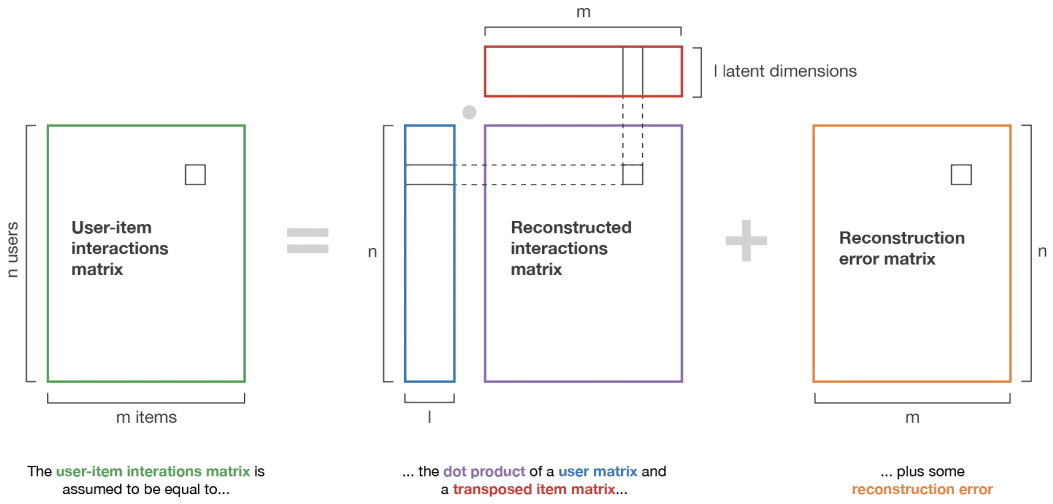
Ke konci této sekce o přístupech založených na modelu, popíšu ten nejběžnější postup pro získání výše zmíněných latentních faktorů, a tím je maticová faktorizace.

Jedná se o metodu, která se pokouší s co nejmenší chybou rozdělit matici interakcí uživatelů a položek do dvou samostatných matic, kde jedná reprezentuje pouze uživatele a druhá pouze položky. Obě matice získané faktorizací jsou vypočteny tak, aby skalárním součinem mezi sebou vytvořily zpátky původní matici interakcí. Myšlenkou je, že latentní faktory, které specifikují jednotlivé uživatele, nebo položky se vyskytují právě v v těchto maticích na jednotlivých řádcích, kde každý popisuje jednoho uživatele, nebo jednu položku. Proto se pak jedna nazývá uživatelská matice a druhá se nazývá matice položek. Po výpočtu takovýchto matic, pak lze například velice rychle vypočítat předpokládanou míru doporučení pro konkrétní položku, když maticově vynásobíme řádek uživatele z uživatelské matice s řádkem zkoumané položky z matice položek.

Nyní přiblížím postup faktorizace více pomocí obrázku 3.7 z publikace [7], která mimo jiné představuje kolaborativní filtrování založené na modelu.

Na obrázku jde zjednodušeně vidět, jak se získává uživatelská matice a matice položek. Obě matice skalárním součinem musí tvořit původní matici interakcí s minimalizací chyby v podobě matice rekonstruující chybu, která je napravo označena oranžovou barvou. Původní matice interakcí, která je nalevo označena zelenou barvou, má dvě dimenze o velikosti n uživatelů a m položek. Matice rekonstruující chybu má ty samé rozměry, ovšem uživatelská matice resp. matice položek, která se uprostřed označena modrou resp. červenou barvou, má kromě své dimenze o velikosti n resp. m, které určují počet uživatelů resp. položek, také druhou dimenzi o velikosti l, která je složená z latentních faktorů.

Tato metoda předpokládá, že nalezne skryté faktory definující uživatele nebo položky roz-



Obrázek 3.7 Ilustrativní popis maticové faktorizace z publikace [7]

dělením řídké matice interakcí do dvou hustých matic, redukcí do menší velikostí se pak může ušetřit výpočetní čas v dalších operacích. Tyto faktory ani není potřeba manuálně hledat, jako by to bylo u filtrování založeném na obsahu, samy se ukážou po faktorizaci původní matice.

K faktorizaci se často používá metoda gradientní sestupu, kde je cílem minimalizovat celkový rozdíl všech prvků z původní matice s prvky po součinu korespondujících řádků uživatelovy matice a matice položek. To samé lze vyjádřit pomocí následující rovnice převzaté ze článku [7].

$$(X, Y) = argmin_{X,Y} \sum_{(i,j)\in E} [(X_i)(Y_j)^T - M_{ij}]^2 \tag{3.1}$$

Kde X reprezentuje uživatelskou matici a X_i představuje i-tý řádek matice, tedy i-tého uživatele, i je mezi jednou až n. Y reprezentuje matici položek a Y_j představuje j-tý řádek matice, tedy j-tou položku, j je mezi jednou až m. E představuje všechny pozice (i,j), u kterých v případě matice M_{ij} hodnota existuje.

Toto je popis základní funkce, která se dá rozšířit např. o regularizační faktor.

3.5.3 Problém řídkosti/ studeného startu

Řídké matice interakcí mohou vést k nepřesným a nepřiměřeným předpovědím a doporučením kvůli nedostatku sousedů cílových položek.

To, že nastane takovýto problém, je pravděpodobné především na začátku fungování celého systému, kdy není k dispozici dost nasbíraných dat, ale také pokaždé pro nového uživatele, který nebude mít v matici interakcí žadný záznam.

Řešení tohoto problému lze vytvořit doplněním implicitních kandidátů na sousedy predikcí propojení. Predikce propojení zahrnuje odhad pravděpodobnosti odkazu mezi dvěma uzly na základě pozorovaných odkazů a atributů uzlů. Existují různé algoritmy pro predikci vazeb v grafech a metrika blízkosti vazeb, jako je Adamic-Adarův koeficient, ten je účinnou metodou pro identifikaci potenciálních uzlů, které mají být propojeny. [6]

3.5.3.1 Predikce propojení

Pro výpočet skóre podobnosti mezi uživateli se často objevují čtyři různé metody měření podobnosti. Jsou jimi:

1. společní sousedé,
2. Jaccardův koeficient
3. Adamic-Adarův index
4. a přednostní/ preferenční připojení

Přestože nakonec při implementaci nebylo potřeba ani jednu metodu použít, protože jsme tento problém vyřešili kombinací různých algoritmů doporučovacích systémů, což je popsáno v sekci níže, představím zde alespoň úvodně tyto metody pro predikci propojení.

3.5.3.1.1 Společní sousedé Výpočet určuje skóre podobnosti mezi dvěma uzly na základě počtu jejich společných sousedů. Pokud mají uzly x a y v čase t vysoký počet překrývajících se sousedů, pak existuje vysoká pravděpodobnost, že získají nové společné sousedy v čase $t + k$. Toto skóre podobnosti se nazývá podobnost společných sousedů.

3.5.3.1.2 Jaccardův koeficient Jaccardův koeficient vypočítává podobnost mezi x a y měřením procenta sousedů x, kteří také korelují s y. Zjednodušeně je to metrika, která zohledňuje korelaci mezi x a y.

3.5.3.1.3 Adamic-Adarův index Adamicova-Adarova metrika podobnosti vylepšuje výpočet společných sousedů tím, že uzlům s menším počtem vztahů přikládá větší váhu. Cílem této metriky je minimalizovat vliv populárních uzlů a upřednostnit uzly s reprezentativnějšími informacemi. Například pokud uzly x a y mají společné sousedy a a b, ale a má méně vztahů než b, budou hodnoty Adamic-Adarovy podobnosti pro uzly x a y založené na společném sousedovi a vyšší než hodnoty založené na b.

3.5.3.1.4 Přednostní připojení V sociálních sítích se často předpokládá, že uzly s velkým počtem spojení mají tendenci vytvářet více vztahů. Tento jev lze pozorovat i v oblasti finančních schopností, kdy bohatí jedinci mají vyšší pravděpodobnost, že se stanou ještě bohatšími [6]. Míra podobnosti preferenčních připojení je metoda používaná k určení "bohatství"uzlů x a y výpočtem součinu počtu spojení, která mají. Tato míra nevyžaduje žádné informace o samotných spojeních, což z ní činí relativně jednoduchý a efektivní výpočet.

3.5.3.2 Řešení pomocí tranzitivního propojení

Pokud jde o problém studeného startu, klíčovým problémem je generování přesných předpovědí s omezenými informacemi. Jednou z možných metod řešení tohoto problému je využití doplňujících informací o uživatelích, jako je jejich věk, pohlaví, vzdělání, zájmy a další relevantní údaje, které slouží k jejich klasifikaci. Alternativně lze také použít hybridní doporučovací systémy, které kombinují více přístupů do jednoho predikčního mechanismu. Tyto techniky již nejsou považovány za čistě kolaborativní a eliminují základní výhodu o doménové nezávislosti, protože vyvstávají otázky týkající se získávání dalších informací a integrace různých klasifikátorů. Přesto mohou být tyto metody užitečné v počátečních fázích nově zavedeného doporučovacího systému, aby bylo dosaženo kritického množství propojení, které je vyžadováno v kolaborativním přístupu. [3]

Postupem času byly navrženy různé metody, z nichž minimálně jedna je i založena na grafech. Tento přístup využívá tranzitivních asociací mezi uživateli na základě jejich minulých interakcí

a zpětné vazby. Hlavním cílem této techniky je využít předpokládané "tranzitivity"preferencí zákazníků, pokud mají společné položky. Kromě toho tento přístup přiřazuje každé hraně váhu, obvykle na základě hodnoty příslušného hodnocení. [3]

Pomocí této metody založené na tranzitivních asociacích je možné implementovat kolaborativní doporučení i přes malý iniciální počet interakcí v databázi. Tento proces zahrnuje výpočet asociací mezi uzly uživatelů a uzly položek. Standardní přístupy kolaborativního filtrování, jako jsou přístupy založené na uživatelích a položkách, uvažují pouze cesty o délce 3 hran. Když existuje vysoký počet jedinečných cest spojujících uzel položky s uzlem uživatele, je asociace mezi těmito dvěma uzly silnější. Vzhledem k malému počtu cest o délce 3 v řídkých databázích, je pro výpočet doporučení nutné uvažovat také delší cesty, známé jako nepřímé asociace. Je jednoduché rozšířit přístup tak, aby zahrnoval tranzitivní asociace v modelu založeném na grafech tím, že se uvažují cesty o délce větší než 3. To umožňuje modelu zkoumat tranzitivní asociace. [3]

Tento mechanismus, který vrací míru doporučení položky p pro uživatele a, je založen na součtu vah jedinečných cest spojujících uživatele a a položku p. V úvahu se však berou pouze cesty, jejichž délka je rovna nebo menší než stanovená maximální hranice M. Mohli bychom tak upravovat hodnotu M, která musí být liché číslo, aby reprezentovala tranzitivní asociace v bipartitním grafu. Je třeba zmínit, že M je parametr, který je třeba definovat, aby byla zajištěna přesnost doporučení. [3]

$$skore(a,p) = \sum_{p \in pathsBetween(a,p,M)} \alpha^{length(p)} \tag{3.2}$$

Ve výsledné formuli 3.2, pak *pathsBetween(a, p, M)* jsou všechny cesty mezi uživatelem a položkou o délce x, kde x je menší nebo rovno M. Váha cesty se vypočítá jako α na délku cesty, kde α je konstanta mezi 0 a 1, což zajišťuje, že delší cesty mají menší vliv. Konkrétní hodnotu α můžeme určit na základě charakteristik základní aplikační domény. Takže v aplikacích, kde tranzitivní asociace mohou být silným prediktorem zájmů spotřebitelů, by α měla nabývat hodnoty blízké 1, zatímco v aplikacích, kde tranzitivní asociace mají tendenci předávat málo informací, by α měla nabývat hodnoty blízké 0.

Pro doporučení položek uživateli se mu vrátí k nejlepších položek ze seřazeného seznamu, přičemž se vyloučí položky, se kterými uživatel již interagoval. Tato metoda je zajímavá z několika důvodů:

1. vytváří uspokojující doporučení, i když jsou k dispozici omezené informace,
2. navíc využívá stejný soubor dat, který používáme pro kolaborativní filtrování
3. a je zcela založena na grafu, tedy využívá stejný model bipartitního grafu [3]

Navrhovaná technika založená na nepřímých vztazích může ve srovnání se standardními algoritmy založenými na uživatelích a položkách výrazně zvýšit kvalitu doporučení, zejména pokud je matice interakcí řídká. Tento algoritmus navíc vykazuje měřitelné zlepšení výkonu pro nové uživatele ve srovnání s tradičními technikami kolaborativního filtrování. Jakmile však matice hodnocení dosáhne určité hustoty, může se kvalita doporučení ve srovnání se standardními algoritmy zhoršit. Doporučení se v popsaném přístupu počítají pomocí podobnosti založené na cestě. [3]

3.5.4 Zkombinování variant kolaborativního filtrování

U doporučovacích systémů založených na kolaborativním filtrování často nestačí spoléhat se pouze na variantě filtrování založené na uživateli nebo na variantě filtrování založené na položce.

K řešení tohoto problému lze k výběru relevantních položek ze skupiny potenciálních kandidátů použít hybridní přístup, který kombinuje obě metody. Tato technika zahrnuje použití

měr podobnosti s predikcí propojení k předpovědi hledané položky kombinací filtrováním podle uživatele s filtrováním podle položky. [8]

Výzkum [8] navrhuje metodu pro využití filtrování podle uživatelů i položek v kolaborativním filtrování založeném na grafech najednou. Navrhovaný přístup zahrnuje dva kroky. Nejprve se vyfiltruje top-N uživatelů s nejvyšším skóre podobnosti s cílovým uživatelem. Místo toho, aby se při doporučování spoléhalo pouze na podobnost uživatelů, jsou položky zakoupené těmito top-N podobnými uživateli považovány za potenciální kandidáty pro další krok. V dalším kroku jsou kandidáti seřazeni podle podobnosti položek. Metoda se nazývá TSUISIMCF, což je zkratka pro Two-Steps Combined User and Item Similarities in Graph-Based Collaborative Filtering (dvoukrokové kombinované filtrování podle podobnosti uživatelů a položek v grafu). [8]

My jsme se rozhodli implementovat *kaskádovité doporučování*. Jedná se o kaskádu doporučení, kde pokud daný přístup není schopen vrátit dostatečně vhodné výsledky, jsou jeho výsledky ignorovány a místo toho se vrací výsledky dalšího doporučení v řadě.

V době psaní této práce se fronta skládá z těchto typů doporučení:

1. kolaborativní filtrování podle uživatelů,
2. kolaborativní filtrování podle položek,
3. sémantické doporučení,
4. a doporučení na základě popularity

Nic ale nebrání tomu, v budoucnu přidat další položky do fronty, kde kolaborativní filtrování tvoří hlavní část mechanizmu. Jak konkrétně je implementováno toto řešení samozřejmě ukážu v praktické části.

3.5.4.1 Co je sémantické doporučení

Tato doporučovací metoda je založena na porovnávání položek a hledání sémanticky podobných položek. Pokud uživatel kladně ohodnotil, nebo jakkoliv projevil zájem o danou položku, lze předpokládat, že by mohl mít zájem o doporučení podobné položky. K položkám, o které uživatel projevil v minulosti zájem, se tak vyhledávají položky, které jsou sémanticky podobné. K reprezentaci položek se používá popisný vektor, který sémanticky položku reprezentuje. Výhodou této reprezentace je, že se s ní dají provádět typicky vektorové operace, jako průměrování množství vektorů k odhadu „průměrné“ položky, o kterou uživatel má zájem. S touto reprezentací se taky dá jednoduše porovnávat podobnost položek prostřednictvím porovnávání podobnosti číselných vektorů.

Toto doporučení tedy průměruje popisné vektory posledně zobrazených položek, o které uživatel projevil zájem. K tomuto zprůměrizovanému vektoru se pak hledají nejbližší sousedé. Funkcí používaných k měření vzdálenosti mezi vektory za účelem určení k-nejbližších sousedů lze zvolit hned několik jako:

1. vzdálenost normy L1, která se vypočítává jako součet absolutních hodnot rozdílu prvků vektorů,
2. vzdálenost normy L2, která se vypočítává jako součet druhých mocnin rozdílu prvků vektorů,
3. kosínova podobnost,
4. atd.

Popisný vektor položky se získá text embedingem (vnořením textu), což je způsob vektorové reprezentace textu. Text vložený do modelu, v našem případě model jménem DistilBERT, je vytvořen slepením textových atributů položky, které se liší od druhu entity. Např. pro knihu se takový text složí ze slepení názvu knihy a popisu knihy.

3.5.5 Výhody kolaborativního filtrování

Na konec popisu kolaborativního filtrování popíšu výhody i nevýhody tohoto doporučování a začnu výhodami. Mezi výhody kolaborativního filtrování lze zařadit:

1. Nezávislost na položkách—Není třeba mít specifické znalosti o doméně nebo typu položek, protože doporučení se zakládá na matici interakcí uživatele s položkami.

2. Doporučení propojením podobně smýšlejících—Má potenciál pomoci uživatelům najít nové zájmy, i když je aktivně nevyhledávají, tím, že jim nabídne nové položky, které jsou podobné jejich stávajícím zájmům. Pro příklad se může jednat o položku vysoce hodnocenou podobným uživatelem, ta nemusí odpovídat typickým preferencím, takže doporučení sice nemusí být jistým hitem, jako něco jiného, ale může uživateli představit dosud neprozkoumaný nový zájem.

3. Doménová nezávislost—Neobsahuje doménová omezení a lze jej využít v různých případech použití a scénářích. Navíc dokáže zpracovávat datové aspekty, které je obvykle náročné identifikovat pomocí filtrování obsahu. [3]

4. Relativní jednoduchost—Použití metod založených na sousedství pro predikci je jednoduché a snadno použitelné. Samo o sobě vyžaduje úpravu především jednoho parametru, kterým je počet použitých sousedů. Oproti přístupům založeným na modelu, které obvykle využívají techniky faktorizace matice, které se při hledání nejlepších možných řešení spoléhají na optimalizační algoritmy. Tyto optimalizační techniky mají mnoho parametrů, které je třeba pečlivě nastavit, aby se zabránilo uvíznutí v suboptimálním řešení. [3]

5. Porozumění výsledkům doporučení—Použitá technika nabízí stručné a srozumitelné vysvětlení provedených předpovědí. V případě doporučení založených na položkách lze uživateli zobrazit seznam podobných položek a hodnocení, které obdržely. To může zlepšit uživatelovo porozumění procesu doporučování a výpočtu relevance, což v konečném důsledku zvyšuje jeho důvěru v doporučovací systém a platformu, na které se nachází. [3]

6. Časová efektivita—Systémy založené na sousedství mají významnou výhodu z hlediska časové efektivity. Na rozdíl od většiny systémů založených na modelech nevyžadují ve velkých komerčních aplikacích nákladné a časté tréninkové fáze. Ačkoli mohou vyžadovat předvýpočet nejbližších sousedů, je to obecně méně nákladné než trénování modelu v přístupu založeném na modelu. Kromě toho lze malou část modelu znovu vypočítat, jakmile jsou k dispozici nové informace, což umožňuje téměř okamžitá doporučení. Ukládání těchto nejbližších sousedů navíc vyžaduje minimální paměť, takže tyto přístupy jsou škálovatelné pro aplikace s miliony uživatelů a položek. [3]

7. Stabilita—Výhodou doporučovacích systémů, které se řídí tímto přístupem, je, že je do značné míry neovlivňuje pravidelný příliv uživatelů, položek a hodnocení, který se běžně vyskytuje v rozsáhlých komerčních aplikacích. Po výpočtu podobnosti položek může systém založený na položkách snadno poskytovat doporučení novým uživatelům, aniž by bylo nutné systém znovu trénovat. Navíc při přidání hodnocení nové položky stačí, aby systém spočítal podobnosti mezi novou položkou a položkami, které již v systému jsou. [3]

8. Grafová forma—Výsledný graf nabízí výhody pro lokální navigaci v datech během aktualizace modelu nebo předpovědí, stejně jako pro porozumění a přístupnost. [3] Kromě toho mohou grafové modely řešit problém studeného startu, který byl zmíněn výše, pomocí tranzitivní asociace.

   Pokud data reprezentujeme jako bipartitní graf, pak jeho výhodou je, že se do něj ukládají pouze nezbytné informace a neplýtvá se pamětí. Tato reprezentace navíc urychluje tvorbu modelu tím, že se zaměřuje na potenciálně relevantní sousedy. [3]

Tato síť grafu je uživatelsky přívětivá během procesu doporučování a poskytuje dodatečné znalosti získané ze stávajícího souboru dat o uživatelích a položkách, které lze využít k různým účelům, například k segmentaci zákazníků a shlukování položek.

K řešení problému řídkosti dat, či problému studeného startu se častokrát dají použít grafové algoritmy. Např. PageRank a navigační metody, jako je pathfinding, se využívají k vyplnění mezer a vytvoření hustší reprezentace souboru dat o uživatelských položkách. [3]

3.5.6 Nevýhody kolaborativního filtrování

Mezi nevýhody kolaborativního filtrování pak lze zařadit:

1. Riziko studeného startu—Pokud je dat málo, může být obtížné doporučovat nové produkty nebo uživatele na základě předchozích interakcí.

   Problém studeného startu je pro kolaborativní filtrování výzvou, protože se snaží nabídnout přesná doporučení pro nové uživatele a to i v případě, že jsou k dispozici jen omezené údaje o interakcích. Existují však způsoby, jak tento problém řešit, například pomocí různých algoritmů popsaných výše, včetně grafového přístupu a to tranzitivní asociací. [3]

2. Omezené pokrytí—Kvůli vázanosti na matici interakcí některé položky nebudou nikdy doporučeny, pokud podobní uživatelé s nimi nikdy neinteragovali.

   Z dlouhodobého hlediska může docházet k nedostatečné rozmanitosti navzdory snahám navrhovat uživatelům nové položky. Je to proto, že algoritmy se spoléhají na historická data a nemusí doporučovat položky s malým množstvím dat, což vede k tomu, že oblíbené produkty se stanou ještě oblíbenějšími a nových možností bude navrženo málo. [5]

3. Škálovatelnost—S rostoucí uživatelskou základnou se algoritmy potýkají s problémy kvůli velkému objemu dat a problémům se škálovatelností. Uvědomuji, že toto může znít mírně jako kontradikce s popisem jednoho z bodů ve výhodách, nicméně lze to formulovat tak, že škálovatelnost tohoto filtrování není bezkonkurenčně nejlepší.

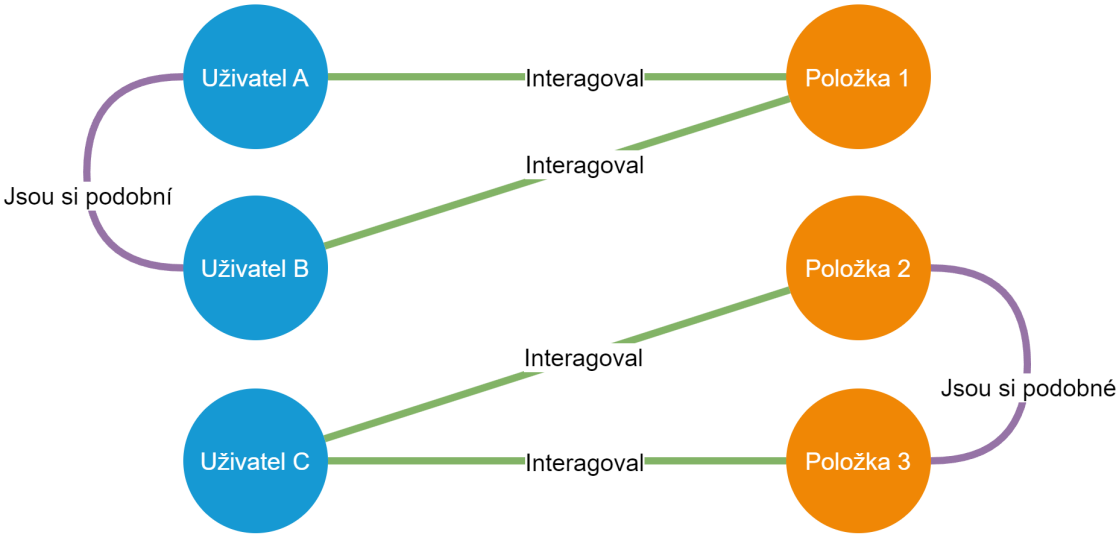
3.6 Představení grafové databáze pro doporučování

Nyní lehce představím, co je grafová databáze a její implementaci jménem Neo4J. Tento typ databáze spolu s Neo4J jsme již popisoval ve své bakalářské práci [1] a to relativně dopodrobna. Zde zmíním znova ty nejdůležitější části, pro detailnější info ale samozřejmě doporučuji korespondující sekce v mé minulé práci.

„Grafová databáze je založena na teorii o grafech. Řadí se mezi *NoSQL* databáze. Je tomu protože databázový designer nedefinuje jasně dané schéma, jak jsou informace uloženy, pomocí kterého by se vynucovaly pravidla pro nová data. To je do určité míry zajištěno integritními omezeními, ale nejedná se o plnohodnotnou náhradu DDL viděného u relačních databází. To znamená a je výhodou, že např. pokud se kus dat mírně liší, není nutné hned modifikovat design databáze. Na druhou stranu to vede k sub-optimalitě dotazování. Přesto u struktur často měnících se, nebo generovaných uživatelem se jedná o výhodu.“ [1]

Jak jsem již popisoval u důvodu k výběru kolaborativního filtrování jako doporučovacího systému, data, která spravuje LearnerOn a jsou určeny k doporučení, nejsou úplně jednotvárná. Konkrétní seznam datových typů je možné najít dopodrobna popsaný také v mé bakalářské práci [1]. Proto grafová databáze mimojiné je dobrou volbou pro ukládání našich položek pro doporučování.

„V samotné formě se — jako graf — skládá z uzlů (Node) a hran (Edge/ Link). Data tedy ukládá do uzlů (ekvivalenty záznamu řádku v relační databázi) a vztahy mezi těmito daty do hran — orientovaných či neorientovaných — mezi nimi. Vztahy tedy jsou First-class objekty,



Obrázek 3.8 Ilustrace propojenosti uzlů v databázi

což znamená, že stejně jako uzly mohou mít popisy (Labels) a přiřazené atributy (jako sloupce v relační databázi). Narozdíl od relačních databází, které zjišťují takové informace o spojeních run-time“. [1]

To je výhodné v našem případě, kde máme velice propojené data. Na tomto obrázku 3.8 jde vidět jádro uzlů, které budeme používat při doporučování. Na obrázku moc hran vidět nelze, ale počet hran v grafu ve skutečném produkčním prostředí při škálování dat velice naroste. Každý nový uživatel totiž může vytvořit hranu podobnosti na jakéhokoliv uživatele. Každý uživatel může interagovat s libovolnou položkou. A pro položky existuje podobný případ, kde každá nová položka může být podobná libovolné existující.

3.6.1 Porovnání s relační databází

„Grafové databáze jsou výrazně rychlejší pro propojená data. Kvůli jiné struktuře, rychlost výsledku nezávisí tak na velikosti uložených dat jako na velikosti prozkoumaného stromu. Grafové databáze ukládají spojení jako First-class objekty, dochází proto k procházení v teoreticky konstantním čase. To je podpořeno měřeními v implementaci, ve stroji Neo4j.“ [1]

V případě použití databáze pro doporučování budou typy hledání přes velké množství hran časté, bude se jednat o hledání na základě kontextu, historie, či preferenci uživatele. Tedy konkrétními důvody pro volbu grafové databáze ze strany firmy LearnerOn byly především:

1. „struktura dat, se kterými portál pracuje a hodlá pracovat má grafový charakter a
2. řada pokročilých vyhledávacích funkcí využívajících metody strojového učení a umělé inteligence je rovněž založena na grafových algoritmech.“ [1]

Pokud se jedná o spoustu dat bez velkých agregací, či spojování, pak použití grafové databáze není nejlepší volbou a je lepší zvolit databázi relační. A protože se ve většině případů spouštějí právě tyto typy dotazů, kompletní přechod na grafovou databázi by pro firmu LearnerOn nedával smysl, místo toho se zvolená grafová databáze používá v současnosti pouze jako prostředek v získání doporučování.

„Implementací bylo zvoleno Neo4J. Díky komunitní verzi je cenově přístupný a zároveň neztrácí na zralosti. Podporuje spoustu knihoven a funkcí právě pro aplikaci metod strojového

učení a umělé inteligence s potřebnými algoritmy jako např. *Node Embedding*. Má vlastní vyhledávací jazyk jménem Cypher, který je podobný SQL. Poskytuje taktéž kvalitní knihovnu pro jazyk Python, který je mezi lidmi pracujícím s algoritmy strojového učení a umělé inteligence velmi oblíben.“ [1]

3.6.2 Neo4J

Neo4J je velice populární implementací výše popisované grafové databáze. „Neo4J má hrany grafu pouze orientované, což nevytváří problém s jakýmkoliv návrhem, jelikož neorientovaná hrana se na orientovanou dá rozšířit. Tyto hrany musí mít vždy definován začátek a konec (muže se jednat o jediný uzel). Jak bylo zmíněno v části o grafových databázích, hrany mají stejné možnosti popisu jako uzly, což je rozdíl oproti ostatním databázím.“ [1]

Databáze je plně ACID podporovaná neboli garantuje pro všechny transakce Atomicitu, Konzistenci, Izolaci a Durabilitu. Tato implementace je založena grafovém modelu (v původním znění *Labeled Property Graph Model*) se štítky a vlastnostmi.

Dotazovacím jazykem je Cypher. Podobně jako SQL se jedná o deklarativní jazyk, který je vyvíjen pro účely Neo4J, nicméně jeho tvůrci z něj chtějí udělat jazyk pro podporu dotazování se i do jiných databázových strojů. Pro bližší popis doporučuji sekci o tomto deklarovacím jazykuv mé bakalářské práci [1]. Ikdyž je syntaxe velice srozumitelná i pro běžného čtenáře, tak pro pochopení, jak se získává doporučení kolaborativním filtrováním přímo z databáze, je důležité rozumět alespoň na základní úrovni syntaxi Cypheru. Místy také použiji funkce z knihovny APOC, která rozšiřuje některé z funkcionalit databáze. Vždy ovšem popíšu, jak se funkce chová a v jakém kontextu byla použita.

Integritní omezení poskytují způsob, jak vymezit pravidla grafu, nicméně jsou extrémně omezená a v době psaní této práce v podstatě limitovaná na zajištění existence a unikátnosti atributů.

V průběhu celé práce se používá Neo4J edice Enterprise verze 4.3.3. Edice Enterprise je sice placenou verzí, která poskytuje množství funkcionalit navíc, nicméně nic z ukázaných příkladů v praktické části této práce nevyužívá těchto možností, nebo možností pro které neexistuje zdarma varianta.

Kapitola 4

Praktická část

V této kapitole ukážu praktický postup mé diplomové práce. Od představení datové části, přes implementaci až k rozvaze nad budoucím rozvojem existujícího doporučovacího systému. První sekce představuje část datového modelu v grafové databázi, která je použita při doporučování a popisuje výhodnost formy bipartitního grafu. Na to navazující sekce popisuje velice důležitou část, bez které by nebylo možné doporučení provozovat, je jí naplnění a synchronizace dat potřebných pro algoritmy k doporučování. Další sekce pak popisuje samotnou implementaci doporučovacího mechanizmu. Jak tato kaskáda doporučovacích kroků se snaží vrátit co nejlepší výsledky, tak které algoritmy tvoří tuto kaskádu. Poté je sekce, která ukazuje výsledky testování doporučování na testovacích datech. A poslední část je rozvahou nad budoucím rozvojem doporučovacího aparátu pro LearnerOn.

Protože předmětem práce je doporučovací mechanizmus, ne systém okolo toho, neodevzdávám celý Backend aplikace, ale pouze útržky, které slouží pro doporučování samotnému, nebo jsou s ním výhradně spojené. Tyto útržky jsou často určitým pseudokódem formátovaným dle syntaxe Pythonu, ve kterém je produkční kód implementován. Samotné dotazy na grafovou databázi, které provádějí kolaborativní filtrování jsou vždy popisovány ve formě, v jaké jsou použity pro implementaci na produkci. Důvodem pro zvolení pseudokódu pro zbytek implementace je, že ukázaný pseudokód je jazykově agnostický a je jednoduché je pochopit. Tímto může sloužit čtenáři jako lepší inspirace pro jeho použití.

4.1 Představení části datového modelu použitého k doporučování

Datový model grafové databáze je široký za účelem pokrytí plánovaných využití, které nutně nebenefitují doporučování, které popisuji v této práci. Dalo by se naopak tvrdit, že zbytečné data zahlcují a zpomalují čas odpovědi pro doporučení a negativně ovlivňují zkušenost zákazníků. Protože, jak jde vidět na obrázku 3.1 entit je v databázi hodně, tak po popisu změn modelu, které se odehrály od zveřejnění mé bakalářské práce, se budu věnovat dále v práci především konkrétní části celého modelu, který obsahuje uživatele, interakce a položky. Tato část bude symbolizovat bipartitní graf, i když se o něj ve skutečnosti jednat nebude. Co je bipartitní graf a proč graf databáze ve skutečnosti bipartitní graf tvořit nebude popíšu částečně v této sekci ale také v sekci o kolaborativním filtrování.

4.1.1 Typy položek k doporučování

Nyní představím všechny typy položek v systému k doporučení uživateli, všechny zmíněné typy položek jsou rozebrané v podrobnější formě v mé bakalářské práci [1].

Vzdělávací materiály, které budu nazývat položkami, jelikož je to zavedený pojem reprezentující předmět doporučování, se dělí na následující typy:

Kniha je reprezentace knihy typicky definovaná svým ISBN (mezinárodním standardním číslem knihy).

Kurz je externí online výukový kurz nabízený LearnerOnem jakožto prostředníkem.

Odkaz na vzdělávací materiál (Odkaz) slouží pro referenci na materiály, které vedou uživatele typicky mimo doménu portálu LearnerOn, rozeznáváme ještě následující podrozdělení odkazů, které definuje obsah, na který *Odkaz* směruje.

Jsou jimi následující podtypy:

- Web,
- Video,
- Článek,
- Podcast,
- Kód,
- Kniha,
- a Kurz

Každý z podtypů svým názvem jednoduše vysvětluje, na jaký typ obsahu musí zvolený odkaz ukazovat, aby mu byl přiřazen daný typ. Pokud při přiřazování typu nelze usoudit, o jaký typ odkazu se jedná, pak se volí automaticky typ *Web*.

Vzdělávací cesta je sada výukových materiálů výše zmíněných typů ale i dalších vzdělávacích cest. Vzdělávací cesta je setříděný strom s posloupností pro postup při vzdělávání.

V ontologickém modelu ale i taky grafové databázi používáme navíc pojem *Resource* (ve smyslu informačního zdroje), ten v ontologickém modelu zastřešuje definičně všechny vzdělávací materiály a v databázi slouží k jejich jednoduchému vyfiltrování.

4.1.2 Typy uživatelských interakcí

Nyní představím všechny typy interakcí v systému, které jsou konkrétněji rozebrané v mé bakalářské práci [1]. A vysvětlím, proč se při doporučování popisovaném v této práci nepoužívají všechny typy, které se používají a jak.

Interakce se sledují u každého návštěvníka, jen některé se ovšem dají důvěryhodně využít při doporučování a jen v některých případech jejich počet ospravedlňuje jejich kontribuci v doporučovacím procesu. Interakce se dělí na veřejně dostupné a privátní. Veřejné interakce se až na výjimky ukládá pouze v kontextu jedné návštěvy a problém je s jejich objektivní reprezentací chování uživatele. To privátní interakce jsou přístupné pouze přihlášeným osobám, v tomto případě pak lze při předpokládání, že se návyky a preference jednoho uživatele nemění, doporučovat položky na základě jejich sledované historie interakcí.

Druhé dělení interakcí se zakládá na typu činnosti. U tohoto dělení je viditelný problém, že většina typů interakcí nemá dostatečně reprezentativní počet záznamů. Pro připomenutí zmiňované dělení činností je následující:

Vytváření entit jako cest, kurzů, linků atd.

Okomentování entit nejčastěji jimi je vzdělávací cesta

Zobrazení entit jako cest, kurzů, linků atd.

Vymazání vytvořených nebo vlastněných entit

Hlasování v anketách

Vyhledávání dotazu, tato interakce se oproti ostatním neváže na ostatní materiály, takže automaticky odpadává z možnosti použití v kolaborativním filtrovaní

Editování vytvořených nebo vlastněných entit

Sledování jiných uživatelů, nebo novinek atd.

Při pohledu na tyto typy interakcí a jejich četnosti jistě není překvapením, že největší množství zastupuje interakce *Zobrazení*, naopak v ostatních případech použití pro kolaborativní filtrování buď teoreticky není možné, nebo prakticky kdy je záznamů neskutečně málo. Navíc také kvůli problému s hledáním váh důležitosti jednotlivých typů interakcí mezi sebou, padlo rozhodnutí brát ohled pouze na interakci *Zobrazení*, které je binární a přináší informaci, zda uživatel danou položku viděl, nebo ne.

4.1.3 Změny v datovém modelu od zveřejnění bakalářské práce

Jak jsem popisoval při představování konceptuální modelu tohoto řešení v mé bakalářské práci [1], tak „Původní MySQL databáze obsahuje také informace, které nejsou v nové databázi potřeba. Ty v ontologickém modelu ani v budoucí implementaci nebudou. Ve většině případů se jedná o pomocné a administrativní tabulky, některé jsou zastaralé a postupným vývojem ztratily význam. Některé ovšem přinášejí důležité informace a nejsou v modelu pro jejich nepotřebnost v současném využití, nicméně v budoucnu mohou být.“ [1] Nejznatelnějšími tabulkami byly:

News je tabulka s novými informacemi pro uživatele

Questions je tabulka s otázkami, které mohou být pokládány k různým entitám v aplikaci

K nim je změna jednoduchá …již neexistují.

Protože jak kolekce tak vzdělávací cesty se skládaly z ostatních zdrojů informací, oba prvky byly vlastně seznam zdrojů. A zatímco myšlenka za kolekcí byla, že je to jednoduchá neseřazená množina zdrojů informací a vzdělávací cesta je komplikovanější. Protože samotná implementace a jejich vlastnosti byly velice podobné, došlo k rozhodnutí se naprosto zbavit podpory pro kolekce a stávající kolekce překonvertovat na vzdělávací cesty. Tyto nové cesty pak mají příznak, který signalizuje, že je cesta neseřazená a neměl by se brát zřetel na zobrazované pořadí. Originální reprezentace cesty pořád představuje strom kvůli vhodnosti při provádění výukovými materiály, jak je potřeba.

4.1.4 Co je bipartitní graf

Doporučovací datový model je obvykle reprezentován bipartitním grafem nebo bigrafem, což je graf složený ze dvou různých množin uzlů nazývaných U a V. V takovém grafu každá hrana spojuje uzel z U s uzlem z V. V našem případě představuje množina U produkty nebo položky, zatímco množina V představuje uživatele. Uživatelé jsou připojeni k položkám, s nimiž interagovali, například je okomentovali nebo se jim líbí. Hranám je možné přiřadit váhy nebo síly, které udávají četnost nákupů uživatele nebo míru, do jaké se mu daná položka líbí. Projekce bipartitního grafu pak poodhalují vztahy mezi uživateli, kteří sdílí historii interakcí, nebo mezi položkami, se kterými interagovali stejní uživatelé. [3]

4.1.4.1 Výhody použití této reprezentace

Bipartitní grafy představují efektivní a srozumitelný způsob reprezentace dat pro náš případ, kde v podstatě o jediné o co nám jde, jsou ojedinělé interakce uživatelů a položek. Soubor dat o uživatelích a položkách je typicky řídký s mnoha uživateli a položkami a jen malým zlomkem interakcí mezi nimi. Vytvoření matice, která by tyto interakce reprezentovala, by vedlo k mnoha nulám a málo užitečným hodnotám. Bipartitní grafy naproti tomu zobrazují pouze vztahy, které zprostředkovávají smysluplné informace. Pro něco jako multimodální doporučovací systémy lze rozšířit reprezentaci dat a modelovat více bipartitních grafů se stejnou sadou vrcholů, ale různými hranami. Tento přístup se používá v multimodálních doporučovacích systémech, které využívají různé typy interakcí k poskytování přesnějších doporučení. [3]

Přestože při pohledu na celý model grafu na obrázku 3.1, nelze mluvit o tom, že by celý graf byl bipartitní, zmiňuji jej a jeho výhody pro určitý důvod. Používáme grafovou databázi, kde data reprezentují graf a pro účel doporučování budeme pracovat s podgrafem, který při „rozložení“ vrcholů interakcí na hrany, je bipartitním grafem. Proto zmiňuji i výhody bipartitního grafu, které jsou aplikovatelné na náš podgraf.

4.2 Synchronizace dat relační a grafové databáze

Pro správné fungování doporučování bylo potřeba zprovoznit dva typy synchonizace dat:

1. dávkovou synchronizaci
2. a kontinuální synchronizaci.

Dávkovou synchronizací se myslí transformace všech potřebných datových entit z relační databáze do grafové databáze. Toto je potřeba především při inicializaci grafové databáze, ale také při velkých změnách relační databáze, nebo když se stane nějaká chyba při kontinuální synchronizaci. Kontinuální synchronizace pak je aplikování změn z relační databáze do grafové databáze u každého záznamu jednotlivě v reálném čase. Oba typy synchronizace fungují na stejné logice BE, pouze dávková synchronizace se spouští jednou za 14 dní na vybraných záznamech.

4.2.1 Napojení Neo4J objektů na Django objekty

Tato část je velice technologicky závislá, přesto jsem si jist, že přináší dobrý náhled, jak řešit synchronizaci entit mezi dvěmi různými databázemi.

Jsem si vědom, že existuje jistá podpora pro Neo4J i ze strany Django, bohužel během implementace tohoto synchronizačního mechanizmu tato podpora nevystačovala našim potřebám. Faktem je, že vymyšlená transformace z relační databáze do grafové není přímočará, v jistých případech totiž některé tabulky z relační databáze v grafové databázi úplně zanikají ve prospěch hran. Nebo někdy vznikají nové typy vrcholů kvůli původnímu návrhu, který ontologicky

nedává smysl. Takovéto transformace Django v té době nepodporoval, proto vznikl vlastní mechanizmus, který pracuje s objekty Neomodelu (knihovna pro mapování objektů pro grafovou databázi Neo4J) a s objekty Django modelu.

Až na výjimky existují vždy korespondující objektová reprezentace Neo4J entity s objektovou reprezentací Django modelu, proto vzniklo rozhodnutí vytvořit vlastní mapování a napojit změny Django objektů na mapování, které automaticky uloží změny i do grafové databáze. Nyní popíšu na zjednodušené variantě, jak zmíněná transformace funguje. Ve skutečnosti je realizace mírně složitější kvůli již zmíněné variabilitě transformace u některých tabulek.

V prvním útržku kódu 4.2.1 máme mapu jménem *model_to_graph_data_sync_map*, ta spojuje korespondující Django modely s modely Neomodelu. Modely grafové databáze jsou zabalené ve třídě *GraphDataSync*. Ta v tomto případě slouží pro vrácení funkce, která se má volat na objektu Neomodelu při uložení objektu Django modelu. Pro napojení Django signálu s ukládací funkcí pro objekt Neomodelu slouží funkce *connect_model_to_graph_data_to_signals*. Tato funkce prochází mapu a zajistí to, že když se uloží objekt Django modelu zavolá se na korespondujícím modelu Neomodelu *create_or_update_node* funkce s obsahem právě ukládané instance.

```
model_to_graph_data_sync_map = {
    User: GraphDataSync(UserNode),
    CookieConsent: GraphDataSync(CookieNode),
    Tag: GraphDataSync(TagNode),
    Visitor: GraphDataSync(VisitorNode),
    ...
    }

class GraphDataSync:
    def __init__(self, node_cls):
        self.post_save_receiver_function = self.create_post_save_receiver_function(
    node_cls)

    @staticmethod
    def create_post_save_receiver_function(node_cls: Type[LearneronNode]) -> Callable
    :
        def default_post_save_receiver_function(instance, **kwargs):
            return node_cls.create_or_update_node(instance)

        return default_post_save_receiver_function

def connect_model_to_graph_data_to_signals():
    for model, graph_data_sync in model_to_graph_data_sync_map.items():
        post_save.connect(graph_data_sync.post_save_receiver_function, sender=model)

connect_model_to_graph_data_to_signals()
```

V druhém útržku je třída *LearneronNode*, která zastřešuje všechny ostatní třídy entit z grafové databáze. V ní je funkce *create_or_update_node*, což je právě ta funkce, která se volá při uložení objektu do relační databáze. Ta vytváří nebo upravuje exitující objekt v grafové databázi dle *id* instance, které se vždy zachovává stejné mezi oběmi databázemi.

```
class LearneronNode(StructuredNode):
    @classmethod
    def create_or_update_node(cls, instance):
        node: LearneronNode = create_or_update_node(cls, instance)
        node.connect_relationships(instance)
        return node

def create_or_update_node(cls: Type[StructuredNode], source_obj) -> StructuredNode:
    node = get_or_create_node(cls, id_=source_obj.id)
    set_node_properties(node, source_obj)
    node.save()
    return node
```

Takto popsaný mechanizmus pak automaticky zajistí kontinuální synchronizaci při uložení jakéhokoliv záznamu přes Django mapování. K tomu aby mechanizmus fungoval kompletně tak je ještě ovšem potřeba analogickým způsobem naimplementovat ošetření při vymazání záznamu a podporu pro many-to-many mapování.

4.2.2 Spouštění dávkové synchonizace

Jak jsem již popisoval, je potřeba periodicky spouštět kompletní synchonizaci. Ta je triviální při použití funkcionality, na které funguje kontinuální synchronizace. Myšlenkou je zavolat kontinuální synchonizaci na všech potřebných záznamech postupně a tím aktualizovat i všechny entity v grafové databázi.

Nyní popíšu část kódu, která se periodicky spouští k provedení dávkové synchonizace.

```
orm_classes = [
    User,
    Tag,
    Visitor,
    CookieConsent,
    ...
]

def run_sync():
    for cls in orm_classes:
        create_index(cls)

        objs_step = 200
        for i in range(0, cls.objects.count(), objs_step):
            for obj in cls.objects.all()[i:i + objs_step]:
                obj.save()

        delete_orphans(cls, sync_start_time)
```

V prvním útržku kódu 4.2.2 je seznam *orm_classes* všech Django modelů, které se mají a uložit do grafové databáze. Tento seznam se prochází ve funkci *run_sync*, kde se prvně vytvoří index pro daný štítek, který odpovídá názvu Django modelu, v grafové databázi ve funkci *create_index*. Pro každý model se pak volá ukládání na každém záznamu, což vyvolá kontinuální synchronizaci s grafovou databází, jak bylo popisováno výše. Pro ušetření pracovní paměti se proces provádí po dvě stě záznamech postupně. Po uložení všech záznamů se pak volá funkce *delete_orphans*, ta odstraní v grafové databázi entity, které již nemají korespondující reprezentaci v relační databázi, a nepodařilo se je korektně vymazat při volání napojeného Django signálu na vymazání Neomodel objektu. To je obdoba napojení, které jsem popisoval výše, jen se provádí při mazání instance, a ne při ukládání.

Následující útržek kódu popisuje zmíněné funkce *create_index* a *delete_orphans*.

```
def create_index(cls):
    node_cls = model_to_graph_data_sync_map[cls].node_cls
    query = f"CREATE INDEX IF NOT EXISTS FOR (n:{node_cls.__name__}) ON (n.id)"
    db.cypher_query(query)

def delete_orphans(cls, sync_start_time):
    node_cls = model_to_graph_data_sync_map[cls].node_cls
    to_be_deleted = run_query_and_inflate(
        f"MATCH (n:{node_cls.__name__}) "
        f"WHERE n.last_updated < datetime(\"{sync_start_time}\").epochMillis / 1000
    or n.last_updated is null "
        f"RETURN n;", node_cls.inflate)
    for to_delete in to_be_deleted:
        to_delete.delete()
```

V druhém útržku kódu 4.2.2 jdou vidět implementace předem zmíněných funkcí *create_index* a *delete_orphans*. První funkce akorát pošle dotaz do grafové databáze s vytvořením indexu se jménem třídy, která je daná parametrem *cls*. Druhá funkce vyhledá všechny vrcholy, který se promítnuly do databáze transformací ze třídy *cls*, a pokud jim nebyl aktualizován čas změny vrcholu od hodnoty parametru *sync_start_time*, který symbolizuje právě běžící dávkovou synchronizace, tak funkce takové vrcholy vymaže. Jinou formou řečeno, pokud se synchronizace nepokusila o uložení tohoto vrcholu, tak to znamená, že v relační databázi již není instance, která tento vrchol vytvořila. A tím pádem ani tento vrchol nemá co existovat v grafové databázi.

4.3 Představení procesu doporučování

Nyní představím avizovanou kaskádu doporučení a podrobně ji popíšu.

```
@staticmethod
def resolve_material_recommendations(self, info, limit):
    user = info.context.user

    # try to give recommendation based on cosine sim of user x item vectors
    # give USER BASED recommendation
    try:
        recommendations = get_user_based_recommendations(user.id, limit)
        if len(recommendations) == limit:
            return get_django_object_types_from_nodes(recommendations)
    except ... as _:
        pass

    # give ITEM BASED recommendation
    try:
        recommendations = get_item_based_recommendations(user.id, limit)
        if len(recommendations) == limit:
            return get_django_object_types_from_nodes(recommendations)
    except ... as _:
        pass

    # give SEMANTIC recommendation
    django_object_typed_recommendations = \
        get_history_based_material_recommendations(info, limit)
    if len(django_object_typed_recommendations) == limit:
        return django_object_typed_recommendations

    # give POPULARITY BASED path recommendation
    return get_popularity_based_recommendations(limit)
```

Jak jde vidět v ukázce kódu 4.3 metoda *resolve_material_recommendations* vrací doporučení pro uživatele vloženého do parametru *info* s maximální kapacitou položek *limit*.

Kaskádu tvoří v následujícím pořadí tyto algoritmy:

1. kolaborativní filtrování na základě uživatele,
2. kolaborativní filtrování na základě položky,
3. sémantické doporučení
4. a doporučení na základě popularity.

Podmínkou pro akceptaci výsledků z algoritmu, který je na řadě, je v současné době ta, že algoritmus byl schopen vygenerovat počet doporučení daný parametrem *limit*. Lze také kontrolovat skóre poslední doporučené položky a akceptovat výsledky pouze při překonání nastavené hranice, která bude pro každý algoritmus jiná, protože míry doporučení mezi algoritmy nelze příliš porovnávat.

Obrázek 4.1 Proces doporučování

Protože z metod implementující kolaborativní filtrování dostáváme položky reprezentované entitami z Neo4J, ale vracíme reprezentace z Django, je potřeba výsledky metod *get_user_based_recommendations* a *get_item_based_recommendations* přemapovat na korespondující typy pomocí funkce *get_django_object_types_from_nodes*.

Implementace obsahuje samozřejmě také ošetřování chyb, které zde nebudu rozvádět, protože se jedná o ošetřování závislé na použité technologii.

Na konci kaskády stojí doporučení na základě popularity, pro něj je zajištěno, že vrací výsledky vždy. Tento typ doporučení není odvozen od chování uživatele a slouží jako záložní doporučení v případě, že nejsou o uživateli známé žádné informace, kterých by se mohly algoritmy výše v žebříčku chytnout.

Popis implementace jednotlivých algoritmů bude popsán v sekci *Popis implementace* u korespondujících algoritmů.

4.3.1 Kolaborativní filtrování jako pilíř doporučování

Tato část se bude zabývat různými tématy včetně toho, jak náš bipartitní graf už vlastně není bipartitní a projekcí do dvou souvisejících grafů. Bude pojednávat také o výpočtu podobností mezi uživateli a položkami na základě jejich interakcí bez použití statických informací a o ukládání těchto podobností v modelu k-NN. Navazující část popíše, jak tyto podobnosti využít k vytváření seznamů doporučení pro uživatele pomocí přístupů založených na uživatelích i položkách, přičemž zohlední i binární a nebinární hodnoty. Kromě toho zdůrazní výhody použití kompaktní verze vektoru, případy, kdy ji použít a poskytne návod k její implementaci.

Na obrázku 4.1 lze vidět základní sekvenci postupů, které jsou klíčové pro implementaci kolaborativního filtrování. Všechny části samozřejmě popíšu.

4.3.1.1 Výpočet podobností/ nejbližších sousedů

V této části bude vysvětleno, jak se počítá a ukládají nejbližší sousedé do grafu, což zahrnuje sdílení uzlů uživatele a položky nedávno vytvořeného bipartitního grafu. To zjednodušuje proces doporučování tím, že je k dispozici jediná datová struktura, která zahrnuje jak preference uživatele tak nejbližší sousedy.

Prvním krokem je výběr funkce podobnosti, která dokáže vypočítat vzdálenosti mezi podobnými prvky v grafu, jako jsou uživatelé nebo položky. Běžně se používá kosinová podobnost, ale existují i další metriky, jako je upravená kosinová podobnost, Spearmanův koeficient korelace pořadí, střední kvadratický rozdíl a Pearsonův koeficient, které se rovněž navrhují jako alternativy [3]. Je zdroj [3], který tvrdí, že doporučení založená na uživatelích je Pearsonův koeficient lepší než ostatní míry, zatímco kosinová podobnost je nejlepší pro doporučení založená na polož-

Obrázek 4.2 Proces výpočtu podobnosti

kách. V našem případě jsem použil k měření podobnosti mezi položkami upravenou kosinovou podobnost, která zohledňuje skutečnost, že různí uživatelé mají různá schémata hodnocením.

$$podobnost(i,j) = \frac{\sum_{u \in U}(r_{u,i} - \overline{r_u}) * (r_{u,j} - \overline{r_u})}{\sqrt{\sum_{u \in U}(r_{u,i} - \overline{r_u})^2} * \sqrt{\sum_{u \in U}(r_{u,j} - \overline{r_u})^2}}$$

Druhý krok zahrnuje reprezentaci každého prvku způsobem, který vyhovuje zvolené funkci podobnosti. Nakonec vypočítáme podobnosti a uložíme je do grafu. [3]

Výpočet podobnosti by se také dal shrnout sekvení viditelnou na obrázku 4.2.

4.3.1.2 Reprezentace elementu jako vektoru pro výpočet podobnosti

Jak bude viditelné na později zmíněných formulích pro výpočet podobnosti, je potřeba uživatele i položky porovnávat jednu s jednou za pohledu na jejich reprezentační vektor.

Již jednou zmíněná tabulka 3.4 obsahuje řádky představující uživatele a sloupce představující položky. Při vytváření vektorů pro účely porovnání je důležité zachovat stejné pořadí sloupců. Příkladové vektory jsou v tomto případě binární neboli booleovské, označují pouze existenci vztahu mezi uživatelem a položkou, aniž by zohledňovaly hodnotu tohoto vztahu. Tuto hodnotu však lze v modelu grafu reprezentovat jako vlastnost, například hodnocení uživatele nebo počet kliknutí na produkt. Pro přechod na nebinární reprezentaci lze jedničku ve vektoru jednoduše nahradit skutečnou váhou vztahu mezi uživatelem a položkou. [3]

Pro představu, při pohledu na tabulku 3.4 lze vyvodit následující. Vektorová reprezentace uživatelů vypadá takto:

1. Vektor pro uživatele A = [9,0,0,0]
2. Vektor pro uživatele B = [10,0,9,0]
3. Vektor pro uživatele C = [0,2,0,0]
4. Vektor pro uživatele D = [0,1,0,2]

Analogicky stejný bude i postup pro získání vektorů položek, jen stačí tabulku transponovat.

Takto použitý vektorový přístup lze použít nejen pro reprezentaci interakce, ale např. vlastnění nebo hodnotu určitých atributů reprezentovaného elementu, kde ukládaný datový typ není Ano, nebo Ne. Nejčastějším myšleným použitím je myšleno to s číselnými hodnotami, jako je například hodnocení položek. Vektorová reprezentace pak může obsahovat přesné hodnoty těchto atributů. Kosinovou vzdálenost mezi vektory lze vypočítat i v případě, že některé složky jsou logické a jiné jsou reálné nebo celočíselné hodnoty. Je však nutné vhodně škálovat jiné než booleovské složky, aby nedošlo k jejich dominanci nebo irelevanci při výpočtu. Toho lze dosáhnout vynásobením hodnot škálovacím faktorem, který může být pro každý číselný prvek jiný a závisí na váze, která je mu ve výsledné podobnosti přiřazena. Nastavením tohoto faktoru u nelogických atributů na hodnotu 1 typicky bude znamenat, že atribut bude mít větší vliv na hodnotu podobnosti, zatímco nastavením na hodnotu 0,5 se jeho vliv sníží na polovinu. [3]

4.3.1.3 Redukce velikosti vektoru

Pro zajištění rychlého výpočtu podobností je důležité reprezentovat vektory způsobem, který šetří paměť a umožňuje efektivní přístup k jejich hodnotám. Nejefektivnější způsob reprezentace vektorů ovšem závisí na typu vektoru a jeho zamýšleném použití. Úroveň řídkosti nebo hustoty vektoru je zásadním faktorem, který je třeba vzít v úvahu, a neexistuje žádná obecná hranice pro rozhodnutí se mezi použitím řídkého a hustého typu vektoru jako reprezentace. [3]

Vektor, který má malé procento nenulových hodnot, se nazývá řídký vektor. Ukládání takových vektorů do pole není efektivní z hlediska využití paměti a výpočetních nákladů. Existují však alternativní způsoby reprezentace řídkých vektorů, které mohou optimalizovat využití paměti a zjednodušit manipulaci.

Zatímco hustý vektor obsahuje mnoho nenulových hodnot v poměru ke své velikosti. Indexy hustého vektoru odpovídají přímo jeho indexům v poli. Hlavní výhodou hustého vektoru je jeho rychlost; lze k němu přistupovat a aktualizovat jej rychle, protože je podložen polem. [3]

V našem případě jsou vektory často velice řídké a nebylo by vhodné je reprezentovat přímou formou v poli. Místo toho jsem zvolil vlastní kompaktní formát, který v Pythonu jde efektivně použít pro porovnávání vzájemných elementů.

```
1     uzivatelB.interakce = {'polozkaA': 10, 'polozkaC': 9}
```

Tento kompaktní formát je ve tvaru slovníku, kde klíčem je identifikátor položky a hodnotou je jednička, nebo nula v případě interakce, která nemá explicitně více hodnot jako např. zobrazení položky. Hodnotou také mohou být libovolné jednotky jako v tomto ilustrativním případě, kde je hodnotou číslo hodnocení oblíbenosti položky. Analogicky stejná forma je použitá pro položky místo příkladu, který ukazuje uživatele. Největší výhodou tohoto formátu je, že k porovnávání ukládáme pouze potřebné informace, na konkrétní podobě čili myšleno jestli je to slovník, nebo chytře organizované pole jako např. |počet nenulových elementů|indexy nenulových elemntů|hodnoty nenulových elementů|, na tom příliš nezáleží. Skutečností totiž je, že když velikost jedné z dimenzí matice interakcí dostane do řádu sto tisíců nebo miliónů, je v podstatě časově neúnosné pracovat s těmito daty jinak než v takto kompaktním formátu.

Jak se vypadá samotná implementace a jak se provádí samotné porovnávání uvedu v následující sekci, která se věnuje vytvářením a aktualizaci podobností v grafu.

4.3.1.4 Implementace výpočtu podobnosti

Je důležité zmínit, že přes vychvalování vlastností bipartitního grafu výše, tak po uložení podobností do grafu vlastně již graf nebude bipartitním, jak lze vidět na příkladu 3.8. Graf nebude bipartitním grafem, protože budou existovat vztahy mezi prvky ve stejném oddílu.

Můžeme však mít více podgrafů téhož grafu, pokud uvažujeme pouze podmnožinu uzlů a vztahů, takže graf lze rozdělit na tři vysoce relevantní podgrafy:

1. Podgraf s uzly všech uživatelů a všech položek, který obsahuje pouze vztahy interakce, ten je bipartitní graf, který jsme měli dříve.

2. Podgraf s uživateli a vztahem jejich podobností je síť nejbližších sousedů pro uživatele.

3. Podgraf s položkami a vztahem jejich podobností je síť nejbližších sousedů pro položky.

Nyní popíšu, jakým způsobem jsem implementoval výpočet a uložení nejbližších sousedů pomocí následujícího útržku kódu.

```
def create_or_update_similarity():
    self.stdout.write("Creating User Similarity")
    make_user_similarity()
    self.stdout.write("Creating Item Similarity")
    make_item_similarity()

def make_user_similarity():
    dense_vector_returning_query = """
        match (u:User)
        with u
        match (u)-[]->(:Interaction)-[]->(r:Resource)
        with u, apoc.map.fromPairs(collect(distinct ([r.resource_id, 1]))) as
    interactions
        return u{.id, interactions: interactions}
    """
    make_similarity("User", dense_vector_returning_query)

def make_item_similarity():
    dense_vector_returning_query = """
        match (r:Resource)
        with r
        match (u:User)-[]->(:Interaction)-[]->(r)
        with r.resource_id as rsc_id, apoc.map.fromPairs(collect(distinct ([u.id, 1])
    )) as interactions
        return {resource_id: rsc_id, interactions: interactions}
    """
    make_similarity("Resource", dense_vector_returning_query, id_name="resource_id")

def make_similarity(label, dense_vector_returning_query, knn=10, id_name="id"):
    results, _ = db.cypher_query(dense_vector_returning_query)
    for first_node in results:
        first_node = first_node[0]
        first_node_similarities = []
        for second_node in results:
            second_node = second_node[0]
            if first_node[id_name] < second_node[id_name]:
                similarity_val = get_cosine_similarity(first_node["interactions"],
    second_node["interactions"])
                if similarity_val > 0:
                    first_node_similarities.append({id_name: second_node[id_name], "
    value": similarity_val})
        sorted(first_node_similarities, key=lambda x: x["value"])

        query = f"""
            MATCH (u:{label} {{{id_name}: $first_node_id}})
            UNWIND $sims as sim
            MATCH (u2:{label} {{{id_name}: sim.{id_name}}})
            MERGE (u)-[s:SIMILAR]->(u2)
            SET s.value = toFloat(sim.value);
            """
        db.cypher_query(query, {"sims": first_node_similarities[:knn], "first_node_id
    ": first_node[id_name]})
```

Jak jde vidět v ukázce kódu 4.3.1.4 metoda *create_or_update_similarity* spouští proces výpočtu a ukládání nejbližších sousedů. Tato metoda se spouští každý den pro zachování aktuálnosti nejbližších sousedů. Je i možnost vypočítávat podobnosti sousedů v reálném čase. Takto časté výpočty ovšem můžou být velice náročné pro databázi a můžou zpomalovat časy odpovědí pro jiné typy dotazu, které běží v tu samou dobu. Proto bylo zvoleno periodické aktualizování podobností, navíc zvolenou frekvenci lze vždy upravit podle potřeb pro aktuálnější doporučení.

Funkce *make_user_similarity* resp. *make_item_similarity* slouží pro zápis Cypher dotazu, který získá vektor interakcí reprezentující uživatele resp. položku pro všechny uživatele resp.

položky, jak bylo popisováno výše. Formát získaného vektoru v tomto případě vypadá jinak, než kompaktní formát popisovaný výše, ale ve skutečnosti je to on, jen je zabalený do mapy, ve které je i identifikátor uživatele. Je tomu proto, aby bylo možné vrátit vektory pro všechny uživatele resp. položky najednou. Tyto funkce pak se svým dotazem volají funkci *make_similarity*, která zařídí výpočet a uložení podobností entit na základě vloženého dotazu.

Funkce *make_similarity* pro všechny získané vektory vypočte podobnost porovnáním všech vektorů se sebou pomocí kosínovy podobnosti, jejíž formule je následující.

$$podobnost(A, B) = \frac{A \cdot B}{||A|| * ||B||} \tag{4.1}$$

Kde podobnost vektoru A a B ve formuli 4.1 je rovna podílu skalárního součinu A a B se součinem druhých norem vektorů A a B.

Pokud vypočtená podobnost přesahuje nastavenou hranici, která je v tomto případě rovna nule, pak se kandidát na nejbližšího souseda uloží do pole *first_node_similarities*. Po porovnání vektoru se všemi ostatními vektory , kteří by mohli být potenciálními nejbližším sousedy, se do databáze vloží podobnost pouze s *knn* nejbližšími vektory reprezentující uživatele, nebo položku s hodnotou podobnosti uloženou v atributu hrany jménem *value*.

Výše uvedený postup vydestiloval z bipartitního grafu čerstvé znalosti a uložil je způsobem, který může sloužit mnoha účelům, kromě poskytování doporučení. Mezi tyto účely patří mimo jiné shlukování položek, které může pomoci identifikovat skupiny položek, které se často kupují či sledují společně. Kromě toho lze tytéž algoritmy shlukování aplikovat na síť nejbližších sousedů uživatelů, což vede k segmentaci uživatelů, kteří obvykle interagují se stejnými položkami. A konečně, samotná síť nejbližších sousedů položek je přínosná při hledání podobných položek, dotazováním grafu je pak možné vygenerovat seznam podobných produktů na základě vztahu podobnosti rychlým způsobem. [3]

4.3.1.5 Vrácení doporučení

Proces tvorby doporučení generuje dva hlavní typy výsledků:

1. skóre relevance, což jsou číselné předpovědi udávající míru preference nebo nezájmu aktuálního uživatele o určitou položku,
2. a doporučení, což je seznam N navrhovaných položek. Tento seznam by neměl obsahovat položky, které uživatel již zakoupil, pokud se nejedná o opakované nákupy.

Rozhodnout se pak lze zda zvolit přístup založený na uživateli, nebo na základě položky. Obě varianty vracejí mírně rozdílné výsledky a hodí se spíše v jiné případy, který popíšu v jejich specifických odstavcích.

4.3.1.5.1 Skóre založené na uživateli Přístup založený na uživateli zahrnuje použití aktuálního uživatele, datasetu interakcí reprezentované bipartitním grafem a sítě nejbližších sousedů k vytváření předpovědí o položkách, se kterými uživatel ještě neinteragoval. Předpovědi jsou založeny na hodnoceních, která těmto položkám udělili ostatní uživatelé ve stejné síti nejbližších sousedů. [3]

Cílem přístupu založeného na uživatelích je najít podobné uživatele, kteří interagovali se stejnými položkami a poté jim navrhnout další produkty, se kterými tito podobní uživatelé již interagovali, ale cílový uživatel ještě ne. Tyto metody vycházejí z předpokladu, že uživatelé s podobným vkusem v minulosti budou mít podobný vkus i v budoucnosti a že preference uživatelů zůstanou v čase stabilní a konzistentní. [3]

Podle okolností lze výpočet skóre doporučení provést následujícími případy:

1. doporučení bez váhy,
2. nebo doporučení s váhou

Doporučení bez váhy V případě, kdy typy interakcí mezi uživateli a položkami neobsahují váhu, ale pouze binární informaci, jako zda uživatel kliknul, nebo se podíval na danou položku, mluvíme o doporučení bez váhy. V takovém případě míra doporučení položky p uživateli a je rovná následující formuli 4.2.

$$skore(a,p) = \frac{1}{|KNN(a)|} * \sum_{b \in KNN(a)} r_{b,p} \tag{4.2}$$

Kde $r_{b,p}$ je jednička, nebo nula pokud uživatel b interagoval s položkou p, nebo ne. *KNN(a)* znamená množinu nejbližších sousedů uživatele a. Takže suma označuje počet nejbližších sousedů, kteří s položkou interagovali. To je normalizováno počtem nejbližších sousedů.

Doporučení s váhou V případě, kdy typy interakcí mezi uživateli a položkami obsahují váhu, jako nejtypičtěji hodnocení oblíbenosti položky uživatelem, mluvíme o doporučení bez váhy. V takovém případě míra doporučení položky p uživateli a je rovná následující formuli 4.3.

$$skore(a,p) = \frac{\sum_{b \in KNN(a)} sim(a,b) * r_{b,p}}{\sum_{b \in KNN(a)} sim(a,b)} \tag{4.3}$$

Kde *KNN(a)* představuje k nejbližších sousedů uživatele a, $r_{b,p}$ představuje hodnocení, které uživatel b přidělil položce p. Tato hodnota samozřejmě může být nulou v případě, že uživatel b nehodnotí položku p. Existují některé varianty tohoto vzorce, které jsou popsány v této knize [3], ale formule uváděny zde jsou přijímány za jedny z nejoptimálnějších prácemi, které referuje také tato kniha.

Popis implementace Nyní popíšu, jakým způsobem jsem implementoval kolaborativní filtrování na základě uživatele pomocí následujícího útržku kódu.

```
def get_user_based_recommendations(user_id, k):
    query = """
        MATCH (:User {id: $userId})-[:SIMILAR]-(u2:User)
        with u2, count(u2) as size
        MATCH (u2:User)-[:Made]->(:Interaction)-[:On]->(r:Resource)
        WHERE NOT EXISTS((:User {id: $userId})-[:Made]->(:Interaction)-[:On]->(r))
        AND r.public = true AND r.deleted is null
        with r, (1.0 / size) * count(distinct u2) as score
        return r order by score desc
        """
    if k >= 0:
        query += "limit $k"

    results, _ = db.cypher_query(query, {'userId': user_id, 'k': k}, resolve_objects=
    True)
    objects = inflate_all(results, (lambda x: x))

    return objects
```

Jak jde vidět v ukázce kódu 4.3.1.5.1 metoda *get_user_based_recommendations* vrací doporučené položky pro uživatele s identifikátorem *user_id* s maximálním počtem položek *k*. Tato metoda vrací doporučení pomocí kolaborativního filtrování na základě uživatele a při doporučení se nebere ohled na váhu interakcí.

Doporučení je založeno na Cypher dotazu do databáze Neo4J se schématem popisovaným v této práci a v mé bakalářské práci. Tento databázový dotaz vyhledává položky, které jsou mu dostupné a se kterými interagovali uživatelé podobní cílovému uživateli, ale ne on samotný, a vrací je seřazené dle skórovací funkce popsané výše. Je možné také volitelně ovlivnit počet záznamů vrácených dotazem dle zvolené hodnoty parametru *k*.

Funkce *inflate_all* slouží akorát ke formátování výstupu, aby metoda vrátila z výsledku dotazu pouze objekty položek.

4.3.1.5.2 Skóre na základě položky Pokud jde o předpovídání podobných sousedů pro rozsáhlou datovou základnu v reálném čase, tak to je při použití grafového přístupu velice náročné vzhledem k obrovskému počtu potenciálních sousedů, které je třeba prověřit. V důsledku toho se používají různé techniky, jako je například doporučování na základě položek. Tato technika umožňuje výpočet doporučení v reálném čase i pro velkou matici hodnocení.

Při použití algoritmů založených na položkách se při vytváření předpovědí zaměřujeme spíše na hledání podobností mezi položkami než na uživatele. V tomto přístupu jsou doporučení založena na interakcích, které uživatelé provedli nad podobnými položkami. Pokud např. cílový uživatel udělil vysoké hodnocení první a druhé položce, algoritmus založený na položkách vypočítá vážený průměr těchto hodnocení a předpoví hodnocení nové položky, které se nachází někde mezi hodnocením první a druhé položky. [3]

Doporučení bez váhy Stejně jako v případě skóre založené na uživateli, kdy typy interakcí mezi uživateli a položkami neobsahují váhu, ale pouze binární informaci, mluvíme o doporučení bez váhy. V takovém případě míra doporučení položky p uživateli a je rovná následující formuli 4.4.

$$skore(a,p) = \sum_{q \in interactedItems(a)} sim(p,q) * |KNN(q) \cap \{p\}| \tag{4.4}$$

V případě skóre na základě položky není cílem předpovědět nutně hodnocení cílovým uživatelem, ale předpovědět, do jaké míry bude položka pro cílového uživatele zajímavá. U doporučení bez váh v tomto případě a taky jako v případě přístupu založeného na uživateli, neexistuje žádný uznávaný standardní přístup k výpočtu hodnocení, proto zde uvádím přístup, který je jednoduchý a efektivní. Je pak vhodné mít na paměti, že formule nevrací predikci, ale spíše obecné skóre.

Doporučení s váhou Stejně jako v případě skóre založené na uživateli, kdy typy interakcí mezi uživateli a položkami obsahují váhu, mluvíme o doporučení bez váhy. V takovém případě míra doporučení položky p uživateli a je rovná následující formuli 4.5.

$$skore(a,p) = \frac{\sum_{q \in interactedItem(a)} sim(p,q) * r_{a,q} * |KNN(q) \cap \{p\}|}{\sum_{q \in interactedItem(a)} sim(p,q) * |KNN(q) \cap \{p\}|} \tag{4.5}$$

Kde *ratedItems(a)* představuje množinu položek, se kterými uživatel a interagoval, *sim(p, q)* představuje hodnotu podobnosti mezi položkou q a cílovým položkou p, $r_{a,q}$ je hodnocení, které uživatel přiřadil q a */KNN(q) p/* je jedničkou, nebo nulou pokud je p v množině nejbližších sousedů q, nebo není. To znamená, že je možné uvažovat pouze nejbližší sousedy q, nikoli všechny podobnosti. Jmenovatel pak normalizuje hodnotu tak, aby nepřekročila maximální hodnotu hodnocení.

Popis implementace Teď popíšu, jakým způsobem jsem implementoval kolaborativní filtrování na základě položky pomocí následujícího útržku kódu.

```
def get_item_based_recommendations(user_id, k):
    query = """
        MATCH (n:Resource)-[s:SIMILAR]-(r:Resource)<-[:On]-(:Interaction)<-[:Made]-(u
    :User {id: $userId})
        WHERE NOT EXISTS((u)-[:Made]->(:Interaction)-[:On]->(n))
        AND n.public = true AND n.deleted is null
        with n, sum(s.value) as score
        return n order by score desc
        """
    if k >= 0:
        query += "limit $k"

    results, _ = db.cypher_query(query, {'userId': user_id, 'k': k}, resolve_objects=
    True)
    objects = inflate_all(results, (lambda x: x))

    return objects
```

Jak jde opět vidět v ukázce kódu 4.3.1.5.2 metoda *get_item_based_recommendations* vrací doporučené položky pro uživatele s identifikátorem *user_id* s maximálním počtem položek *k*. Tato metoda vrací doporučení pomocí kolaborativního filtrování na základě položky a při doporučení se nebere ohled na váhu interakcí.

Doporučení je opět založeno na Cypher dotazu do databáze Neo4J se schématem popisovaným v této práci a v mé bakalářské práci. Tento databázový dotaz vyhledává položky, které jsou mu dostupné, se kterými ještě neinteragoval a které jsou podobné položkám, se kterými cílový uživatel již interagoval v minulosti, a vrací je seřazené dle skórovací funkce popsané výše. Je možné také volitelně ovlivnit počet záznamů vrácených dotazem dle zvolené hodnoty parametru *k*.

4.3.2 Sémantické doporučování

Sémantické doporučování je založené na hledání položek s podobným obsahem dle historie posledních interakcí. V databázi má každá položka svůj identifikující vektor, který ji celou shrnuje. Tento vektor je získán pomocí *text embeddingu*, k čemuž slouží vytrénovaný model jménem DistilBERT. Komunita Hugging Face poskytuje použití tohoto modelu k transformaci textu do vektoru čísel. Řetězec textu použitý k získání vektoru je vytvořen spojením atributů položky jako např. pro knihu je to popis knihy a název knihy. Ale pro jiné typy položek jsou to jiné atributy, které tvoří použitý řetězec. Takovéto vektory posledně interagovaných položek se pak zprůměrují běžným matematickým průměrem a hledají se v databázi položky s nejbližším možným vektorem.

Nyní popíšu, jakým způsobem je implementováno sémantické doporučování pomocí následujícího útržku kódu.

```
def get_history_based_material_recommendations(user_id, limit):
    MAX_LAST_VIEWED_ITEMS = 10
    last_viewed_items = get_last_visited_items(user_id, MAX_LAST_VIEWED_ITEMS)

    vectors = list()
    for item in last_viewed_items:
        vector = get_item_descriptor_vector(item)
        if vector is not None:
            vectors.append(vector)

    if len(vectors) == 0:
        return list()

    query_vector = get_mean_vector_from_vectors(vectors)
    return get_similar_items_by_vector(info, query_vector, limit)
```

Jak jde vidět v ukázce kódu 4.3.2 metoda *get_history_based_material_recommendations* vrací doporučené položky pro uživatele s identifikátorem *user_id* s maximálním počtem položek *limit*. Tato metoda vrací doporučení na základě sémantického doporučování popsaného výše. Z listu posledně zobrazených deseti položek vytvoří nový vektor, který je jejich průměrem a poté je poslán dotaz do používaného vyhledávače OpenSearch, což je odnož známější technologie jménem Elasticsearch. Tento vyhledávač udržuje seznam položek i s jejich vektory a poskytuje možnost vyhledávání k nejbližších sousedů. Takový dotaz na zmíněný vyhledávač požaduje vrácení nejbližších sousedů s použitím normy L2 jako funkce k měření vzdálenosti, ta je popsána v této sekci3.5.4.1 i s ostatními alternativami.

4.3.3 Doporučování dle popularity

Doporučování dle popularity je bráno jako záložní možnost pro doporučení položek, která je poskytnuta všem příchozím. Je tomu totiž, protože nehledí na historii jednotlivce/ cílového uživatele, ale bere zřetel na nejpopulárnější položky v poslední době mezi všemi uživateli a ty doporučí.

Teď popíšu, jakým způsobem je implementováno doporučování dle popularity pomocí následujícího útržku kódu.

```
def get_popularity_based_recommendations(k):
    latest_viewed_path_ids = (view[0] for view in
                              PathView.objects.filter(path__public=1,
    path__deleted__isnull=True)
                              .order_by("-created")[:500].values_list("path_id"))

    most_common_paths = []
    for path_tuple in Counter(latest_viewed_path_ids).most_common(k):
        try:
            path = Path.objects.get(id=path_tuple[0])
            most_common_paths.append(path)
        except DoesNotExist as _:
            pass

    return most_common_paths
```

Jak jde vidět v ukázce kódu 4.3.3 metoda *get_popularity_based_recommendations* vrací doporučené položky bez ohledu na dotazujícího se uživatele s maximálním počtem položek k. Tato metoda vrací doporučení na základě doporučování dle popularity popsaného výše, které je uzpůsobené našemu případu použití, kde se účelově vrací pouze vzdělávací cesty tímto doporučováním. Důvody proč se vracejí akorát vzdělávací cesty jsou následující:

1. Proklik na vzdělávací cestu oproti ostatním variantám je výrazně cennější, jelikož cesta obsahuje již množství ostatních položek, které úzce souvisí s prokliknutou cestou.
2. Vzdělávací cesty jsou jedním z hlavního originálního obsahu stránky LearnerOn, který udržuje uživatele na stránce samotné. Ostatní položky jako Odkazy by totiž uživatele vedly ze stránky pryč.

Z listu posledně zobrazených pět seti uživatelům dostupných vzdělávacích cest se vrací k nejčastějších vzdělávacích cest.

4.4 Testy vrácení výsledků

Nyní na přiložených testovacích datech předvedu, jak se doporučení chová pro různé typy uživatelů. Protože se tato diplomová práce tyká kolaborativního filtrování a nechci ztěžovat čtenáři reprodukovatelnost výsledků mých doporučení použitím technologií jako text embedding, které se

netýkají předmětu této diplomové práce, tak uvedu pouze případy, kdy kaskádový mechanizmus doporučování zvolí výsledky kolaborativní filtrování a nouzové varianty v podobě doporučení dle popularity.

K ověření výsledků doporučování lze použít anonymizované produkční data z databáze LearnerOnu, a nebo uměle vytvořené testovací data. Rozhodl jsem se popsat správnost funkčnosti doporučovacího mechanizmu na testovacích datech, které byly vytvořeny k přehledné demonstraci výsledků doporučení. Hlavním důvodem pro toto rozhodnutí je skutečnost, že tímto způsobem mohu zajistit, aby výsledky jednoznačně odpovídaly předpokladům o fungování použitých algoritmů. Není tak potřeba odfiltrovat anonymizované účty uživatelů, které lidé použili pouze pro náhodné procházení materiálů, a mohly by znejednoznačnit původ doporučení při vysvětlování funkčnosti v této práci.

Také chci poukázat na to, že problém studeného startu není příliš velkou překážkou pro správnou funkčnost, i když se operuje s relativně malou maticí interakcí uživatelů a položek, když existují velice populární položky, kolem kterých se agreguje znatelná část uživatelů.

Testovací data jsem si nechal vygenerovat OpenAI chatbotem ChatGPT, konkrétně modelem *text-davinci-002*. Nechal jsem si vygenerovat kurzy jakožto doporučované položky. Kurzy oproti odkazům, vzdělávacím cestám, nebo knihám samy o sobě více vypovídávají, o čem jsou a o co uživatel s nimi interagující se zajímá. V příkladu tedy použiji pouze kurzy jako položky, protože jejich doporučování je jednodušší pro pochopení. Kurzy jsem si nechal vygenerovat pomocí následujícího dotazu:

```
generate for me dataset of courses in json format and add integer identifier as a
 field to every course
```

Uživatele jsem si nechal vygenerovat pomocí následující dotazu:

```
now generate me users in json format
```

Zkoušel jsem i jiné způsoby generování dat jako např. Mockaroo, ovšem tento ChatGPT chatbot byl schopen nejlépe vygenerovat kontrolované data, které dávají smysl a mohly by být daty v reálném použití.

Nechal jsem si vygenerovat přes 150 kurzů, v přiloženém souboru i v příkladu je jich použito přesně 168, některé kurzy vygenerované chatbotem jsem vymazal, abych se zbavil duplicitních záznamů. Některé kurzy jsem také odstranil, protože tématicky ležely naprosto mimo všechny ostatní. Tento počet jsem vyhodnotil jako dostačující, jelikož jsem byl schopen identifikovat v každé z později zmíněných kategorií alespoň 20 kurzů, které se tématicky blíží jednomu zájmu, který člověk může mít.

Uživatelů jsem si nechal vygenerovat přesně 9. Je to protože chci uživatele, na kterém budu moct demonstrovat případ, kdy nelze doporučit položky na základě historie a je potřeba využít doporučení dle popularity, které vrací výsledky společné por všechny uživatele. Poté zbývá osm uživatelů, potřeboval jsem vytvořit shluky uživatelů s podobným zájmem a chci demonstrovat výsledky doporučení pro více než jednu skupinu uživatelů, proto jsem se rozhodl vygenerovat si uživatele pro dvě skupiny o čtyřech členech. V týmu jsou čtyři členové namísto menšího počtu proto, aby při doporučování kolaborativním filtrováním existovalo pro každého uživatele více kandidátu pro nejbližší sousedy, a tím se mohly vrátit lepší výsledky doporučení.

Valná většina vygenerovaných kurzů se tématicky pohybuje v oboru IT nebo byznysu. Proto jsem identifikoval následující obory zájmů, které měly reprezentativní počet kurzů. Těmito obory zájmů jsou:

1. programování v jazyce Python,
2. marketing,
3. hluboké učení
4. a vývoj webových aplikací.

Na základě těchto zájmů jsem pak uživatelům přiřadil interakce, které jsou taky uložené v datovém souboru s uživateli. Uživatele jsem shromáždil do následujících zájmů, tak aby existovalo mezi nimi propojení interakcemi s položkami z jednoho z výše zmíněných oborů zájmů.

1. Uživatelům s identifikátorem číslo 1, 2, 3, 4 jsem přidal interakce s položkami ohledně programování v jazyce Python.

2. Uživatelům s identifikátorem číslo 5, 6, 7, 8 jsem přidal interakce s položkami ohledně marketingu.

3. Uživatel s identifikátorem číslo 9 jsem nepřidal žádné interakce, ten reprezentuje úplně nového uživatele na stránce.

Oba soubory dat jsou přiložené k práci, kde *users.json* je soubor s uživateli použitými v tomto příkladu a *courses.json* je soubor se všemi kurzy.

4.4.1 Naplnění Neo4J databáze testovacími daty

Za běžných okolností by se naplnění grafové databáze provedlo pomocí dávkové synchronizace s relační databází. Ale uvedu zde i Cypher dotaz, kterým se dá naplnit databáze přímo z přiložených souborů bez nutnosti zprovozňování ostatní infrastruktury jako je relační databáze, nebo Opensearch.

Nejprve je ale potřeba do Neo4J nainstalovat plugin jménem *APOC*. Ten přidává spoustu funkcionalit, které samotná databáze nemá, my použijeme funkci *apoc.load.json*, která umí načítat data z přiloženého JSON souboru. K povolení načítání ze souboru je potřeba ve složce s konfiguračními soubory Neo4J vytvořit, pokud neexistuje, konfigurační soubor pro knihovnu APOC jménem *apoc.conf* a do něj vložit následující nastavení:

```
    apoc.import.file.enabled=true
```

Soubory, které bude načítat Cypher dotaz lze popsat lokální cestou ze složky pro importované soubory jménem *import*, nebo absolutní cestou. V mém případě jsem přes vrácené chyby nemohl na soubor odkázat lokální cestou, takže jsem v přiloženém dotazu na ně referoval globální cestou. K tomu je ovšem nutné do konfiguračního souboru *apoc.conf* přidat následující nastavení:

```
    apoc.import.file.use_neo4j_config=false
```

Samotné dotazy pro naplnění databáze, které splňují schéma definované v mé bakalářské práci [1], vypadají následovně.

```
CALL apoc.load.json("file:///D:\\path-to-file\\courses.json")
YIELD value as json_course
MERGE (c:Resource:BaseCourse:Course:LearningItem {
    id: json_course.id,
    resource_id: json_course.resource_id,
    title: json_course.title,
    description: json_course.description,
    instructor: json_course.instructor,
    price: json_course.price,
    duration: json_course.duration,
    level: json_course.level,
    public: json_course.public
});
```

```
CALL apoc.load.json("file:///D:\\path-to-file\\users.json")
YIELD value as json_user
MERGE (u:User {
    id: json_user.id,
    name: json_user.name,
    email: json_user.email
})
WITH u, json_user
UNWIND json_user.viewed_courses AS course_id
MATCH (c:Course {id: course_id})
CREATE (v:Interaction:View)
MERGE (u)-[:Made]->(v)-[:On]->(c);
```

Jak jde vidět v ukázce prvního dotazu 4.4.1, pomocí funkce *apoc.load.json* načítáme soubor s kurzy a ukládáme je rovnou do databáze. V ukázce druhého dotazu 4.4.1 se opět stejným způsobem načítá soubor s uživateli, ale navíc se vytváří propojení mezi uživatelem a kurzy, se kterými interagoval.

Ukázka druhého dotazu 4.4.1 obsahuje dodatečné operace, abychom splňovali mnou sestavený ontologický model z mé bakalářské práce [1]. Na obrázku 4.3 jde vidět extrakt ontologického modelu, který obsahuje námi testovanou část. Tato část obsahuje entitu *Uživatele*, *Interakci*, její dva poddruhy *Privátní interakci*, kterou může provést pouze přihlášený uživatel, a *Zobrazení*, což je charakterizace interakce. Předmětem *Zobrazení* je entita jménem *Prohlídnutelná položka*, mezi které patří i *Kurz*.

Tímto schématem se musí řídit i model v grafové databázi, proto je potřeba v ukázce druhého dotazu 4.3 postupovat dle transformačních postupů vypsaných v mé bakalářské práci [1]. Výstupem pak je graf s modelem na obrázku 4.4. Jak lze po transformaci vidět, tak grafová forma oproti ontologickému modelu je výrazně čitelnější a přehlednější. Hlavním důvodem je možnost skládání množství štítků (labels) do jednoho vrcholu, kde například každý vrchol označující kurz obsahuje štítky Course, LearningItem a Resource. Může obsahovat i štítek PrivateCourse, PublicCourse, nebo BaseCourse, pokud je daný kurz privátní, veřejný, nebo bez identifikace. O dalších vlastnostech se lze více dočíst v mé bakalářské práci [1].

A proto je potřeba v ukázce druhého dotazu 4.4.1 vytvořit vrchol reprezentující interakci zobrazení a napojit jej na uživatele a zobrazený kurz.

Je důležité zmínit, že model grafu na obrázku 4.4 je podgraf skutečného výsledku dotazu na schéma grafu. Je tomu kvůli tomu, že Neo4J při zobrazení schématu databáze, zobrazuje všechny štítky jako vrcholy. My ovšem, jak jsem avizoval výše, typicky skládáme více štítků dohromady, které tvoří jistou hierarchii, která slouží k jednoznačnému popisu entity ale také k možnosti lepšího filtrování entit pomocí Cypher dotazů.

Takové schéma pak je velice nepřehledné a nepřináší žádnou novou informace, když je nám známa použitá hierarchie viz. obrázek 4.5, který reprezentuje ten samý graf jako model na obrázku 4.4, proto zde uvádím jako model grafu ve skutečnosti model podgrafu. Uváděný podgraf jsem získal následujícím Cypher dotazem.

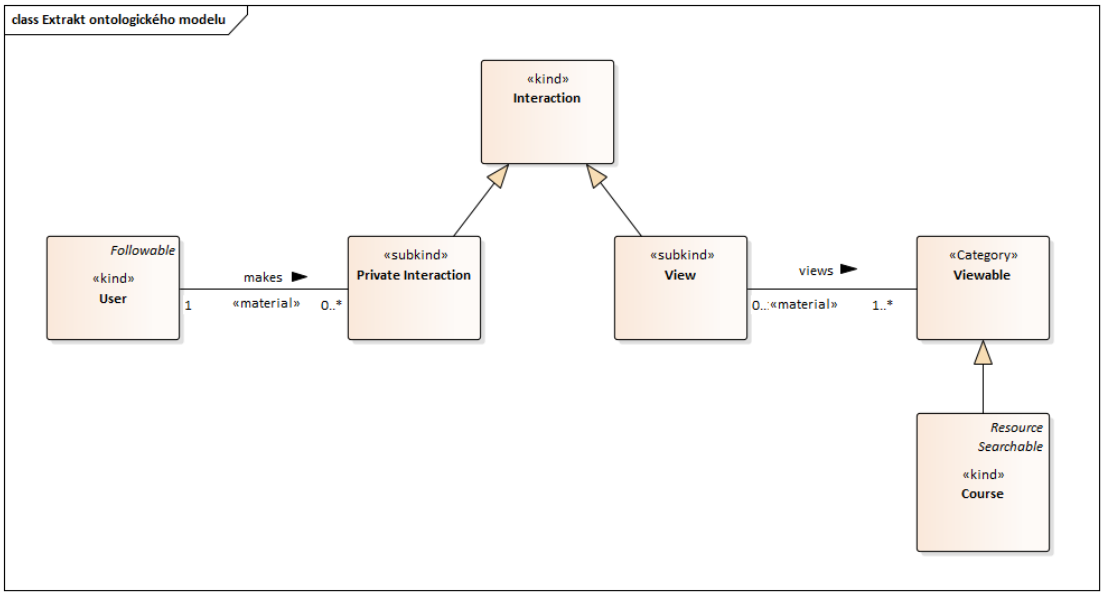
```
CALL apoc.meta.subGraph({
  includeLabels: ["User", "Interaction", "Course"]
});
```

Po naplnění grafové databáze kurzy a uživateli vypadá databáze, jak je ukázáno na obrázku 4.6. Kde růžové vrcholy označují kurzy, červené vrcholy jsou interakce zobrazení a modré vrcholy jsou uživatelé. A veškeré vazby se řídí modelem na obrázku 4.4.

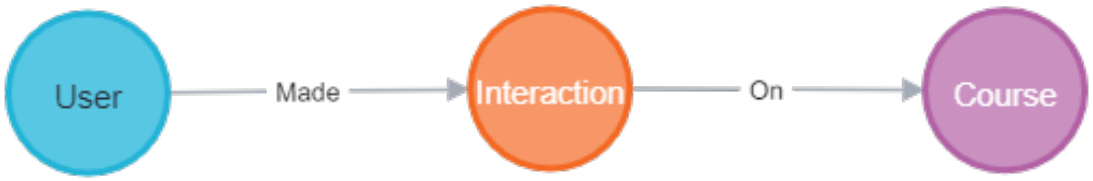
Tuto vizualizaci jsem získal následujícím dotazem.

```
MATCH (n) RETURN n;
```

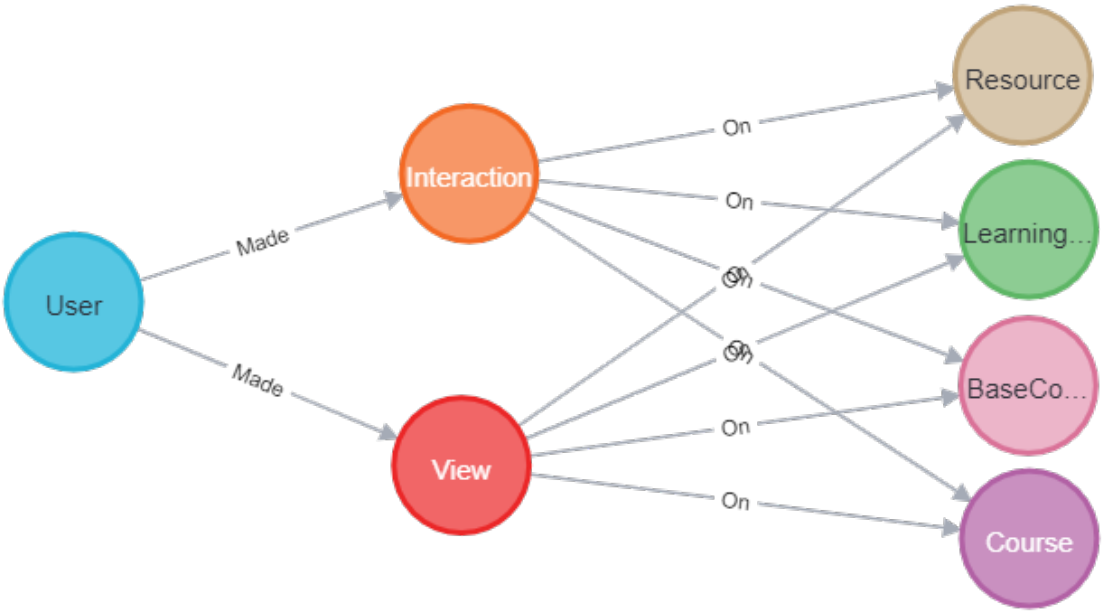
Posledním krokem přípravy databáze před doporučováním je výpočet nejbližších sousedů, k tomu byl použit postup přesně, jak je zmíněno v sekci 4.3.1.4. Po naplnění databáze hranami mezi nejbližšími sousedy, vrcholy kurzů a uživatelů vypadají, jak je ilustrováno na obrázku 4.7.



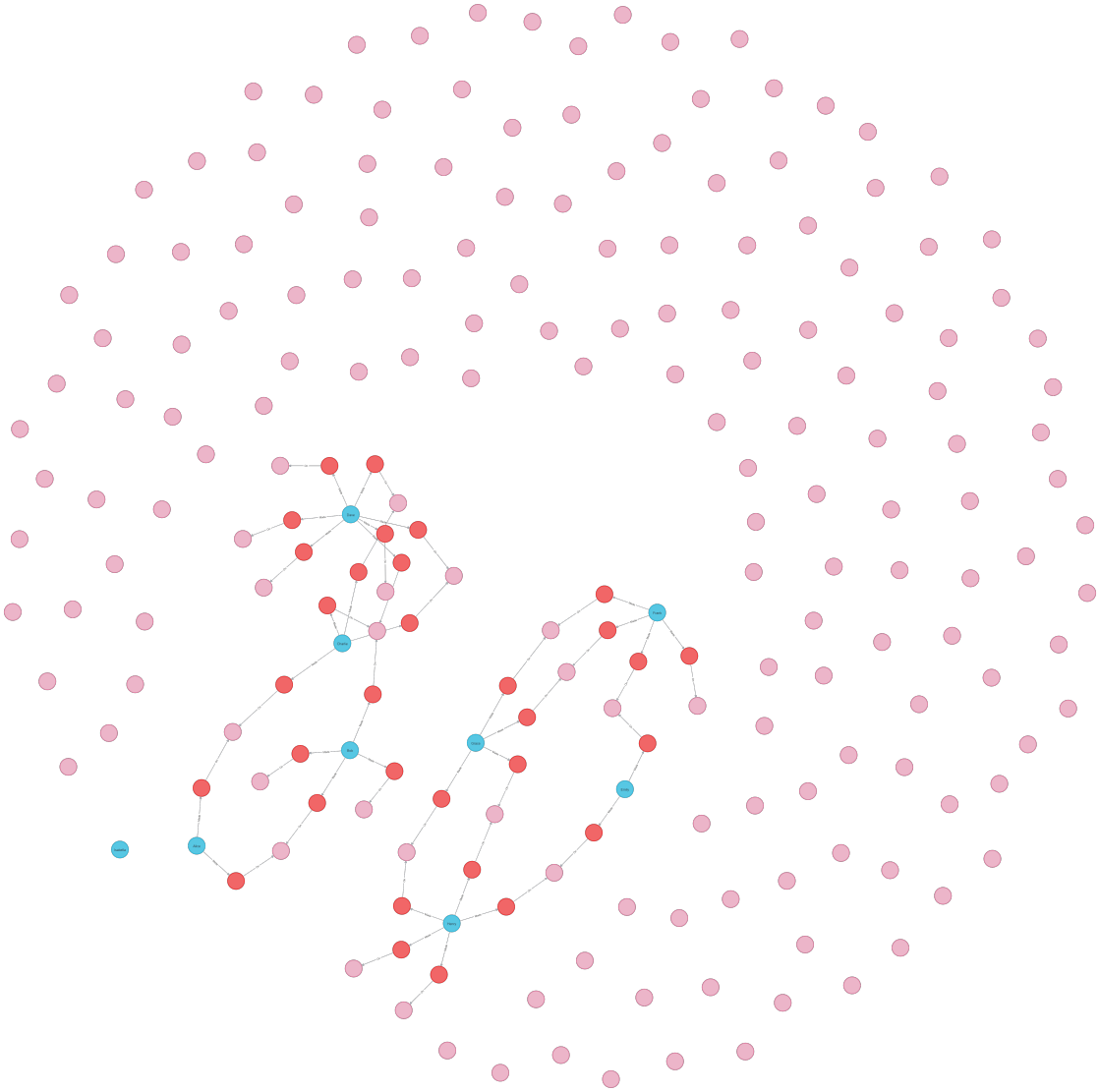
Obrázek 4.3 Extrakt ontologického modelu



Obrázek 4.4 Model grafu před přidáním nejbližších sousedů



Obrázek 4.5 Neupravený model grafu před přidáním nejbližších sousedů



Obrázek 4.6 Databáze po naplnění a kurzy a uživateli

Kde růžové vrcholy označují kurzy, modré vrcholy jsou uživatelé a vazby označují podobnost s nejbližším sousedem.

Tuto vizualizaci jsem získal následujícím dotazem.

```
MATCH (n) WHERE n:Course OR n:User RETURN n;
```

Výsledný grafový model je pak zobrazen na obrázku 4.8. Stejně jako v případě obrázku 4.4, tak i v tomto případě se jedná o podgraf ze stejného důvodu, který jsem opět získal stejným Cypher dotazem 4.4.1.

4.4.2 Ukázka výsledků doporučení

Při použití grafové databáze naplněné s přiloženými daty a způsobem jak bylo popsáno výše, lze spustit kaskádu doporučení, jejíž implementace byla popsána v této sekci 4.3.

Při dotazu na doporučení pěti položek pro uživatele s identifikátorem číslo jedna, který patřil do shluku uživatelů se zájmem pro programování v jazyce Python, dostaneme výsledek vygenerovaný kolaborativním filtrováním na základě uživatele. Výsledkem doporučení jsou položky s id: 272, 240, 256, 279, 286. Ty reprezentují v pořadí tyto kurzy:

1. Data Analysis with Python
2. Introduction to Data Science (in Python)
3. Advanced Python Programming
4. Machine Learning with Python
5. Advanced Python Programming (jiný kurz se stejným jménem)

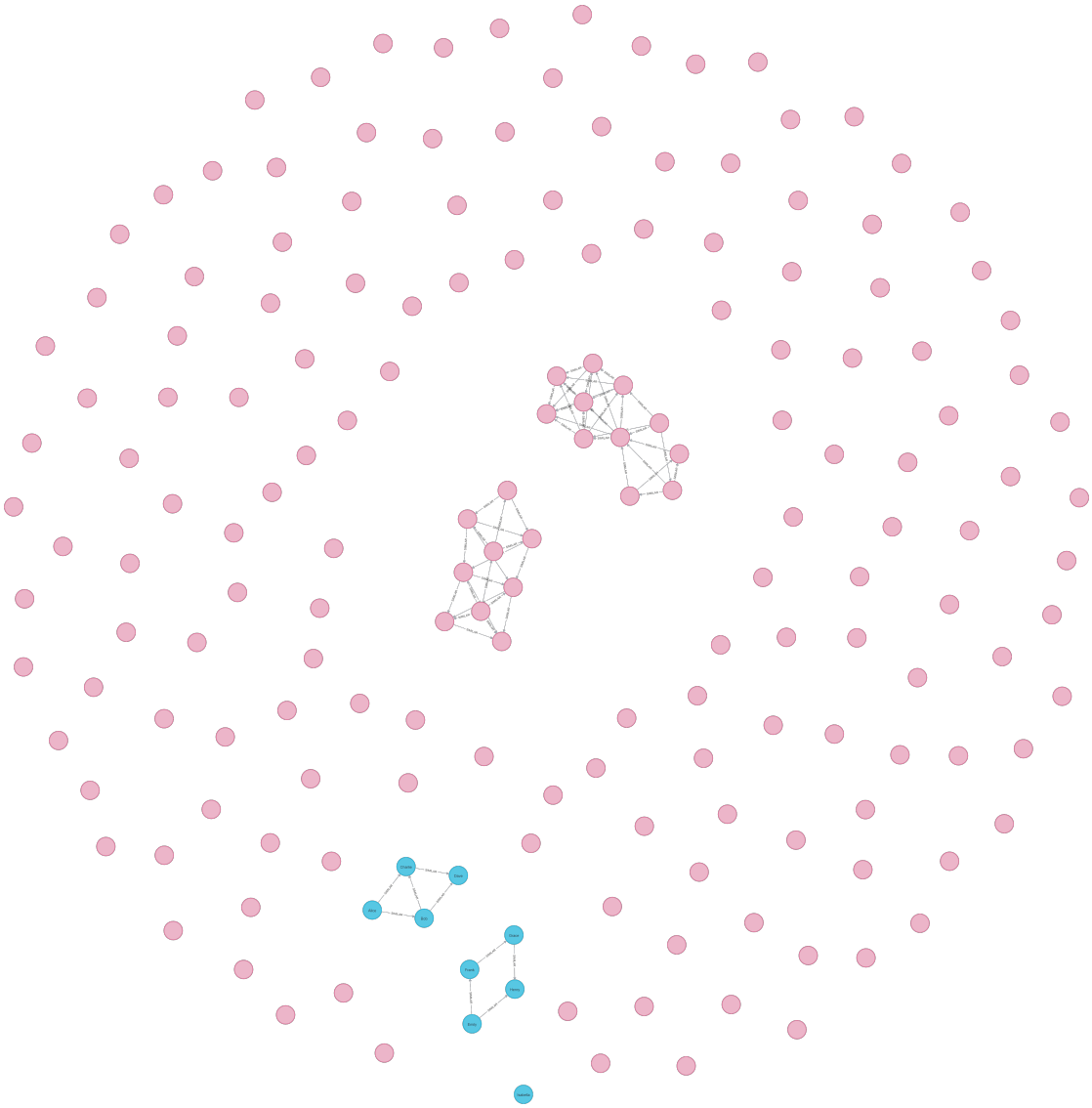
Jak jde vidět při pohledu na data-set interakcí uživatelů se stejným zájmem, byly tomuto uživateli doporučeny položky, se kterými on ještě neinteragoval, ale ty se kterými interagovali uživatelé mu podobní. Což je přesně to, co se od výsledků předpokládá.

Při dotazu na doporučení pěti položek pro uživatele s identifikátorem číslo pět, který patřil do shluku uživatelů se zájmem pro marketing, dostaneme výsledek vygenerovaný kolaborativním filtrováním na základě uživatele. Výsledkem doporučení jsou položky s id: 324, 371, 375, 300, 362. Ty reprezentují v pořadí tyto kurzy:

1. Digital Marketing
2. Data Science for Business
3. Marketing Analytics
4. Introduction to Digital Marketing
5. Digital Marketing

Jak jde vidět při pohledu na data-set interakcí uživatelů se stejným zájmem, byly tomuto uživateli doporučeny položky, se kterými on ještě neinteragoval, ale ty se kterými interagovali uživatelé mu podobní. Což je přesně to, co se od výsledků předpokládá.

Při dotazu na doporučení pěti položek pro uživatele s identifikátorem číslo devět, který nepatřil do žádného shluku uživatelů se sdíleným zájmem, protože mu nebyly přiřazeny žádné interakce, dostaneme výsledek vygenerovaný doporučením na základě popularity. Výsledkem doporučení jsou pak položky s nejvíce interakcemi. Jsou jimi kurzy s id: 272, 240, 220, 216, 279.



Obrázek 4.7 Databáze po naplnění hranami nejbližších sousedů

Obrázek 4.8 Model grafu databáze po naplnění hranami nejbližších sousedů

Ty reprezentují v pořadí tyto kurzy:

1. Data Analysis with Python
2. Introduction to Data Science
3. Advanced Python Programming
4. Natural Language Processing with Python
5. Machine Learning with Python

Při takto malém data-setu, jaký je použit v tomto příkladu, tak lze jednoduše pochopit výsledky tohoto doporučení. Jedná se totiž především o položky, se kterými interagují dvě výše zmíněné skupiny uživatelů, takovéto položky se staly těmi, se kterými se nejvíce interaguje (jsou nejpopulárnější), a proto byly vráceny tímto algoritmem.

Implementace této varianty v LearnerOnu se dotazuje na relační databázi, kterou jsem v této práci příliš nezmiňoval. Nicméně stejné funkcionality lze docílit následujícím Cypher dotazem.

```
MATCH (c:Course)<-[:On]-(i:Interaction)
WITH c, count(i) as freq
RETURN c ORDER BY freq DESC LIMIT 5
```

Čistě pro ilustraci bych zde i rád ukázal, jak vypadá výsledná implementace na portálu LearnerOnu. Panel na obrázku 4.9 obsahuje personalizované doporučení pro mě, pisatele této práce. Tento panel se uživateli zobrazí na levé části stránky vedle hlavního obsahu stránky. Panel se objevuje pro uživatele na jeho stránce stránce profilu, v hubu (domovské stránce), v seznamu uživatelových materiálu, ale i na stránce ke správě uživatelovy kariéry.

Doporučené materiály jsou určeny kolaborativním filtrováním na základě uživatele. Mezi doporučenými položkami jsou:

LearnerOn Dev Stack: Frontend je vzdělávací cesta, která se zabývá principy a technologiemi použitými ve frontendové části portálu LearnerOn.

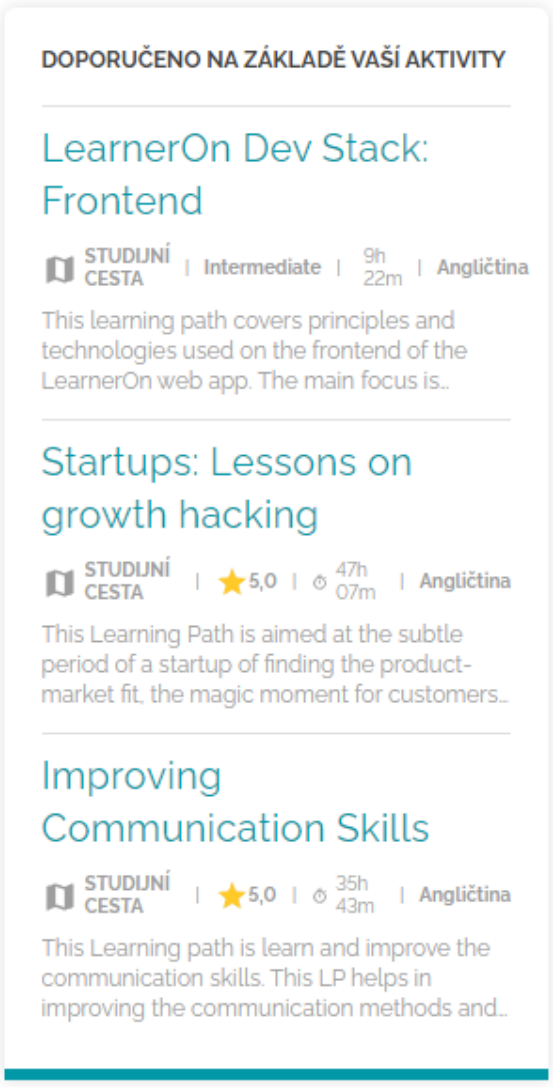
Startups: Lessons on growth hacking je vzdělávací cesta, která mluví o správných hodnotách a cílech při zakládání startupu.

Improving Communication Skills je vzdělávací cesta, která se zabývá zlepšením komunikačních schopností.

Všechno to jsou položky, které jsem ještě neviděl a byly vytvořeny nebo prohlídnuty členy ze stejné *společnosti*, stejného *týmu*. Nebo jenom někým, kdo náhodou se mnou sdílí zájmy, jako v případě třetího doporučené položky. Pojem *společnost* a *tým* v tomto případě reprezentuje uskupení uvnitř portálu, které firmy mohou tvořit k organizaci svých zaměstnanců, přiřazování profesí, materiálů ke vzdělání atd.

Tato část se zabývala především prokázáním funkčností algoritmů popsaných v teoretické i praktické části. Druhou věcí ovšem je kalibrace tohoto mechanizmu, aby vracela výsledky co nejuspokojivější pro uživatele. Na konkrétní kalibraci hranic u jednotlivých algoritmů, kdy doporučené položky ještě lze považovat za dostatečně dobré a kdy už ne, neexistuje jednoznačná metoda. Lze typicky využít kvalitativního testování uvnitř týmu, nebo pomocí uspořádání workshopu na konferencích, nebo s potenciálním zákazníkem. Je také možné vytvořit samo zdokonalující se mechanizmus, který na základě přímé, nebo nepřímé zpětné vazby bude upravovat zmíněné hranice. Varianta testování jako je A/B testování manuálně nastavenými hodnotami je také možnost zlepšení výsledků doporučení, ale jistě se nejedná o způsob jak kontinuálně vylepšovat doporučování, pokud stojí na manuální kalibraci a relativně malé uživatelské základně.

Verze implementace popsána v této práci se příliš nezabývá tímto aspektem, který rozhodně není nepodstatný. Současná verze, jak jde vidět v útržku kódu 4.3 nastavuje hranici pro akceptaci



Obrázek 4.9 Panel s doporučeními na portálu LearnerOn

výsledků, tak že vyžaduje po algoritmu vrátit minimálně tolik výsledků, kolik se vyžaduje po doporučovacím mechanizmu jako celém. To je relativně nízko nastavená hranice, kterou je jistě v plánu pomyslně povýšit. A rozprava nad tím, co dalšího má LearnerOn v plánu pro zlepšení doporučování, je předmětem další sekce.

4.5 Rozprava nad zlepšením procesu doporučování v budoucnu

V týmu LearnerOn padlo několik potenciálních nápadů pro budoucí rozvoj doporučovacího procesu, nyní popíšu ty nejzajímavější:

1. rozšíření výsledků doporučení o jedné nebo více položek, které jsou nutně zdarma,
2. vrácené položky budou pouze v jazycích vybraných uživatelem,
3. mechanizmus, který na základě přímé, nebo nepřímé zpětné vazby bude kalibrovat hranice pro akceptaci výsledků jednotlivých algoritmů z kaskády doporučovacích algoritmů,
4. přidání doporučování založené na aktuální relaci uživatele,
5. a přidání doporučování založené na vlastním modelu vycvičeném strojovým učením

Mezi další možnosti vylepšení by jistě mohl patřit paralelní výpočet výsledků doporučení jednotlivých algoritmů, zkombinování výsledků doporučení různých algoritmů dle navrhnuté váhy výsledků jednotlivých algoritmů nebo další. Možností je často nespočetně a je potřeba zvážit, co přinese největší hodnotu zákazníkovi a zároveň pro firmu LearnerOn. V další části právě popíšu, co je prioritou v nejbližší době.

4.5.1 Popis plánovaného řešení, které rozšiřuje výsledky doporučení

V době psaní této práce se již pracuje na implementaci prvního a druhého bodu. Dle nás se v těchto případech jedná o velice nápomocné možnosti, jak pomoct velice široké bázi uživatelů. Kde první bod pomůže především lidem, kteří chtějí zkusit něco nového bez nutnosti finanční investice a druhý bod pomůže především lidem, kteří nepreferují vzdělávání v anglickém jazyce ale typicky ve svém rodném jazyce. Tato úvaha stojí na tom, že valná většina obsahu na stránce je v angličtině.

Díky rozšiřitelné implementaci synchronizačního mechanizmu mezi relační a grafovou databázi tyto modifikace ani nejsou komplikované. Nyní uvedu přibližný postup práce potřebný k implementaci prvních dvou bodů.

Prvně je potřeba získat informace o ceně a jazyku všech nabízených položek, to je paradoxně nejtěžší část, jelikož především u webových odkazů, které se berou taky jako vzdělávací materiály k doporučení, existuje příliš mnoho variability, aby se dalo jednoznačně určit, jaký jazyk nebo jakou cenu máme odkazu přiřadit. Často se totiž nejedná o odkazy na služby jako YouTube nebo Papers With Code, u kterých lze takové informace jednoduše vyvodit pomocí jimi poskytovaného API, ale často odkazy mohou směrovat na neznámé, nezabezpečené služby, kde ani pomocí samotného odkazu, nelze přesně identifikovat, co na stránce má být obsahem ke vzdělávání.

Poté je potřeba aktualizovat ontologický model, aby stávající entity nově obsahovaly cenu a jazyk jako své atributy. S tím je potřeba upravit modely relační a grafové databáze, kde je potřeba korespondujícím entitám také přidat atributy jako cena, měna a jazyk. Pokud jména atributů jsou stejná v modelech obou databázích, pak není potřeba nic upravovat a synchronizační mechanizmus začne od uvedení změn modelů data transformovat z relační databáze do grafové

databáze při jakékoliv změně záznamu. Aby se propsaly změny u všech záznamů, je potřeba spustit proces dávkové synchonizace, ten zajistí úplné propsání všech nových dat do Neo4J.

Poté je potřeba upravit doporučovací algoritmy, aby zohledňovaly volbu jazyka a byly schopny doporučit i zdarma položky. V případě doporučování položek jen se specifickými jazyky, se upraví množina procházených položek při doporučování, aby byla složena pouze z položek se zvolenými jazyky. V případě doporučování položek, kde alespoň jedna položka je zadarmo, se nabízejí dvě varianty, jak něco takového implementovat. První variantou je upravit skórovací funkce všech algoritmů, tak aby zdarma položky měly uměle vyšší skóre. Nebo druhou variantou, jak implementovat doporučení položek, které jsou zdarma a položek s nejvyšším skóre, je provést výpočet na dvakrát. Poprvé standardní doporučení, kde se prochází všechny položky a v případě, že se mezi nimi nenachází položka, která je zdarma, tak provést doporučení, kde se prochází pouze položky zadarmo.

Kapitola 5
Závěr

Práce se v souladu se zadáním věnuje navázáním mou bakalářskou práci a popisuje stávající datovou strukturu ve firmě LearnerOn, která je pro doporučování relevantní, čímž opět ukazuje užitečnost ontologií, zejména konceptuálního modelování.

Dalším dílčím cílem práce je nastudování problematiky kolaborativního filtrování a zaměření se na metody doporučování, které jsou vhodně aplikovatelné na grafové databáze se zmíněním alternativ.

Na základě této studie a na základě specifičnosti domény LearnerOn se pak diskutuje nad zvolením vhodné metody.

Při plnění tohoto cíle byl navržen postup více úrovňového doporučování při využití grafové databáze. Díky faktu, že se jedná o frontu doporučovacích mechanizmů, je pak možné budoucí rozšíření ve formě zařazení dalších doporučovacích algoritmů do fronty.

Dále se práce zabývá návrhem implementace a to nejen formou funkčního prototypu, ale produkčním řešením, které je možné již při psaní této práce vyzkoušet na webu LearnerOnu. Výsledkem této části práce jsou pak části reálného produkčního řešení, které se týká doporučování.

Poslední část práce přináší ověření funkčnosti na vhodných datech a také polemiku nad výsledky a budoucím rozvojem.

Rád bych zmínil skutečnost, že výsledky popsané v této práci se prakticky využívají a navíc se bude na nich dále stavět v budoucnu. Je také důležité podtrhnout, že předložená práce rovněž splňuje všechny dílčí cíle, vytyčené v jejím oficiálním zadání.

Bibliografie

1. SIKORA, Filip. *Modelování a řízení dat v doméně online vzdělávání*. České vysoké učení technické v Praze. Vypočetní a informační centrum., 2021. Dostupné také z: https://dspace.cvut.cz/handle/10467/95442. České vysoké učení technické v Praze.
2. SEN, Sudipta; MEHTA, Akash; GANGULI, Runa; SEN, Soumya. Recommendation of Influenced Products Using Association Rule Mining: Neo4j as a Case Study. *SN Computer Science*. 2021, roč. 2, č. 2, s. 74.
3. NEGRO, A. *Graph-Powered Machine Learning*. Manning, 2021. ISBN 9781638353935. Dostupné také z: https://books.google.cz/books?id=e1E%5C_EAAAQBAJ.
4. AKBAR, Ali; AGARWAL, Parul; OBAID, Ahmed. Recommendation engines-neural embedding to graph-based: Techniques and evaluations. *International Journal of Nonlinear Analysis and Applications*. 2022, roč. 13, č. 1, s. 2411–2423. ISSN 2008-6822. Dostupné z DOI: 10.22075/ijnaa.2022.5941.
5. TURING. *How Does Collaborative Filtering Work in Recommender Systems?* [online]. 2023-04. [cit. 2023-04-14]. Dostupné z: https://www.turing.com/kb/collaborative-filtering-in-recommender-system.
6. YANG, Xiao; ZHANG, Zhaoxin; WANG, Ke. Scalable Collaborative Filtering Using Incremental Update and Local Link Prediction. In: *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*. Maui, Hawaii, USA: Association for Computing Machinery, 2012, s. 2371–2374. CIKM '12. ISBN 9781450311564. Dostupné z DOI: 10.1145/2396761.2398643.
7. ROCCA, Baptiste. *Introduction to recommender systems* [online]. 2023-04. [cit. 2023-04-14]. Dostupné z: https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada.
8. PUTRA, Aghny Arisya; MAHENDRA, Rahmad; BUDI, Indra; MUNAJAT, Qorib. Two-steps graph-based collaborative filtering using user and item similarities: Case study of E-commerce recommender systems. In: *2017 International Conference on Data and Software Engineering (ICoDSE)*. 2017, s. 1–6. Dostupné z DOI: 10.1109/ICODSE.2017.8285891.

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `src`
  - `data` ..... testovací data
    - `courses.json` ..... datový soubor s testovacími kurzy
    - `users.json` ..... datový soubor s testovacími uživateli
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF