FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Serverová aplikace pro plánovač týdenního jídelníčku
Student:	Bc. Pavel Baroš
Vedoucí:	Ing. Jaroslav Kuchař, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Webové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Cílem je navrhnout, vyvinout a otestovat serverovou část aplikace, která bude z poskytnuté databáze doporučovat týdenní plán receptů, personalizovaný podle preferencí reálných uživatelů. Na doporučovací algoritmus budou kladeny nároky na doménově specifické vlastnosti, jako např. pestrost jídel či sezónnost receptů.

- Seznamte se s problematikou a sestavte stručný popis byznys požadavků a jejich analýzu.
- Seznamte se s daty poskytnutými zadavatelem a proveďte jejich základní analýzu.
- Proveďte rešerši současných doporučovacích systémů a přístupů vhodných pro danou oblast.
- Navrhněte a implementujte serverovou stranu aplikace pro generováni a úpravy týdenního plánu.
- Navrhněte a implementujte vhodný algoritmus pro doporučení receptů respektujících požadované vlastnosti.
- Proveďte testování pro ověření funkčnosti aplikace a zvolené architektury.
- Proveďte experimenty pro ověření vlastností doporučených plánů a jejich vyhodnocení.

Elektronicky schválil/a Ing. Jaroslav Kuchař, Ph.D. dne 18. ledna 2022 v Praze.

Diplomová práce

SERVEROVÁ APLIKACE PRO PLÁNOVAČ TÝDENNÍHO JÍDELNÍČKU

Bc. Pavel Baroš

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Jaroslav Kuchař, Ph.D.
5. května 2022

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Chtěl bych poděkovat všem, kteří se na této práci podíleli přímo i nepřímo, především ale vedoucímu za účelné poznámky a směrování mé práce. Velké díky za podporu a motivaci také patří mé snoubence, rodině a kolegům, bez nichž by tato práce s velkou pravděpodobností vůbec nevznikla.

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. Dále prohlašuji, že jsem s Českým vysokým učením technickým v Praze uzavřel dohodu, na jejímž základě se ČVUT vzdalo práva na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona. Tato skutečnost nemá vliv na ust. § 47b zákona č. 111/1998 Sb., o vysokých školách, ve znění pozdějších předpisů.

V Praze dne 5. května 2022 .....

Abstrakt

Práce popisuje návrh a implementaci serverové služby pro plánování a doporučování jídelníčku z dostupné databáze receptů. V práci jsou popsány všechny fáze vývoje takového systému - od vymezení požadavků, seznámení se s teorií doporučovacích systémů, až po evaluaci vybraných algoritmů. Webová služba je implementována jako RESTful API. Doporučovací systém je hybridní, kombinuje doporučení populárních položek a content-based filtrování.

Klíčová slova doporučení, jídelní plán, recepty, content-based filtrování, RESTful, webová služba

Abstract

The thesis describes the design and implementation of a server service for planning and recommending a meal menu from an available recipe database. The thesis describes all phases of the development of such a system - from the definition of business requirements, acquaintance with the theory of recommendation systems, to the evaluation of selected algorithms. The web service is implemented as a RESTful API. The system is hybrid, combining popular item recommendation and content-based filtering.

Keywords recommendations, meal plan, recipes, content-based filtering, RESTful, web service

Seznam zkratek

ALS	alternating least square
ASGI	Asynchronous Server Gateway interface
CSV	comma separated values
ORM	object relation mapping
RS	recommender system
DF	Django framework
DRF	Django rest framework

Kapitola 1
Úvod

Aplikaci, která umí sestavit jídelní plán, jsem hledal dlouho ale neúspěšně. Žádná neodpovídala mým představám. Využíval jsem ovšem jiné aplikace, jako například mobilní aplikaci plnou skvělých receptů, říkejme ji např. *MojeRecepty*. Čím déle jsem ji používal a víc a víc jídel vařil pravidelně, tím častěji se mi vkrádala myšlenka na jídelní plán právě z těchto receptů.

Idea této práce vznikla nezávisle v mé hlavě, ale také v hlavách vedení stejnojmenné společnosti *MojeRecepty*, kterou jsem s mou myšlenkou oslovil v ten pravý čas. Rád jím zdravě a chutně a proto, jsem se pokusil kontaktovat právě společnost, která staví své podnikání na těchto základech. Disponují databází vlastních receptů, kterou jsem využíval, dlouho před tím, než jsem vůbec věděl, že dojde i na bližší spolupráci.

V aplikaci *MojeRecepty*, dostupné i v základní, neplacené verzi, může zákazník prohlížet a vybírat z řady receptů, hodnotit recepty, ukládat si je do oblíbených nebo sdílet komentáře s ostatními uživateli. Recepty jsou rozděleny do různých kategorií např. snídaně, obědy a večeře, sladká jídla nebo zákusky, těstoviny a pod. Co se tedy nabízelo a také vykrystalizovalo až do této práce, bylo z těchto receptů nabídnout uživateli rovnou celotýdenní plán jídel. Vymyslet ovšem jídelníček není jednoduchá záležitost a nakonec poskytnout skutečný jídelní plán ani nebylo prvotním cílem.

Naše spolupráce se rodila standardně, po seznamovací emailové komunikaci, dospěla k osobní schůzce, kde byly sepsány základní požadavky a definováno jaká bude má úloha v celém projektu plánovače jídel.

Nejprve bylo třeba se v kap 2 teoreticky seznámit se základními typy doporučovacích systémů, jejich výhodami nebo naopak s jejich případnými nedostatky. V kap. 3 popisuji výsledek konzultací, které proběhly, tedy specifikace, jak má budoucí aplikace fungovat a co vše má splňovat. Nutné seznámení se s daty popisuje kap. 4. V Dalších kapitolách 7 a 6 pak seznamuji čtenáře s procesem vývoje jak doporučovacího systému tak vlastní webové služby.

Cílem této práce je vyvinout základní verzi plánovače jídel, schopného doporučit recepty podle chutí uživatele. Cílem je také ověřit funkčnost, i vizuálně, a užitečnost doporučovacího systému, což se snažím popsat v kapitole 8.

Kapitola 2
Doporučovací systémy

Dnešní doba je charakteristická záplavou dat a vyznat se v nich, nebo najít potřebnou informaci může být pro člověka velký problém. Často se jedná o rozsáhlé kolekce entit, pár příkladů za všechny: knihy na Amazon.com, hudba na Spotify, videa na Youtube, nebo téměř všechny větší online obchody.[12]

Naštěstí s tím přišla potřeba tento problém řešit a jednou z možností jak toho dosáhnout jsou doporučovací algoritmy. Běžně se můžeme setkat se systémem pro doporučování v internetových obchodech, které pracují s historií našich nákupů. Doporučovací systémy lze rozdělit podle více kritérií, asi nejběžnějším známým je rozdělení jako na obrázku 2.1[16].

Oba základní typy, kolaborativní i content-based filtrování se opírají o matematické vyjádření podobnosti.

2.1 Podobnosti

V doporučovacích systémech se často pracuje s podobnostmi vektorů. Zde uvádím ty základní, často zmiňované nebo demonstrované:

2.1.1 Kosínova podobnost

která je definována jako kosínus úhlu mezi dvěma vektory. Používá se v mnohodimenzionálních prostorech např. v IR[1] sytémech pro vyhledávání v dokumentech.[11]

$$cos(X, Y) = \frac{X * Y}{||X||||Y||} \tag{2.1}$$

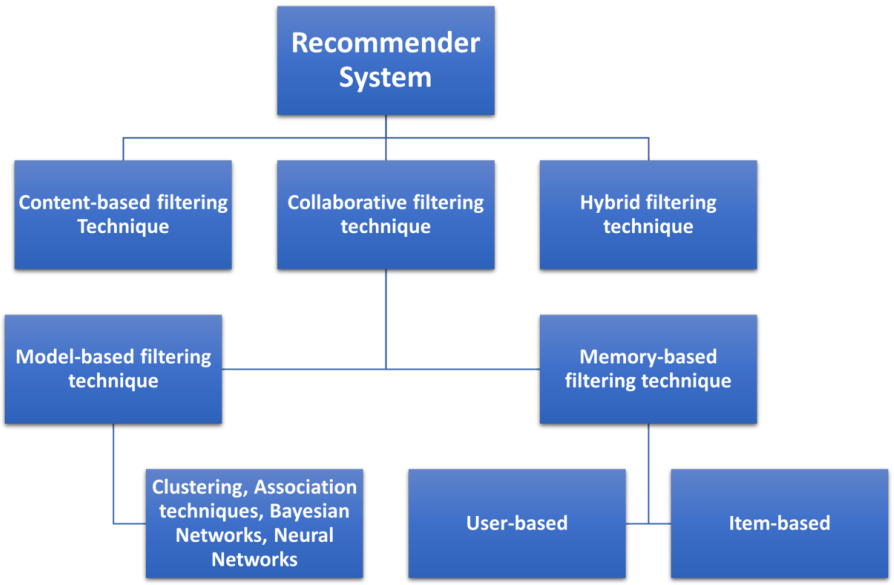
2.1.2 Pearsonova podobnost

která říká jak moc dva vektory mezi sebou korelují:

$$\text{pearson}(i, j) = \frac{\sum_{i=u}^{U} (R_{ui} - R_i)(R_{uj} - R_j)}{\sqrt{\sum_{i=u}^{U} (R_{ui} - R_i)^2} \sqrt{\sum_{i=u}^{U} (R_{uj} - R_j)^2}} \tag{2.2}$$

Nabývá hodnot $[-1, 1]$ kde hodnota blíže 1 znamená, že vektory jsou více korelované a naopak hodnota blíže -1 znamená, že vektory jsou korelované negativně.

[1] Information retrieval



Obrázek 2.1 Rozdělení doporučovacích sytémů[16]

2.2 Kolaborativní filtrování

Kolaborativní filtrování staví na předpokladu, že lidé mající stejný vkus mohou zajímat i stejné entity. Narozdíl od Content-based přístupu nevyžaduje informace o doporučované entitě.

Rozlišujeme dva přístupy:

- User-based
- Item-based

User-based hledá podobné uživatele, např. podle toho, jak hodnotí. Výstupem jsou entity, které vybraný uživatel nezná, ale jsou kladně hodnoceny jemu podobnými uživateli. Item-based CF hledá podobné entity např. podle toho, jak jsou hodnoceny.

Příklad kolaborativního filtrování by v našem případě mohl vypadat následovně:

Tabulka 2.1 Příklad User-based varianty kolaborativního filtrování

uživatel	guláš	carbonara	ceasar	hrachovka	pudink
A	4	-	3	5	3
B	-	4	1	5	-
C	-	4	-	5	3
D	5	5	3	2	2
E	4	5	1	1	-

Předpokládejme, že máme informace o hodnocení receptů od uživatelů jak ukazuje tabulka 2.1 s hodnotami 1 až 5 hvězd, kde jako nejlepší se označuje recept s 5 hvězdami. Pomlčka pak značí absenci hodnocení.

Řekněme, že bychom chtěli doporučit uživateli C nějaký z receptů. Nejprve potřebujeme najít uživatele, kteří hodnotili stejné recepty, a podívat se na jejich vzájemná hodnocení. Při použití kosínovy podobnosti bychom získali následující:

$$cos(C, A) = \frac{5 * 5 + 3 * 3}{\sqrt{5^2 + 3^2} * \sqrt{5^2 + 3^2}} = 1 \tag{2.3}$$

$$cos(C, B) = \frac{4 * 4 + 3 * 3}{\sqrt{4^2 + 3^2} * \sqrt{4^2 + 3^2}} = 1 \tag{2.4}$$

$$cos(C, D) = \frac{4 * 5 + 5 * 2 + 3 * 2}{\sqrt{4^2 + 5^2 + 3^2} * \sqrt{5^2 + 2^2 + 2^2}} = 0.886 \tag{2.5}$$

$$cos(C, E) = \frac{4 * 5 + 5 * 1}{\sqrt{4^2 + 5^2} * \sqrt{5^2 + 1^2}} = 0.768 \tag{2.6}$$

Nejpodobnější jsou tedy uživatelé A a B. Nyní nám zbývá se jen podívat na recepty, které tito uživatelé hodnotili kladně a námi sledovaný uživatel C nehodnotil vůbec. Lze tvrdit, že podle uživatele A, který hodnotil recept guláše 4 hvězdami můžeme zařadit do doporučení pro uživatele C, ovšem recept Ceasar salátu ani jednoho z uživatelů výrazně neohromil, tedy pokud zprůměrujeme jejich hodnocení, vychází 2 hvězdy a tento recept by nebyl vhodný kandidát na doporučení.

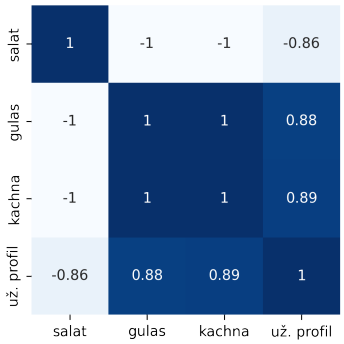
2.3 Content-based

Jak bylo zmíněno výše, content-based nevyžaduje informace o doporučované entitě, naopak zkoumá obsah doporučovaných entit. U receptů by se např. mohly porovnávat[2] ingredience, postup, časová náročnost nebo cokoli, co popisuje obsah a strukturu receptu. Výhodou může být transparentnost. Lze tak snadno zjistit, proč je daná entita vybrána.

Postup doporučení pak naznačuje obrázek 2.2a. Hledáme podobné položky těm, které má vybraný uživatel v oblibě a ty mu doporučíme. Často se snažíme jak položky, tak i uživatele, resp. profil, který představuje jeho zájmy, reprezentovat vektorem.

Obrázek 2.2 Content-based metoda

(a) princip doporučení nových položek *content-based* metodou[15]



(b) Výpočt Pearsonovy podobnosti pro data z tabulky 2.2

Takovým příkladem může být tabulka 2.2, kde máme recepty reprezentovány různými atributy. Zda je recept tradiční, jak je složitý na přípravu nebo prospěšný zdraví. V posledním řádku

[2] zde se uplatní různé metriky pro podobnost2.1

Tabulka 2.2 Příklad obsahových dat vhodných pro content-based algoritmus. Poslední řádek reprezentuje uživatelský profil.

recept	tradiční	složitost	masový	zdravý
šopský salát	0	0.1	0	1
guláš	1	0.9	1	0.3
kachna se zelím	1	0.9	1	0.4
profil	1	0.5	1	0.2

tabulky je profil uživatele, který mohl vzniknou např. průměrem atributů jeho oblíbených receptů. V dalším kroku využijeme jedné z metrik a vypočteme vzájemnou blízkost vektorů. Zde jsem pro ilustraci použil Pearsonovu podobnost a výsledek ukazuje obrázek 2.2b. Podle této metody, bychom uživateli s takovýmto profile doporučili guláš i kachnu se zelím.

2.4 Hybridní řešení

Ukazuje se, že správnou kombinací různých řešení můžeme eliminovat nevýhody jedné či druhé metody a dosáhnout tak lepších výsledků. Nejčastější používaným hybridním řešením je kolaborativní filtrování s další metodou, často kombinováno vážením obou řešení.[6] Konec konců jedno z možných hybridních řešení jsem vyzkoušel implementovat právě v této práci.

2.5 Zpětná vazba

Klíčové pro fungování doporučovacích systémů jsou informace o vztahu uživatel - entita, které máme k dispozici. Tento vztah nazýváme zpětnou vazbou (angl. feedback) a dělíme podle způsobu, jak ji získáváme na explicitní a implicitní.

Explicitní zpětná vazba vzniká explicitní akcí uživatele, významem nám říká, jestli uživatel hodnotí kladně, či záporně, příkladem je často používané hodnocení počtem hvězdiček. Implicitní zpětná vazba vzniká sběrem interakcí uživatele s entitou v naší aplikaci. Příkladem mohou být počty návštěv stránky, zobrazení produktu, nebo i přidání do košíku, bez následného dokončení nákupu.

2.6 Evaluace

Základnímy typy evaluace jsou:

- offline experimenty
- uživatelské studie (angl. user studies)
- online experimenty

Zatímco u offline experimentů není třeba interakce skutečných uživatelů a tudíž jsou logicky první volbou, u studií, kde menší skupina vybraných osob hodnotí daný systém, už roste cena i časová náročnost. V tomto ohledu jsou online experimenty nejnáročnější a také cílí na větší množství osob, cílových uživatelů systému. Příkladem online experimentu je tzv. A/B testování. Principem je rozdělit uživatele na dvě skupiny a každé skupině selektivně vnutit původní a ověřovaný systém. Podle vybraných kritérií pak porovnat úspěšnost nového algoritmu. Typicky lze porovnávat množství uživatelů, kteří nakoupí nebo si vyberou doporučenou položku v internetovém obchodě.

Kapitola 3

Popis byznys požadavků

Jak bylo zmíněno v úvodní kapitole, aplikace *MojeRecepty* nabízí vlastní zdravé a chutné recepty, často jakoby na míru specifickým potřebám. Jsou dietní, lehké na trávení, lehké nebo rychlé na přípravu. S rostoucím počtem receptů a uživatelů, ale rostla snaha nabídnout zákazníkovi něco navíc a jeden z nápadů bylo realizovat plánovač jídelníčku.

Týdenní plán se bude skládat z receptů, rozdělených do dní v týdnu - pondělí až neděle - a do jednotlivých chodů. Chodů bude fixně pět a to: snídaně, dopolední svačina, oběd, odpolední svačina a večeře. Naznačení toho, jak bude výstup vypadat naznačuje obr 3.1. Uživatel získá plán dvěma možnými způsoby: manuálně nebo doporučeně (bude vygenerován aplikační logikou).

- Manuální

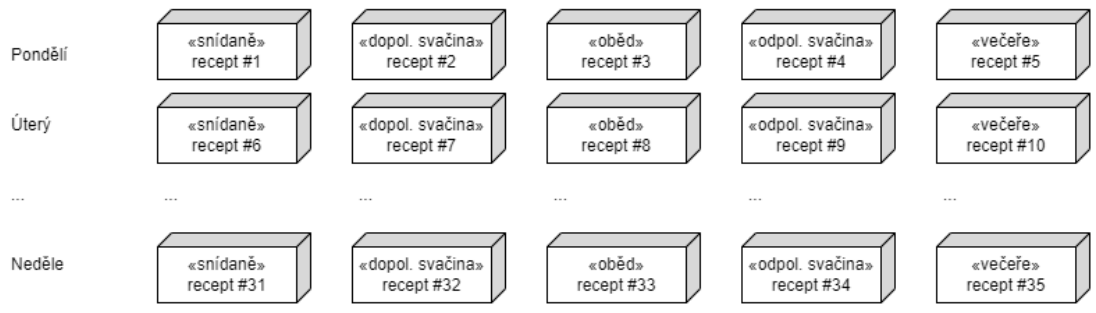
  Jeden z důležitých požadavků bylo, dát možnost uživateli si manuálně sestavit plán podle vlastního výběru. Rozhraní umožní uživateli nastavit vybraný recept na konkrétní den a chod.

- Doporučená varianta

  Vycházet bude z dostupných dat o uživateli a volitelných parametrů. Doporučovací algoritmus sestaví kompletní týdenní plán. Fungování algoritmu od výběru po sestavení plánu popisuji v kapitole 7

Přestože si uživatel vybere možnost automatického sestavení plánu, bude mít možnost libovolně upravovat jednotlivé chody. V aplikaci tak bude volba nahrazení konkrétního chodu za libovolný recept. Pro nahrazení doporučeného receptu tak primárně nabídneme alternativní kandidáty, které algoritmus našel. Pro výběr receptu v manuální variantě sestavování plánu všechny nalezené kandidáty využijeme.

Obrázek 3.1 Struktura výstupu



V dnešní době, kdy jsou data velmi ceněným artiklem společnosti, je logický požadavek na ukládání historie všech plánů. Jednak bylo požadováno, aby doporučený plán byl v námi definovaném časovém horizontu variabilní, ale také bude tato historie platná pro následnou analýzu dat. Zaznamenány budou jednak alternativy, které algoritmus vygeneruje pro daný chod. Zaznamenávat se bude také kolikrát byl daný recept změněn konkrétním uživatelem. Časté změny receptů budou analyzovány, jelikož se jedná o cennou informaci o konkrétním receptu, může to naznačovat, že je s ním něco špatně, nebo také to, že algoritmus doporučení je třeba poupravit.

Shoda panovala i na možnosti programově ovlivnit poměr receptů, které uživatel zná, ku receptům, které algoritmus nabídne uživateli jako nové.

3.1 Nároky specifické doméně

Ze společné analýzy o tom, jakými kritérii bude uživatel hodnotit vygenerovaný plán, vyvstaly následující otázky:

- Je jídelníček dostatečně pestrý?

  je třeba pohlížet na plán i jako na celek, tzn. pokusit se o minimalizaci opakujících se receptů

- Jsou ingredience, ze kterých se recept skládá, závislý na ročním období?

  kritérium (dále pod pojmem ”sezónnost receptu”) lze objektivně snadno implementovat v případě, že jsou k dispozici důvěryhodná data. Tento požadavek vyplynul z filozofie společnosti a čím dál větší snahy o udržitelné stravování.

3.2 Filtrováni podle preferenci

Každý uživatel bude mít možnost si explicitně nastavit předem definované preference jídel. U receptu bude specifikováno jestli spadá do kategorie:

- veganské
- vegetariánské
- bezlepkové
- bez laktózy

Pro tento požadavek bude klíčová analýza dat a to proto, aby měl systém možnost výběru z dostatečného množství receptů pro doporučení plnohodnotného plánu.

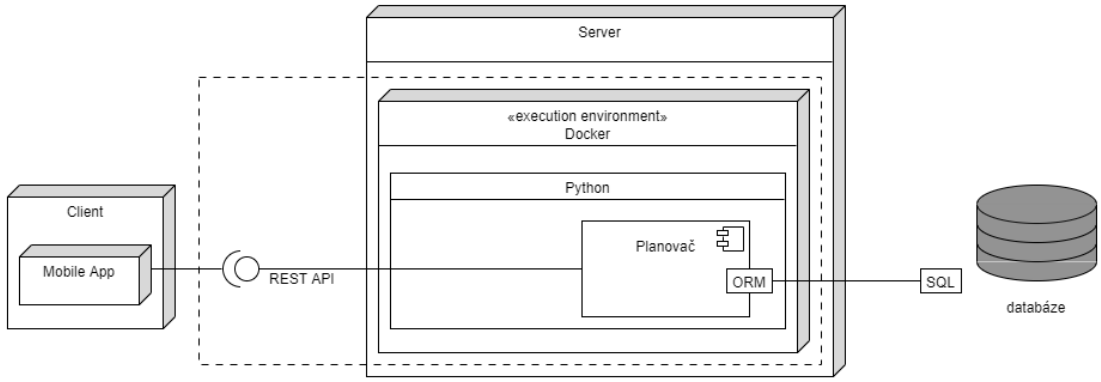
3.3 Architektura

Aplikace se skládá ze tří high-level komponent. Jak naznačuje obr. 3.2 jedná se o primární uložiště dat, konkrétně Postgres databázi, REST API[27] služby běžící ve virtuálním prostředí založené na kontejnerech Docker a uživatelské rozhraní, přesněji mobilní aplikaci, která je plně v zodpovědnosti spolupracovníků.

Mým úkolem je návrh a implementace těchto součástí:

- datového modelu pro vytvoření plánu, uložení, změnu a jeho analýzu
- REST API služby běžící ve virtuálním prostředí

Obrázek 3.2 Architektura aplikace - čerchovaná část ohraničuje cíle této práce



3.4 Technické požadavky na serverovou část

Serverová část aplikace je a bude nezávislá na technologiích použitých jinde v aplikaci. Přesto bylo vhodné zohlednit kompatibilitu se současným řešením mobilní aplikace. Ve výběru konkrétních technologií jsem na přání kolegů zohlednil i to, aby se jednalo o jimi známé produkty v případě, že bych ve spolupráci nepokračoval.

Zdrojové kódy serverové části jsou psány v jazyce Python. Hlavními a logickými důvody byly v první řadě to, že se jedná o kolegům známý programovací jazyk, ale také fakt, že je na něm postaveny stávající serverové služby.

Čistě technický požadavek na formát výstupu padl už při první osobní schůzce a tím byl: výstup má obsahovat identifikátory receptu, rozdělené do dní a chodů.

Posledním přáním technického zástupce společnosti bylo zjednodušit proces testování a následného nasazení komponenty. Toho dosáhneme využitím kontejnerizace technologií Docker.

Kapitola 4
Analýza dat

Žádná práce na doporučovacím systému se neobejde bez řádného seznámení s dostupnými daty. Přestože máme k dispozici data o uživatelích a receptech za několik let činnosti, je nutné provést alespoň základní analýzu. Dostatek dat ne vždy znamená kvalitní data.

4.1 Datový model

Pro účely práce jsem dostal přístup ke kopii skutečné databáze a s pomocí kolegy jsme vybrali tabulky a data, která budou nutné a užitečné pro splnění zadání. Jejich seznam a stručný popis uvádím níže.

Data z databáze:

- RECIPES - tabulka receptů
- RECIPECATEGORY - kategorie receptů, každý recept patří alespoň do jedné z typových kategorií
- SAVEDRECIPE - uchovává uživatelem uložené recepty, v aplikaci akce ”přidat do oblíbených”
- INGREDIENT - každý recept se skládá z několika ingrediencí, které mají jednotku a množství
- EDIBLE a EDIBLE_ALIAS - popis potraviny, ze které se daná ingredience skládá
- RATING - hodnocení receptů uživateli, hodnotí se hvězdičkami v rozmezí žádná až pět hvězdiček

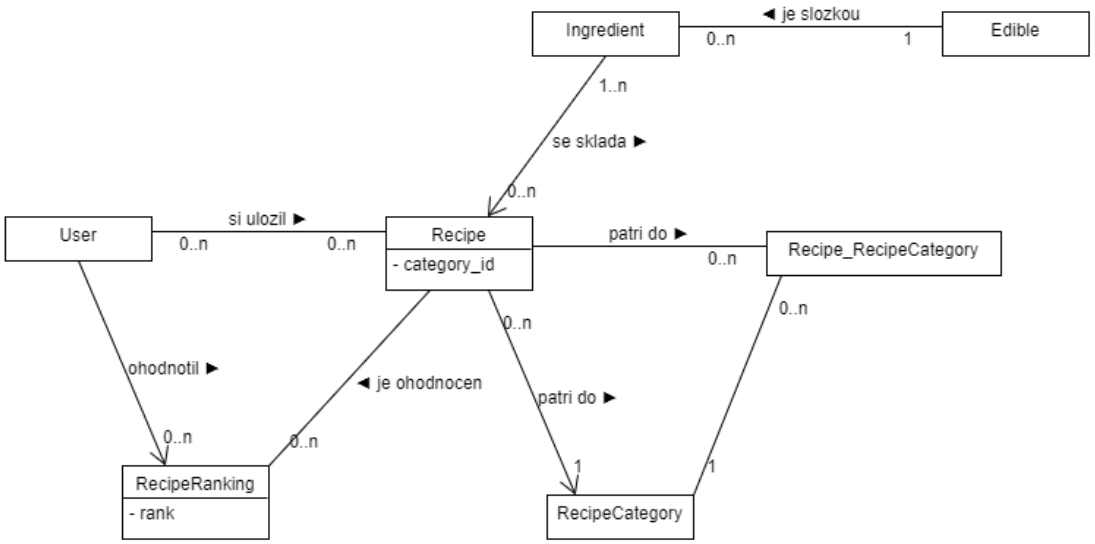
Data exportovaná z databáze nebo jiných zdrojů:

- VIEWS - ve formátu CSV[1] se sloupci o poslední návštěvě daného receptu, identifikátor receptu, identifikátor uživatele a suma návštěv

V celém dokumentu používám slova *rating*, *saved* a *views* pro pojmenování dat, která popořadě odkazují na: hodnocení receptu uživatelem, uložení receptu do oblíbených a návštěvu neboli zhlédnutí receptu uživatelem.

[1] comma-separated values

Obrázek 4.1 Datový model receptu a vybraných tabulek



4.1.1 Kategorie receptů

U receptů bych ocenil explicitní informaci, kterému dennímu chodu ho lze přiřadit. Tuto znalost nemáme, lze ji ale dovodit z následujících typových kategorií.

- Snídaně a svačiny
- Obědy a večeře
- Zdravé mlsání
- Polévky
- Saláty a přílohy
- Těstoviny a rizota

Každý recept patří alespoň do jedné typové kategorie. Kategorií je však více a ty, zajímavější jsou:

- Pomazánky
- Recepty na cesty
- Obědy a svačiny do práce i do školy

4.2 Statistiky

Důležitým sumárním údajem o datech je tzv. *data sparcity* (řídkost). Užívá se i recipročí hodnota, která vyjadřuje hustotu dat. Udává se v procentech a říká nám jak moc dat nám chybí.

Hodnota vznikla vždy z velikosti dimenze “počet uživatelů“ x “počet receptů“.

Tabulka 4.1 Řídkost dat - v tabulce jsou vyneseny hodnoty řídkosti jednotlivých datasetů interakcí popořadě pro nefiltrovaný dataset, pro dataset, v kterém jsou jen uživatele s minimálním počtem 5 interakcí ($> 5int.$), a v posledním sloupci je dataset, v kterém jsou jen uživatelé s minimálním počtem 10 interakcí ($> 10int.$)

	řídkost v %		
dataset	plný dataset	$> 5int.$	$> 10int.$
rating	99,71	98,24	96,36
saved	99.11	98,46	97,95
views	98.77	98,25	97,82

Tabulka 4.2 Počty interakcí na uživatele

dataset	průměr interakcí na uživatele
rating	2.2
saved	13.8
views	38.1

4.2.1 Recepty a uživatelé

Celkový počet receptů, 1986, není velké číslo, rozhodně ne pro současné doporučovací systémy, které dnes pracují s vyššími řády počtu položek a milióny uživatelů. Počty uživatelů, kterým nabídneme jídelníček, budou záviset na strategii, která se ve finální verzi produktu zvolí. Těch, kteří mají alespoň nějakou zpětnou vazbu jsou vyšší desítky tisíc.

Počty receptů náležících konkrétním chodům pak shrnuje tabulka 4.3.

Stejně jako množství kategorií mě zajímal celkový počet receptů v pro jednotlivé uživatelské preference shrnuté v tabulce 4.4.

4.2.2 Návštěvnost

Průměrný počet návštěv na uživatele je přibližně 38, medián je 12 a 90% uživatelů má maximálně 96 návštěv.

Graf 4.2 vykresluje počet návštěv na ose X a počet uživatelů, kteří tohoto počtu dosáhli na ose Y. Počty uživatelů jsou vyneseny v logaritmické stupnici. Vidíme tedy, že je jen málo uživatelů, kteří navštívili aplikaci více než 1000x oproti celkovému počtu cca 90.000 uživatelů.

4.2.3 Hodnocení

Tyto data jsou užitečná pro svou vypovídající hodnotu, kdy oproti oblíbenosti, známe i informaci o receptech, které uživateli nechutnají nebo neodpovídají jeho představám. Navíc procento receptů s aspoň jedním hodnocením je cca 40%.

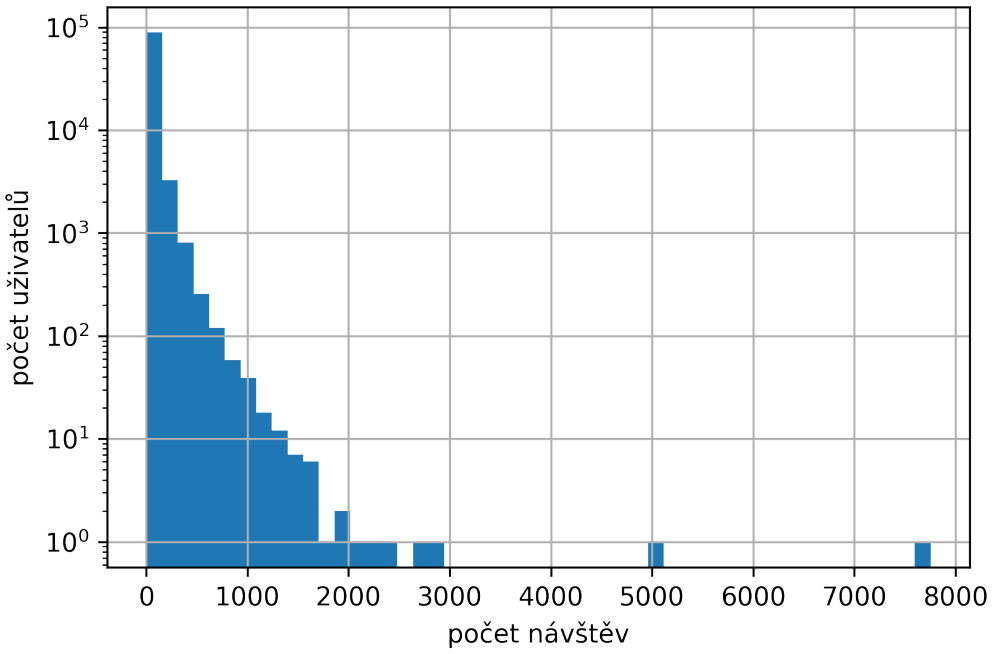
Tabulka 4.3 Kategorie - *pravý sloupec vyjadřuje poměr počtu receptů dané kategorie ku celkovému počtu receptů*

kategorie	počet	% z 1986
snídaně	395	19.99
svačiny dopolední	825	41.5
obědy	842	42.4
svačiny odpolední	825	41.5
večeře	1087	54.7

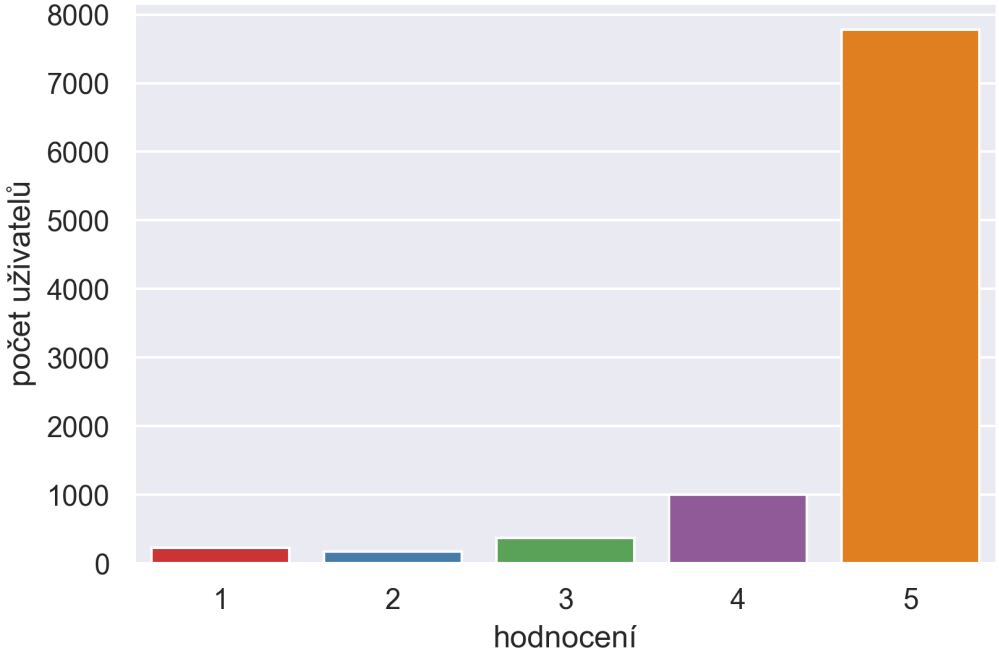
Tabulka 4.4 Počty receptů v kategoriích diet - z celkového počtu receptů 1986 nás zajímá, kolik v dané kategorii diety takovýchto receptů najdeme. V prvním sloupci je daný počet, v druhém pak procenta z celkového počtu. Kategorie nejsou disjunktní, proto součet nemůže dávat dohromady 100% ovšem poměr nám alespoň nastíní bohatost sady receptů.

dieta	počet	% z 1986
veganské	420	21.1
vegetariánské	710	35.8
bez laktózy	1570	79.1
bezlepkové	1182	59.5

Obrázek 4.2 Graf návštěvnosti *histogram ukazuje počet uživatelů rozdělených podle sumy počtu návštěv všech receptů. Ypsilonová souřadnice je škálována logaritmicky. Reálně tak počet uživatelů, kteří mají mají více návštěv prudce klesá.*



Obrázek 4.3 Graf hodnocení - *uživatelé výrazně více hodnotí recepty, které jim chutnají a velmi málo ty, které naopak nejsou dobré.*



Z grafu 4.4 lze vyčíst vztah mezí tím, jak uživatelé hodnotí a počtem těchto hodnocení. Je zřejmé, že častěji jsou hodnoceny recepty, které mají zároveň vysoké hodnocení.

4.2.4 Oblíbené

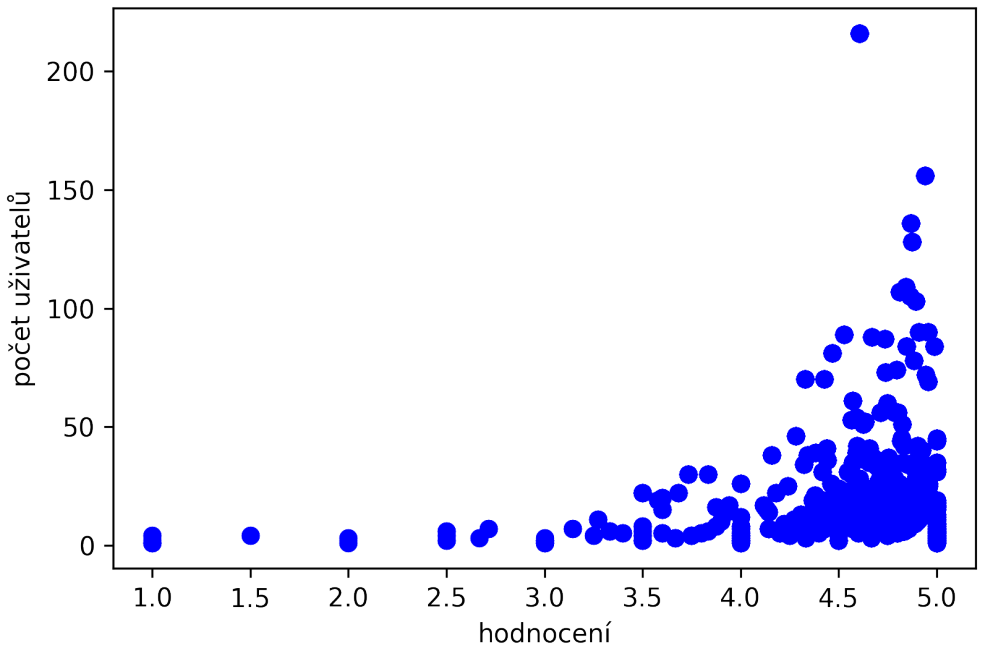
Už z tabulky 4.1 vidíme, že disponujeme výrazně více hodnotami o oblíbenosti receptu uživatelem oproti explicitnímu hodnocení. Tento fakt může hrát roli v užitečnosti jedné nebo druhé skupiny dat.

4.2.5 Závěr

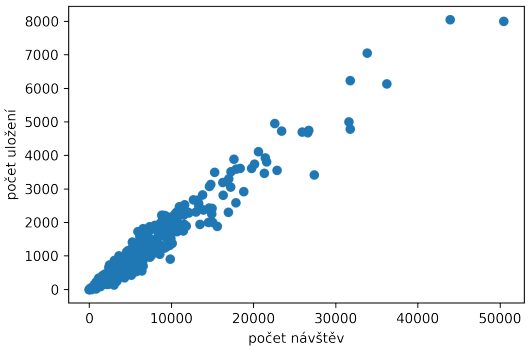
V této části práce jsem se pokusil nahlédnout do dat, která jsou k dispozici. Dá se říci, že žádný chod neobsahuje výrazně menší podmnožinu receptů, než by byl průměr na jeden chod. Průměrem na jeden chod je myšlena jedna pětina (počet chodů) z celkového počtu receptů.

Až na data o hodnocení, která jsou poměrně řídká a neobsáhlá máme možnost využít především data o oblíbených receptech. nenesou sice informaci o míře obliby, ovšem pořád se jedná o explicitní kladnou zpětnou vazbu. Jsou také výrazně obsáhlejší nežli data o hodnocení.

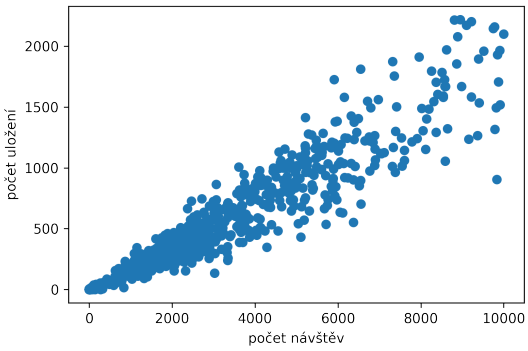
Obrázek 4.4 Graf hodnocení *ukazuje vztah hodnocení a počet uživatelů, kteří takto hodnotili*



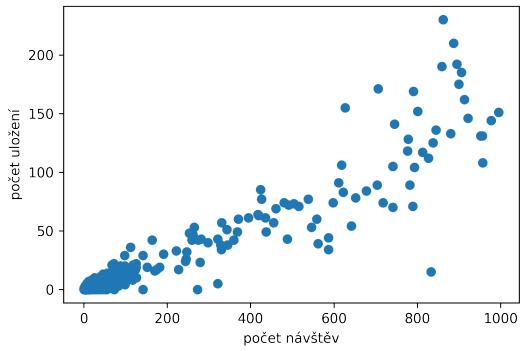
Obrázek 4.5 Grafy závislostí *views* a *saved* - body grafu reprezentují jednotlivé recepty. Na vodorovné ose grafu lze najít sumu shlédnutí všemi uživateli dohromady a na svislé ose pak kolikrát byl recept uložen do oblíbených. První graf 4.5a zahrnuje celý dataset. Ostatní datasety limitované popořadě počtem 10000, 1000 a 100 návštěv.



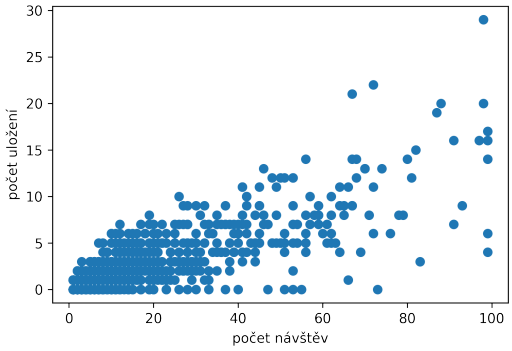
(a) celý dataset



(b) $views \leq 10000$

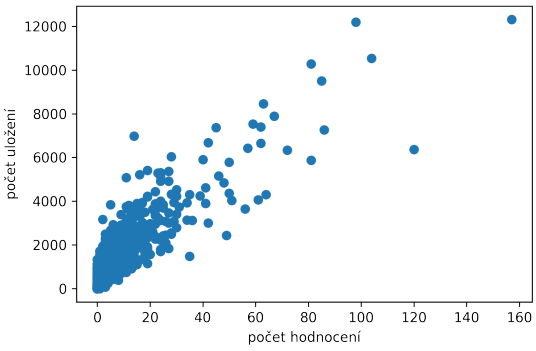


(c) $views \leq 1000$

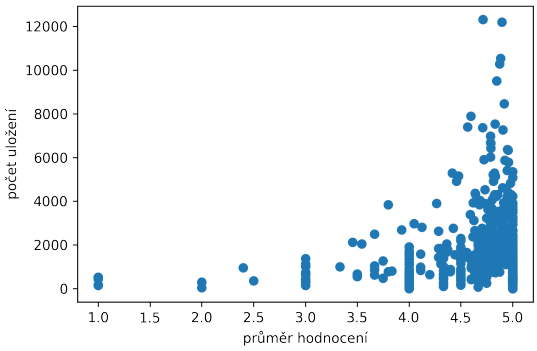


(d) $views \leq 100$

Obrázek 4.6 Grafy vztahu *saved* a *rating* - body jsou opět jednotlivé recepty. Na ose Y jsou vyneseny počty uložení receptu a na osách popořadě: 4.6a počet hodnocení a 4.6b průměr hodnocení daného receptu



(a)



(b)

Kapitola 5
Použité technologie

To této doby jsem pracoval s jazykem Python pouze na školních projektech. Proto bylo pro mne důležitým prvním krokem, abych mohl začít psát kód serveru, oživit si a prohloubit znalosti v tomto programovacím jazyce [13]. V následujících řádcích se snažím popsat další technologie, které jsou součástí projektu.

5.1 Server a REST service frameworky

RESTful API je standard využívající technologii HTTP [1]. Popisuje to, jak má vypadat rozšiřitelné webové rozhraní a nabízí pohled na jednotnou architektúru služeb[27].

RESTful službu lze vytvořit různými způsoby, ale nejsnadnější je využít existujících frameworku, které pomáhají dodržet standard. Jaký konkrétní framework bude použit, jsem se rozhodl díky menší rešerši dostupných řešení a výsledná volba padla na Django REST framework.

- **Django** Django je populární webový framework a open-source projekt v jazyce Python. Rychlý a jednoduchý na použití, přesto komplexní, poskytující řadu užitečných nástrojů a modulů.

  Pro mé účely nejvýznamnějším je systém mapování relačních tabulek na objekty - ORM[2]. Pomáhá usnadňovat komunikaci s databází a také výrazně zpřehledňuje zdrojový kód. Jelikož tabulky zdrojové databáze jsou spravovány právě systémem Django ORM, bylo nasnadě využít tento systém i pro mou práci.

- **Django REST framework**

  Jedná se o vcelku robustní řešení[2], vycházející z webového frameworku Django. Přejímá jeho ORM, což je nesporná výhoda pro kompatibilitu se současnou databází.

- **psycopg2** PostrgeSQL databázový adapter pro jazyk Python. Výhodou je efektivita a zabezpečení ve vícevláknovém prostředí.

- **Flask Restful** Jde o jednoduchý a svižný nástroj na tvorbu microslužeb. Jedná se o rozšíření webového frameworku Flask.

- **Fast API** Jak jméno napovídá, jde o velmi rychlý, moderní webový framework. Staví na asynchroním rozhraní ASGI[3].

[1] Hyper Text Transfer Protocol
[2] objektově relační mapování
[3] Asynchronous Server Gateway interface

Poslední dvě jmenované knihovny byly hlavními konkurenty pro Django REST framework, který jsem vybral i kvůli své robustnosti a z části pro snažší integraci se současným systémem.

5.2 Manipulace s daty a algoritmy doporučování

- **Pandas** Hojně využívaná knihovna v Pythonu pro manupilaci s daty a statistickou analýzu. Je rychlá a efektivní při práci s daty i z důvodu optimalizace kritických částí kódu.[26]
- **scikit-learn** [1] Velmi populární open-source knihovna obsahující množství modulů pro datovou analýzu nebo algoritmů strojového učení. Staví na nástrojích *NumPy*, *SciPy*, and *matplotlib*.
- **Surprise** Knihovna nabízí mnoho doporučovacích algoritmů připravených k použití, ovšem varianty vhodné výhradně pro jejich explicitní data sety. [13]
- **Case Recommender** Open-source knihovna nabízí široký výběr doporučovacích algoritmů. Výhodou je rozšířitelnost a možnost vytvoření a evaluace vlastních implementací. Nabízí algoritmy pro predikci hodnocení a také systémy pro doporučení v dobře známém scénáři doporučování položek uživatelům.[9]
- **Implicit** Knihovna několika různých populárních algoritmů doporučování na práci s implicitními daty optimalizovaná pro rychlý výpočet tréninkové fáze modelování. Všechny algoritmy požívají paralelizaci na úrovni vláken a s pomocí knihoven Cython a OpenMP rozloží zátěž na všechna dostupná jádra CPU. [7]

5.3 UI a testování

- **Jupyter Notebook a Google Colab** Pro průběžné testování a základní zobrazení výstupu jsem využil software Jupyter. Tento open-source projekt je hojně využíván ve data-science sféře pro svou interaktivitu a podporu mnoha programovacích jazyků.
- **Jupyter Widgets** Protože jsem potřeboval také vizuálně ověřit výstupy doporučovacího algoritmu, hledal jsem způsob, jak rychle a efektivně výsledky zobrazit a dočasně tak nahradit budoucí uživatelské rozhraní. K tomu účelu mi posloužil subsystém Jupyter Widgets, který umožňuje použít interaktivní prvky v Jupyter Notebooku.

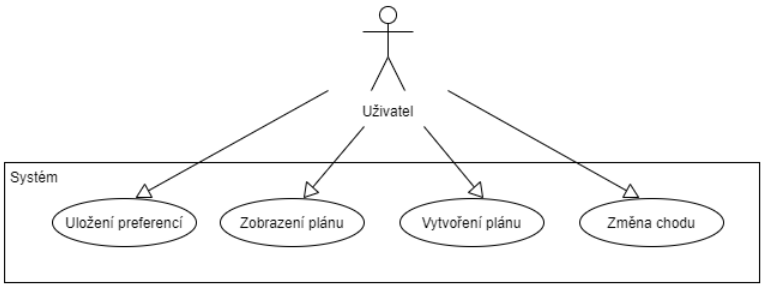
Kapitola 6

Datový model a REST API

Základními kameny, na kterých se dá stavět náš plánovač, jsou vlastní datový model a serverová služba, která obsluhuje požadavky aplikace. Nejdříve je třeba přesně specifikovat, jaké jsou případy užití naší služby.

6.1 Případy užití

Základní případy užití pro serverovou část aplikace jsou popsány níže. Nejvýznamnější akcí uživatele je vytvoření plánu, od kterého se odvíjí jeho zobrazení nebo změna chodu. Uložení preferencí je pak nezávislou a samostatnou akcí, která se může uplatnit i jiných aplikacích.



Obrázek 6.1 Případy užití

- Uložení preferencí
- Vytvoření plánu
- Zobrazení plánu
- Změna chodu

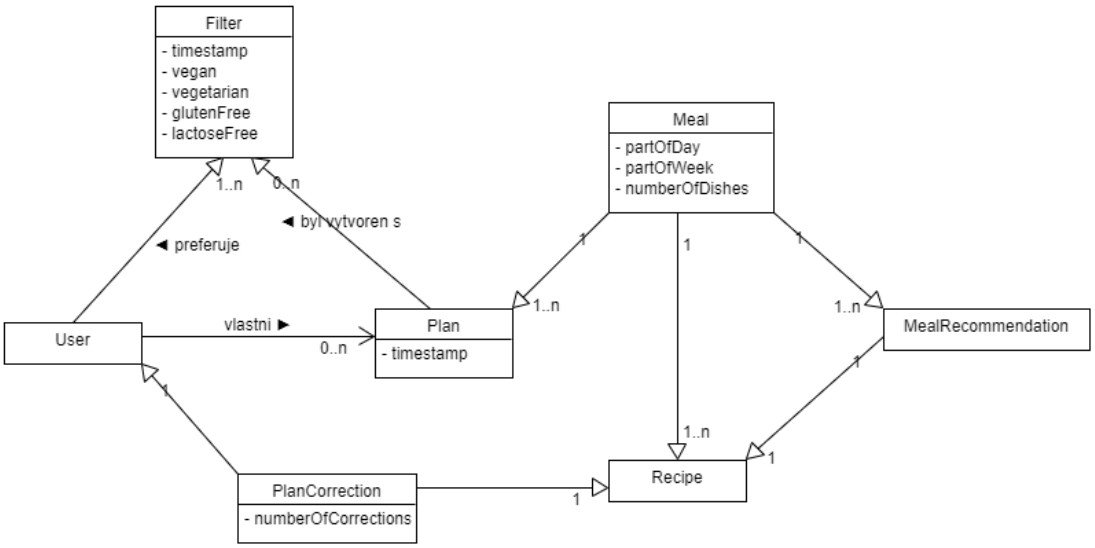
6.2 Datový model

Z požadavků jsem vyvodil několik základních objektů, které reprezentuji reálný svět. Mezi ně patří uživatel *USER* který si vybírá recepty *RECIPE* a buďto si nechává vytvořit plán *PLAN* nebo si jej sestavuje manuálně z receptů a uspořádává si je do chodů *MEAL* podle potřeby.

Uživatelské preference byly definovány už od okamžiku sepisování požadavků jako něco, co si uživatel zvolí vědomě v aplikaci. V datovém modelu jsou reprezentovány tabulkou *FILTER* a ve stejném významu jako filtr fungují.

- MEAL - chod, který má přesně daný den a čas, na který je naplánován.
- PLAN - plán seskupuje chody od pondělí do neděle od snídaně po večeři.
- USER - reprezentace uživatele, pro mé účely jsem dostal kvůli ochraně dat identifikátor uživatele pozměněný hash funkcí
- RECIPE - recept, s atributy o alergenech a dietách
- FILTER - reprezentuje preference uživatele pro daný plán
- MEALRECOMMENDATION - uchovává alternativní kandidáty v případě, že by nebyl uživatel spokojen s nabízeným receptem.
- PLANCORRECTION - uchovává informaci o změně receptu uživatelem

Obrázek 6.2 Datový model receptu a přidružených tabulek



6.2.1 Historie

Pro ukládání historie všech doporučených plánů není třeba dodatečných tabulek, jelikož je model koncipován tak, aby nedocházelo ke ztrátě těchto dat.

Kvůli dlouhodobé analýze doporučování a ukládání historie nastavení vznikly dvě tabulky v databázi. Současně slouží pro ukládání ladících zpráv a dalších výstupů.

- CONFIGLOG - pro ukládání nastavení a parametrů, které ovlivňují běh programu.
- OUTPUTLOG - pro záznam receptů doporučovaných v rámci algoritmu jako prostý seznam před sestrojením plánu.

6.3 Příprava dat

K tomu, aby bylo možné vyvíjet nezávisle s produkčním prostředím, jsem dostal přístup do kopie databáze, v které mi byla přiřazena práva k patřičným tabulkám. Přesto bylo třeba v průběhu vývoje některá data doplnit nebo transformovat.

Kvůli tomu, abych aspoň částečně zachoval vazby uživatele s ostatními relacemi, jsem se nakonec rozhodl vytvořit duplicitní tabulku uživatelů. Bylo to pro tuto fázi vývoje výhodné a rychlé řešení.

6.3.1 Import

Jak poznat zda daný recept odpovídá nějaké preferenci, resp. dietě, se řešilo nakonec manuálně i když jsem měl hrubou představu i o strojovém zpracování těchto atributů. Vzhledem k tomu, že všechny recepty jsou z dílny užšího okruhu lidí, a jejich celkový počet není manuálně nezpracovatelný, bylo rozhodnuto, že informace budou doplněny jednou osobou. Důvodem pro, byla i snaha zachovat vysokou kvalitu dat. Dostal se mi tedy k rukám soubor ve formátu Excel z kterého jsem si vytvořil CSV soubor *diet.csv* pro snadnější import dat do databáze.

6.3.2 Transformace

Některá data byl třeba upravit do podoby vhodné ke zpracování naším algoritmem. Například specifikovat, který recept je vhodný ke snídani a který k obědu, odvozuji z kategorií. Stačilo tedy vytvořit pro každý typ chodu SQL příkaz a specifikovat, které kategorie zahrnují recepty vhodné do daného chodu. V ukázce 6.1 jsou definovány snídaně podle těchto kategorií: *Snídaně a svačiny* (1), *Recepty na cesty* (110) a *Obědy a svačiny do práce i do školy* (111). Abych měl při sestavování plánu co nejbohatší výběr, zahrnul jsem do chodů recepty nejen z typových kategorií, ale i z těch sekundárních.

Výpis kódu 6.1 Transformace kategorií na chody

```
update  mealplanner_recipesrecipe
set     "partOfDay1" = true
from    recipes_recipe as r
join    recipes_recipe_list_in_categories as lcat
        on r.id = lcat.recipe_id
where   (lcat.recipecategory_id in (1, 110, 111)
or      r.category_id in (1, 110, 111))
and     mealplanner_recipesrecipe.id = r.id
```

6.4 Přejaté modely

V případě, že bych nebyl svázán omezením práv k tabulce uživatelů, bych vytvářel své modely s cizím klíčem do originální relace uživatelů. Jak jsem zmínil výše, bylo třeba vytvořit vlastní relaci reprezentující uživatele a na ni odkazovat cizím klíčem z mých modelů. U ostatních přejatých tabulek jsem využil možností Django ORM a toho, že v databázi jsou modely vytvořeny a spravovány právě tímto ORM systémem.

Pro práci s Django ORM slouží administrační příkazy a pro export modelu slouží příkaz *inspectdb*, který se pouští z příkazové řádky jako argument administračního nástroje *manage.py*:

Výpis kódu 6.2 Export definice objektového modelu z databáze

```
> python manage.py inspectdb
```

Výpis kódu 6.3 Příkaz pro nasimulování migrace modulu v Django ORM

```
> python manage.py migrate recipes --fake
```

Export mi posloužil pro vytvoření existujících modelů. Vybrané tabulky jsem vložil do projektu do modulu *recipes* a vytvořil migraci příkazem *makemigrations*.[23] Adaptace modelu na databázi jsem řešil přepínačem *fake* příkazu pro migraci modulu6.3.

Problém nastává u relací s cizím klíčem odkazujícím na tabulku, která není potřeba modelovat. Jsou dvě možnosti řešení. V případě, že využijeme hodnotu klíče lze nahradit vazbu definicí standardního typu jako má typ primárního klíče odkazované relace. Druhou možností je odstranit vazbu úplně.

Důležité je v definici tabulek označit atribut “managed = False“:

Výpis kódu 6.4 Atribut pro označení externě spravované tabulky

```
...
class Meta:
    managed = False
...
```

Django ORM takto explicitně říkáme, že model mu nenáleží a je spravován jiným systémem.

6.5 REST API

Stejně jako u modelování datových objektu postupujeme u modelování API. Omezil jsem je ale na objekty, které spravujeme a na operace, které jsou nutné k splnění požadavků.

6.5.1 Plan

Nejvýznamnější akcí, kterou lze provést je vygenerování týdenního plánu. Podobná akce se stejným dopadem na data má i započetí manuálního sestavování. Rozlišení těchto dvou akcí je provedeno dodatečným parametrem v query segmentu URL cesty.

Výpis kódu 6.5 Vytvoření plánu

```
POST /api/plans
```

Výpis kódu 6.6 Získání celého plánu se všemi chody

```
GET /api/plans/(id)
```

Aby API odpovídalo standardu REST API, metoda POST vrací identifikátor nově vzniklého objektu, v našem případě plánu. Navíc v hlavičce požadavku je uveden parametr Location s hodnotou cesty URL identifikující nově vzniklý plán.

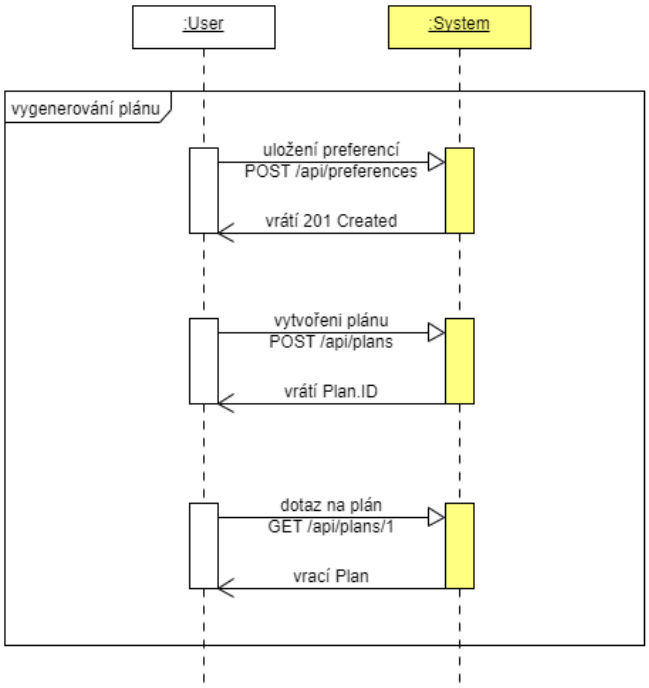
6.5.2 Meal

Nejmenší jednotka celého plánu, reprezentuje konkrétní jídlo pro daný den i chod, ukázka 6.7 nám říká, že recept 234 je čtvrteční oběd.

Výpis kódu 6.7 Ukázka objektu Meal

```
Meal.partOfWeek == 3       # 3 = ctvrtecni
Meal.partOfDay == 2        # 2 = obed
Meal.recipe_id == 234
```

Obrázek 6.3 Procesu vytvoření plánu může předcházet uložení uživatelských preferencí.



Meal entita je vytvářena spolu s plánem, ke kterému náleží, nemá proto vlastní akci pro vytvoření. Prerekvizitou úpravy chodu je logicky vytvoření plánu. Získání alternativ jako podle vzoru v 6.10 je pro konzumenta služby volitelné, ovšem tohoto volání využije aplikace, aby uživateli nabídla alternativní recepty pro daný chod.

Výpis kódu 6.8 Úprava chodu

```
PUT /api/meals/(id)
```

Výpis kódu 6.9 Úprava chodu

```
GET /api/meals/(id)
```

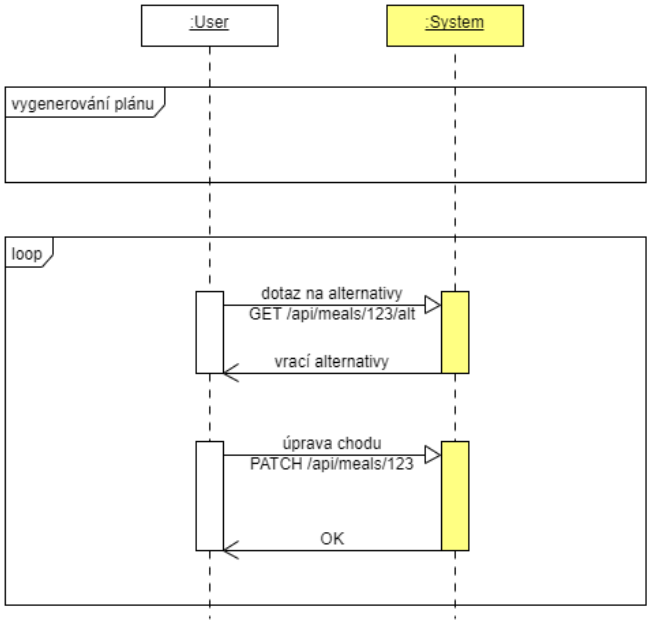
Výpis kódu 6.10 Získání alternativ k chodu

```
GET /api/meals/(id)/alt
```

6.5.3 Preferences

Uživatelské preference vznikají nezávisle na všem ostatním. Jsou ale předpokladem pro generování plánu a mohou se časem měnit, stejně jako lidské chutě. Hlavním důvodem nezávislosti na plánu je skutečnost, že se jedná o atributy uživatele, tedy nejedná se o vlastnosti jídelního plánu. Uživatelské preference tak lze uplatnit i v dalších aplikacích.

Obrázek 6.4 Procesu úpravy chodu předchází vytvoření plánu a může mu předcházet získání alternativních receptů.



Výpis kódu 6.11 Příklad volání změny uživatelských preferencí

```
POST /api/preferences
{
    'vegan': false,
    'vegetarian': true,
    'glutenFree': false,
    'lactoseFree': false
}
```

6.6 Testování

Pro otestování aplikace jsem použil kopii databáze. Obecně se sice předpokládá testování na datech, které si připravíme a testujeme očekávaný výstup, ovšem v případě tohoto projektu, by tento postup nedával smysl. Testované API neslouží pro vytváření ani uživatelů, ani receptů a sestavování plánu předpokládá vytváření plánu z receptů pro uživatele. obojí musí být v systému přítomno, aby měl systém co doporučit.

6.6.1 Unit testy

Pro otestování funkčnosti tříd pomocí unit testů používám knihovnu *pytest*.

Výpis kódu 6.12 Spuštění unit testů a integračních testů dohromady

```
pytest --ds=mojerecepty.test-settings
```

Výpis kódu 6.14 Výstup po spuštění příkazu pytest

```
========================= test session starts ==========================
platform win32 -- Python 3.9.10, pytest-7.1.2, pluggy-1.0.0
django: settings: mojerecepty.test-settings (from option)
rootdir: c:\Work\mojerecepty\development\src, configfile: pytest.ini
plugins: anyio-3.5.0, django-4.5.2
collected 11 items

mojerecepty\tests\test_filtering.py ....                [ 36%]
mojerecepty\tests\test_meal.py ..                       [ 54%]
mojerecepty\tests\test_preference.py ..                 [ 72%]
mojerecepty\tests\tests.py ...                          [100%]

========================== 11 passed in 3.45s ==========================
```

6.6.2 Testování API

Knihovna *pytest* posloužila jak pro unit testy tak pro spuštění integračních testů. 6.12.[19]

Pro testování poskytuje Django rest framework (DRF) několik tříd, které rozšiřují možnosti samotného Django frameworku (DF).[21] Jsou to především třídy *APITestCase*, která zapouzdřuje třídu *TestCase* z DF, čímž získáme přístup k *APIClient* namísto výchozí *Client*.

Pro správnou funkci služby bude třeba vyřešit autentizaci uživatele. V produkčním nasazení bude použita jedna z moderních bezpečných technologií, s největší pravděbodností to bude *django-rest-auth* nově pod názvem *dj-rest-aut*.[20]

Pro účely testování jsem službu nakonfiguroval pro použití authentizace pomocí tokenu, který lze získat příkazem z ukázky 6.13.

Výpis kódu 6.13 Příkaz pro získání autentizačního tokenu

```
> python manage.py drf_create_token <jmenouzivatele>
```

Kapitola 7

Návrh a implementace doporučovacího systému

7.1 Algoritmy

Možností, jak najít recepty pro uživatele, přibývá s tím, jaké a jak kvalitní jsou data. V následujících odstavcích se pokusím stručně popsat ty, které se mohou hodit.

7.1.1 Náhodný výběr

Princip je takový, že uživateli jsou doporučeny náhodné recepty, které ještě nezná, resp. se kterými doposud neinteragoval.

Náhodný výběr může být ve specifických případech překvapivě efektivní, není to ovšem pravidlem. Je rychlý a může sloužit jako referenční algoritmus při porovnávání efektivity. Užitečný je pak ten algoritmus, který dosáhne lepších výsledků v efektivitě než právě náhodný výběr.

7.1.2 Popularita

Pokud bychom chtěli dosáhnout obstojné spokojenosti uživatelů a efektivity bez složitých algoritmů, můžeme nabídnout všeobecně oblíbené recepty. Předpokládá se ovšem dostatek informací o oblíbených receptech a hrozí tzv. uváznutí v bublině a také problém s novými recepty, které nebudou nikdy doporučeny. Výhodou je jednoduchost a hlavně netrpí problémem u uživatelů s malým nebo žádným objemem informací, prakticky pro každého doporučí jen ty stejné, všeobecně oblíbené recepty.

7.1.3 Kolaborativní filtrování

Kolaborativní filtrování patří mezi nejvíce používané algoritmy doporučování. Trpí ale tzv. ”cold start”problémem. Ten vzniká pokud nemáme dostatek podobných uživatelů. Jedním z částečných řešení může být selekce těch uživatelů, u kterých dostatek dat máme. Pro uživatele, u kterých postrádáme informace o interakcích, můžeme využít výhod ostatních algoritmů jako je např. doporučení podle popularity7.1.2. V našem případě je třeba řešit také řídkost dat.

Jak bylo zmíněno v kapitole 2.2, lidé mající podobné chutě mohou mít v oblibě i podobné recepty.

Postup doporučení je následující:

1. z dat o hodnocení sestavíme matici uživatel x recept s hodnotami o hodnocení
2. zvolenou metrikou najdeme podobné uživatele, podle hodnot v matici
3. podíváme se na recepty, vyjma těch, které vybraný uživatel hodnotil, těchto podobných uživatelů
4. tyto recepty seřadíme je podle relevance a vrátíme prvních N

7.1.4 Content-based filtrování

Tam kde známe obsah (content) doporučovaných entit, můžeme využít content-based filtrování. V našem případě je obsahem receptu jeho složení ale i postup. Odpadá problém s novými recepty ale i uváznutí "v bublině".

Postup doporučení je následující:

1. obsah receptů transformujeme do vektorové reprezentace, v našem případě na recept pohlížíme jako na dokument popsaný ingrediencemi, postupem a slovním popisem
2. vytvoříme matici podobností (*similarity matrix*) říká nám, jak jsou si podobné recepty, každý s každým
3. pro daného uživatele získáme recepty, které má v oblibě
4. vypočítáme podobnost vektorů od uživatele s každým receptem a doporučíme prvních N nejrelevantnějších

Možnou nevýhodou tohoto přístupu je nutnost spočítat matici podobností při každém novém přidaném receptu. Vzhledem k tomu, že databáze čítá necelé dva tisíce receptů, nejde o akutní problém.

7.2 Doporučovací systém

Vzhledem k výše zmíněným problémům s nedostatkem dat a snaze vyhovět co možná největšímu množství uživatelů je moje řešení hybridní, kombinací doporučení podle popularity7.1.2 a *content-based* metody7.1.4. Tou podmínkou, která rozdělí běh algoritmu, je počet interakcí konkrétního uživatele.

Proces doporučení plánu rozděluji do tří fází a to filtrování7.2.1, hledání kandidátů7.2.2 a plánování7.2.3.

7.2.1 Preference

Každý uživatel má své preference a první myšlenka, když jsem začal na tomto pracovat, byla jak zajistit, aby byly uživatelské preference brány v potaz. Pokusil jsem se vcítit do role uživatele, který bude systém používat a představil jsem si následující scénář:

▶ **Příklad 7.1.** V případě, že bych jako preferenci vybral například možnost, že jsem vegetarián a systém by mi nabídl na oběd kuřecí roládu, myslel bych si, že systém nefunguje.

Z této úvahy jsem si odvodil funkcionalitu preferencí, která z mého pohledu má fungovat jako filtr. Buďto jako předřazený filtr, který nejdříve odstraní ty nevyhovující z celého balíku receptů. Nebo pokud je to vhodné pak jako "ad hoc"filtr, který odstranění provede až po výběru kandidátů.

Pokud by výše zmíněná úvaha 7.1 nemohla být brána jako chyba, dalo by se uvažovat o použití algoritmu kolaborativního filtrování podle preferencí uživatelů.

7.2.2 Hledání kandidátů

Tak jak to často v praxi bývá, výsledná implementace je hybridní. To jaký algoritmus je v danou chvíli použit rozhoduje vlastně, to v čem je ten či onen algoritmus silnější a navzájem tak korigují slabší stránky toho druhého.

7.2.2.1 Content-based implementace

Hlavním ”motorem hybridu”je content-based filtrování. Rozhodl jsem se pro tento algoritmus ze dvou důvodů. Prvním důvodem byla úvaha o receptech a hledání vzájemné podobnosti. Když se zamyslím, jaké dva recepty jsou téměř podobné, pak to jsou ty, které mají především stejné nebo podobné ingredience, stejný nebo podobný postup a pak další vlastnosti. Podmínkou samozřejmě byla skutečnost, že jsou k dispozici podrobná a kvalitní data. Druhým důvodem byl fakt, že nové recepty snáze proniknou do finálního doporučení, jelikož content-based filtrování netrpí tímto problémem jako je tomu kupříkladu u kolaborativního filtrování.

Z výčtu 7.1.4 zbývá objasnit především první dva body. Na recepty, jejich ingredience, popis a obsah pohlížím jako na dokument popisující daný recept. Použil jsem tedy dobře známou vektorovou reprezentaci textových dokumentů zkracovanou jako TF-IDF, což znamená Term-Frequency - Inverse Dokument Frequency. Díky tomu, že tato metoda je hojně využívána, je také mnoho volně dostupných knihoven pro práci s textem. Zvolil jsem si k tomu knihovnu *scikit-learn*.

Příkladem použití může být následující kus kódu[5]:

Výpis kódu 7.1 Vytvoření matice podobností

```
# ceske "stop words"
stop_words = Stopwords.objects.all()
self.stop_words = list(pd.DataFrame(stop_words.values())['word'])

# slouci slova do jedne radky
recipes['desc'] = recipes.title + ' ' \
                  + recipes.description + ' ' \
                  + recipes.edibles

self.vectorizer = TfidfVectorizer(analyzer = 'word',\
                                  ngram_range = (2, 3),\
                                  min_df = 0, \
                                  stop_words = self.stop_words)

tfidf_matrix = self.vectorizer.fit_transform(recipes['desc'])

self.sim_matrix = linear_kernel(tfidf_matrix, tfidf_matrix)
```

7.2.2.2 Doporučení podle popularity

U uživatelů, u kterých máme málo, nebo žádnou zpětnou vazbu, je těžké odvozovat, jaká jídla by jim mohly chutnat a není příliš vhodných algoritmů, které by byly schopny doporučit relevantní recepty. Vzhledem k celkovému počtu receptů kolem dvou tisíc by například náhodný výběr nemohl dosahovat dvakrát skvělých výsledků. Při průměru 13 oblíbených na uživatele je šance se trefit do chutí méně než 1%.

Vcelku dobrá volba v těchto krajních případech je doporučení populárních receptů. Pro některé datasety může být použití populárních receptů nejlepší volba a i pokročilejší algoritmy můžou mít problém popularitu v efektivitě porazit.

Jak ale v našem případě zjistit, které recepty jsou ty populární? Mám k dispozici hned dva datasety, které nám k nalezení mohou dopomoci. Data o hodnocení (rating) jsou poměrně

řídká, ovšem udávají jak moc je daný recept pro uživatele chutný. Druhou možností jsou data o oblíbených receptech, kterých je pro změnu výrazně více, na druhou stranu zde nemáme informaci o míře oblíbenosti a je faktem, že této funkce můžou uživatelé v některých případech využít jako úložiště pro pozdější podrobné zhlédnutí receptu, aniž by recept vyzkoušeli.

Lehce naivní přístup by byl získat počty v průměru nejlépe hodnocených receptů. A zde nastává problém v krajních situacích, např. u receptů, které mají třeba jen pár nebo jedno vynikajících hodnocení. Zcela jistě by se objevily mezi těmi nejlepšími.

Prostý součet hodnocení nebo uložení sice vystihuje popularitu v právem slova smyslu, tj. nejpopulárnější je ten nejznámější, ovšem nereflektuje jaký mají uživatelé k danému receptu názor.

Proto jsem se inspiroval váženým ratingem používaným u filmové databáze IMDB "weighted rating"[5].

$$WR = \frac{v}{v+m}R + \frac{m}{v+m}C \tag{7.1}$$

kde v je počet hodnocení, m je minimum hlasů, k tomu aby se recept objevil v seznamu, R je průměr hodnocení daného receptu a C je průměr hodnocení přes celý dataset.

7.2.3 Plánování

Když máme konečně seznam vhodných receptů zbývá provést už jen samotné sestavení plánu. To jak bude konečný jídelníček vypadat je nemálo ovlivněno parametrem *ratio*. Ten udává poměr počtu receptů známých K ku těm novým doporučovaným N. Pokud je počet receptů, které uživatel zná menší než $K * 35$, použijí se v jídelníčku všechny jemu známé recepty.

$$ratio = \frac{K}{N} \tag{7.2}$$

Jde o snahu ovlivnit jak bude celkový výsledek uživatel posuzovat. Důvěryhodnost, kterou bude systému uživatel přisuzovat závisí i na tom, jestli najde v jídelníčku oblíbené, známé nebo povědomé recepty.

Jestli bude tento parametr pro uživatele nastavitelný je na rozhodnutí vedení společnosti.

Ve chvíli, kdy máme přesně danou množinu receptů, které chceme uživateli sestavit do podoby jídelníčku, je třeba rozlišit ke kterému chodu konkrétní recept přiřadit. K tomu slouží metoda:

Výpis kódu 7.2 Rozdělení receptů do seznamu chodů

```
split_by_dish(self, recipes)
```

Tato metoda z parametru listu receptů vytvoří list listů receptů, rozdělených podle chodu. V současné verzi neobsahuje zvláštní logikou, pouze vytvoří popsanou strukturu, ovšem v další fázi vývoje bych si dovedl představit možnosti, které tato na první pohled jednoduchá operace naskýtá.

V poslední fázi plánování dochází k finálnímu umístění receptu na konkrétní den. Algoritmus zpracuje popořadě snídaně až večeře a v každém cyklu přiřadí recept tak jak jsou seřazeny v listu od pondělí do neděle. U svačin, které jsou od sebe nerozeznatelné - nevíme, jak odlišit dopolední od odpolední svačiny, je situace trochu jiná. U odpoledních svačin přiřazuji recepty z pozic 8 až 14.

Kapitola 8

Ověření vlastností doporučení

V této kapitole se pokusím ověřit úspěšnost vlastní varianty *Content-based* doporučovacího algoritmu a porovnat výsledky s vybranými alternativami. Nejprve je třeba se ale seznámit s metrikami používanými u offline evaluace.

8.1 Metriky

Obecně lze rozdělit metriky pro offline evaluaci na dvě skupiny. První skupina, do které patří např. RMSE a MAE, ověřuje explicitní data. Algoritmy se u těchto dat snaží predikovat míru relevance, například u hodnocení predikujeme jak by dané položky uživatel hodnotil. Odchylka predikce od skutečných hodnot pak dává výslednou chybu algoritmu. Druhá skupina metrik, jako zmíněné Precision@K, Recall@K nebo NDCG@K, vhodná i pro algoritmy pracujícími s implicitními daty, pak zkoumá ne chybovost, ale úspěšnost doporučených položek.[8][4]

8.1.1 Root Mean Squared Error

Široce používaná metrika také pro svou odolnost proti velkým chybám, které eliminuje pomocí odmocniny.

$$RMSE(X,Y) = \sqrt{\frac{\sum_{i=1}^{N}(x_i - y_i)^2}{N}} \tag{8.1}$$

kde x_i vyjadřuje skutečnou hodnotu i^{th} položky a y_i vyjadřuje predikovanou hodnotu i^{th} položky.

8.1.2 Mean Absolute Error

Oproti RMSE citlivější na velké rozdíly u odlehlých hodnot.

$$MAE(X,Y) = \frac{\sum_{i=1}^{N}|x_i - y_i|}{N} \tag{8.2}$$

8.1.3 Precision@K

Snadná na pochopení, výpočet i interpretaci. Definována je jako podíl relevantních položek r ku počtu doporučovaných:

$$Prec(k) = \frac{r}{k} \tag{8.3}$$

Říká kolik položek z k bylo zkoumaným algoritmem správně doporučeno.[17]

8.1.4 Recall@K

Je definován jako podíl relevantních položek r_k mezi k doporučených ku r celkovému počtu relevantních položek.

$$Recall(k) = \frac{r_k}{r} \tag{8.4}$$

8.1.5 Normalized Discounted Cumulative Gain @K

Tato metrika bere v potaz nejen pořadí, ve kterém jsou položky řazeny, ale také bere v potaz jejich počet.

$$DCG = \sum_{i=1}^{n} \frac{2^{relevance_i} - 1}{log_2(i+1)} \tag{8.5}$$

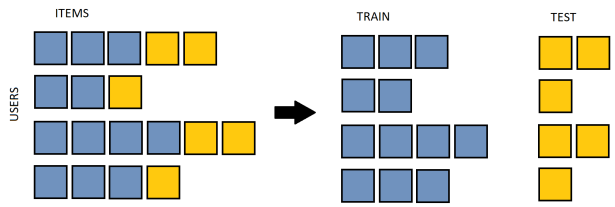
$$NDCG = \frac{DCG}{iDCG} \tag{8.6}$$

kde $iDCG$ značí hodnotu DCG pro ideální řešení[14][10], tím nabývá hodnot $[0, 1]$ a tedy odtud přívlastek *normalizovaný*.

8.2 Metodika

Před samotnou evaluací je třeba si data správně připravit. Princip je rozdělit dataset do trénovací a testovací množiny, naučit model na trénovacích datech a pro každého uživatele zkoumat výsledek predikce oproti testovacím datum. Z toho vyplývá, že potřebujeme, abychom měli interakce o uživateli jednak v trénovací množině, tak v té testovací. Popsaný princip rozdělení ilustruje obrázek 8.1.

Obrázek 8.1 Princip rozdělení dat do trénovacích a testovacích množin.



8.3 Výsledky

Pro naše účely jsem použil *Precision@K* pro svou jednoduchost a také proto, že ve výsledku nezkoumáme v jakém pořadí jsou recepty doporučeny. Ve fázi plánování jsou vybrané recepty rozloženy podle denní doby a v budoucnu podle jiných kvalit, než je relevance.

Pro evaluaci jsem použil třídy z knihovny *Case Recommender*[9] buďto přímo:

```
from caserec.evaluation.item_recommendation \
   import ItemRecommendationEvaluation

eval = ItemRecommendationEvaluation(sep = ',', metrics = ['PREC'])

eval.evaluate(predictions = predictions, test_set = test_set)
```

nebo jsem pro ostatní algoritmy[3] vytvořil trénovací a testovací datasety a s nimi volal metodu *compute()* na konstruktoru konkrétní implementace algoritmu. Jedinou výzvou bylo přijít na korektní strukturu parametrů metody *evaluate()*.

Tabulka 8.1 Rating dataset - Precision@10

algoritmus	plný dataset	>= 5 int.	>= 10 int.
random	0.001097	0.003537	0.012903
Item KNN	0.001463	0.007395	0.022581
User KNN	0.006155	0.028617	0.053763
Most Popular	0.011822	0.038264	0.05914

V tabulce 8.1, kde jsou vyneseny hodnoty měření pro *rating* dataset, lze vyčíst nevalné výsledky.

Tabulka 8.2 Saved dataset - Precision@10

algoritmus	plný dataset	>= 5 int.	>= 10 int.	>= 20 int.
random	0.003343	0.004878	0.00639	0.008771
Item KNN	0.07014	0.054767	0.065149	0.083418
User KNN	0.035402*	0.058652	0.07002	0.088868
Most Popular	0.046825	0.050033	0.06263	0.081121
SVD	0.034183	0.044879	0.056185	0.076873
Content-Base	0.019697	0.027774	0.034552	0.04413

Tabulka 8.3 Views dataset - Precision@10

algoritmus	plný dataset	>= 5 int.	>= 10 int.	>= 20 int.
random	0.004262*	0.004874	0.006038	0.007958
Item KNN	0.081353*	0.107041	0.128239	0.155714
User KNN	0.092518*	0.142367*	0.166998*	0.169084
Most Popular	0.08423*	0.125455*	0.15234*	0.150537
ALS	0.010616	0.009688	0.011259	0.012344
SVD	0.043123	0.053059	0.079689	0.106013

Na informačně bohatších datasetech *rating* a *views* jsou doporučení relativně úspěšné.

Hodnoty označené hvězdičkou (*) byly získány z dat tzv. samplováním s pomocí metody *random.sample()*.

8.4 Praktické ukázky

Praktickou a osvědčenou metodou evaluace doporučení je vizuální kontrola výtupu. Podívejme se tedy na to, jaké recepty nalezneme v pár příkladech využití vybraných algoritmů.

První věc, která mne zajímala, bylo jaké podobné recepty nalezne *content-based* metoda pro vybrané recepty. Odpovědi na tuto otázku ukazují tabulky 8.4, 8.5 a 8.6.

Tabulka 8.4 Recepty podobné receptu *Asijský coleslaw salát*, nalezené *content-based* metodou

název receptu	skóre
Zdravý coleslaw	0.098624
Salát coleslaw	0.083295
Salát z kysaného zelí s mrkví	0.076842
Wrap s tempehem, rýží a arašídovým dresinkem	0.069502
Cibulová marmeláda	0.069096
Domácí tatarka	0.065138
Salát z červeného zelí a mrkve	0.063717
Boršč	0.057223
Rajčatový dip	0.056897
Jogurtová ovesná kaše	0.054944

Tabulka 8.5 Recepty podobné receptu *Cottage bulky*, nalezené *content-based* metodou

název receptu	skóre
Ovesné lívance z cottage sýru	0.392314
Vafle s parmazánem	0.265125
Sýrové palačinky	0.241302
Cottage bulky	0.232472
Tuňákové muffiny	0.221840
Celozrnné linecké	0.184591
Hrášková pomazánka	0.178913
Ředkvičková pomazánka s cottage sýrem	0.163884
Red velvet layer cake	0.163877
Cuketové kuličky	0.162666

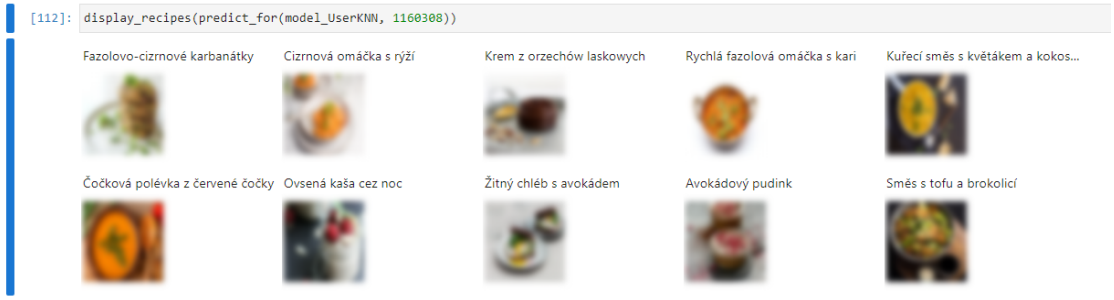
8.5 Závěr

Výsledky měření různých algoritmů a vlastní hybridní varianty jsou vcelku vypovídající. Varianty kolaborativního filtrování předčily implementovanou content-based metodu. Očekávání nebyla nijak přehnaná, přestože výstupy z prvotní verze systému byly slibné.

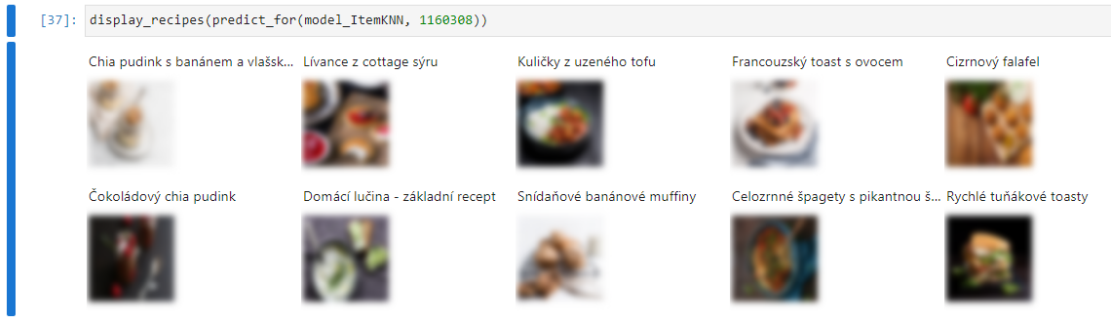
Tabulka 8.6 Recepty podobné receptu *Tvarohové knedlíky s meruňkami*, nalezené *content-based* metodou

název receptu	skóre
Špaldová linecká kolečka	0.099885
Tvarohový závin s jablky	0.099501
Malinová miska s chia	0.097894
Celozrnná vánočka	0.084484
Strouhaný koláč s tvarohovomakovou náplní a meru+nkou	0.079760
Tvarohové knedlíky	0.071530
Čokoládové muffiny z tvarohového těsta	0.069253
Japonský cheesecake	0.069187
Mandlové ovocné knedlíky	0.067333
Bublanina z kuskusu	0.065168

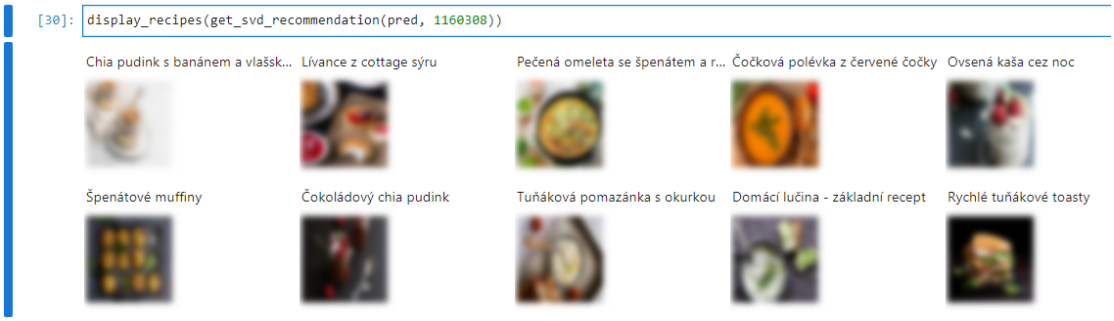
Obrázek 8.2 Ukázka doporučení Top-10 receptů pro vybraného uživatele různými algoritmy



(a) UserKNN



(b) ItemKNN



(c) SVD

(d) Content-based metoda

	snidane	dopoledni svacina	Obed	odpoledni svacina	vecere
Monday	Mrkvové overnight oats	Jahodový dortík	Krémové těstoviny s citronem a…	Nanuky z kešu	Kuřecí směs s kurkumou a kok…
Tuesday	Batoniki marchewkowo-owsiane	Mrkvové overnight oats	Jahodový dortík	Kakaový termix s jahodami	Domácí celozrnná pizza
Wednesday	Chia pudink s mangovým pyré	Pohanková bábovka s brusinkami	Pohanková bábovka s brusinkami	Tvarohový dort s jahodami	Nanuky z kešu
Thursday	Kakaový termix s jahodami	Batoniki marchewkowo-owsiane	Kuře s hráškovým pyré	Ovesné lívance z cottage sýru	Celozrnné penne s tuňákem
Friday	Ovesné lívance z cottage sýru	Chia pudink s mangovým pyré	Batoniki marchewkowo-owsiane	Rýžové palačinky	Mrkvové placky
Saturday	Rýžové palačinky	Palačinkový dort s jahodami	Celozrnné penne s tuňákem	Ředkvičková pomazánka	Čočka s bylinkami a bramborov…
Sunday	Tvarohový koláč s pudinkem	Palačinky se skořicí a jablky	Kuřecí směs s kurkumou a kok…	Tvarohový koláč s pudinkem	Minestrone

Obrázek 8.3 Ukázka vygenerovaného plánu

Kapitola 9
Zprovoznění

9.1 Běhové prostředí

Použití virtualizace je v dnešní době skoro standard a tato práce není výjimkou. Použití kontejnerové virtualizace s pomocí Dockeru je díky široké komunitní podpoře poměrně snadné. Návodů na vytvoření kontejneru je celá řada.[18][25]

Pro to vytvořit spustitelný kontejner odpovídající REST službě, kterou tato práce popisuje, stačí vytvořit textový soubor, tzn. *Dockerfile*. V tomto souboru se pak pomocí speciálních příkazu popíše z čeho kontejner vychází a co je třeba udělat pro sestavení a spuštění.

Výpis kódu 9.1 Dockerfile

```
FROM python:3.9
WORKDIR /var/www/mojerecepty
# set environment variables
ENV PYTHONDONTWRITEBYTECODE 1
# -> prevents Python from copying pyc files to the container.
ENV PYTHONUNBUFFERED 1
# -> ensures that Python output is logged to the terminal,
# making it possible to monitor Django logs in realtime.
COPY requirements.txt /var/www/mojerecepty
RUN pip install -r requirements.txt
COPY . /var/www/mojerecepty
EXPOSE 8000
CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]
```

9.1.1 Python 3.9

Při vytváření první verze jsem narazil na problém se závislostmi v knihovně *scikit-learn*. [1] Z toho důvodu jsem byl nucen definovat verzi Pythonu pro běh aplikace na *3.9* přestože byla vydána novější verze *3.10*.

9.2 Spuštění

Po nutné instalaci samotného *Docker enginu* je třeba nejprve sestavit kontejner pomocí příkazu *docker build*9.2. Kontejner vzniká podle instrukcí definovaných ve souboru pojmenovaném *Docker-*

[1] https://github.com/scikit-learn/scikit-learn/issues/21511

file[22], což je jakýsi návod pro engine na jeho sestavení. To provedeme příkazem z ukázky 9.2 v kořenovém adresáři zdrojového kódu. Přepínačem *tag* pojmenujeme kontejner, tak abychom se na něj mohli odkázat v následujícím kroku. Nesmíme zapomenout na poslední argument "." (tečka), kterým říkáme, že návod na sestavení, zmíněný *Dockerfile*, leží v tomto adresáři.[18]

Výpis kódu 9.2 Příkaz pro sestavení kontejneru: docker build

```
docker build --tag dipmealplanner:latest .
```

Po úspěšném sestavení zbývá spustit kontejner příkazem z ukázky 9.3, který vystaví serverovou službu lokálně na portu 8000.

Výpis kódu 9.3 Příkaz pro spuštění kontejneru: docker run

```
docker run --name dipmealplanner -d -p 8000:8000 dipmealplanner:latest
```

Kapitola 10
Závěr

V této práci jsem si vyzkoušel všechny fáze vývoje softwaru, od samotné specifikace požadavků až po závěrečné testování. Začalo se vymezením požadavků i zodpovědností za různé komponenty a nastínilo se očekávání. Následovalo první seznámení s dostupnými daty a podrobnější analýza, která ukázala relativní bohatost a ukázala se jako velmi slibná.

Podle požadavků se vytvořil datový model a definovalo se webové rozhraní. Obojí proběhlo důkladnou analýzou kvůli budoucím možnostem rozšíření a případných dalších přání investora. Vytvořil jsem základní verzi systému pro doporučení receptů spolu s tzn. MVP[1] pro prvotní ověření funkčnosti. Průběžné výsledky se ukázaly dostačující a tak se ve vývoji systému pokračovalo dále.

Současně v vývojem pak probíhala evaluace systému a zkoušely se další a další algoritmy pro doporučování. U některých vychází měřená efektivita výrazně lépe a tak se bude dále zkoumána a nelze vyloučit, že možná najde uplatnění jiné řešení, než je popsáno v této práci.

V dosavadnímu studiu jsem měl základní znalosti o doporučovacích systémech a díky této práci si je zopakoval a v praktických oblastech výrazně prohloubil, což je pro mne největším přínosem.

10.1 Pokračování vývoje

Odevzdáním práce vývoj na tomto projektu nekončí a možností rozvoje bylo více než dost už na začátku. Některé nápady jsou popsány zde:

- Zefektivnění algoritmu doporučujících recepty - buď to zahrnutím většího množství informací o receptech v současném řešení content-based algortimu, nebo nahrazením některou z implementací zkoumaných v kapitole 8.
- Uživatel si vybere pro kolik osob bude vařit a na kolik dní dopředu.
- S pomocí jednoduchých heuristik sledovat diverzitu celkového plánu.
- Variabilita ve vytváření plánu. Dát uživateli možnost sestavovat plán jen na některé dny nebo jen vybrané chody.
- Přidání dalších atributů k receptům, včetně kalorií, časové náročnosti vaření, apod. Podle nich pak nabídnout uživateli filtrování.
- Celý proces sestavení plánu by šlo vylepšit, jak je zmíněno u metody 7.2. Například skládáním receptů tak, aby bylo možné je vařit na více dní.

[1] Minimum viable product[24], česky: minimální životaschopný produkt

Bibliografie

1. PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M.; DUCHESNAY, E. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*. 2011, roč. 12, s. 2825–2830.
2. CHRISTIE, Tom. *Web API performance: profiling Django REST framework*. 2013. Dostupné také z: `https://www.dabapps.com/blog/api-performance-profiling-django-rest-framework/`.
3. ROSENTHAL, Ethan. *Intro to Implicit Matrix Factorization: Classic ALS with Sketchfab Models*. 2016. Dostupné také z: `https://www.ethanrosenthal.com/2016/10/19/implicit-mf-part-1/`.
4. ARGUELLO, Jaime. *Evaluation Metrics*. 2017. Dostupné také z: `https://ils.unc.edu/courses/2013_spring/inls509_001/lectures/10-EvaluationMetrics.pdf`.
5. BANIK, ROUNAK. *Movie Recommender Systems*. 2017. Dostupné také z: `https://www.kaggle.com/code/rounakbanik/movie-recommender-systems/notebook`.
6. ÇANO, Erion. Hybrid Recommender Systems: A Systematic Literature Review. *Intelligent Data Analysis*. 2017, roč. 21, s. 1487–1524. Dostupné z DOI: `10.3233/IDA-163209`.
7. FREDERICKSON, Ben. *Implicit Python library*. 2017. Dostupné také z: `https://implicit.readthedocs.io/en/latest/`.
8. MALAEB, Maher. *Recall and Precision at k for Recommender Systems*. 2017. Dostupné také z: `https://medium.com/@m_n_malaeb/recall-and-precision-at-k-for-recommender-systems-618483226c54`.
9. COSTA, Arthur da; FRESSATO, Eduardo; NETO, Fernando; MANZATO, Marcelo; CAMPELLO, Ricardo. Case Recommender: A Flexible and Extensible Python Framework for Recommender Systems. In: *Proceedings of the 12th ACM Conference on Recommender Systems*. Vancouver, British Columbia, Canada: ACM, 2018, s. 494–495. RecSys '18. ISBN 978-1-4503-5901-6. Dostupné z DOI: `10.1145/3240323.3241611`.
10. GHADIALLY, Husain. *Is This What You Were Looking For?* 2019. Dostupné také z: `https://www.gojek.io/blog/is-this-what-you-were-looking-for`.
11. CHRISTOPHER MANNING, Pandu Nayak. *Lecture: introduction to information retrieval*. 2019. Dostupné také z: `https://web.stanford.edu/class/cs276/19handouts/lecture6-tfidf-6per.pdf`.

12. RASCHKA, S.; MIRJALILI, V. *Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2, 3rd Edition*. Packt Publishing, 2019. ISBN 9781789958294. Dostupné také z: https://books.google.cz/books?id=sKXIDwAAQBAJ.
13. HUG, Nicolas. Surprise: A Python library for recommender systems. *Journal of Open Source Software*. 2020, roč. 5, č. 52, s. 2174. Dostupné z DOI: 10.21105/joss.02174.
14. CHANDEKAR, Pranay. *Evaluate your Recommendation Engine using NDCG*. 2020. Dostupné také z: https://towardsdatascience.com/evaluate-your-recommendation-engine-using-ndcg-759a851452d1.
15. SCIFORCE. *Inside recommendations: how a recommender system recommends*. 2020. Dostupné také z: https://medium.com/sciforce/inside-recommendations-how-a-recommender-system-recommends-9afc0458bd8f.
16. ALI, Iman. *AI Powered Search and Recommendation System*. 2021. Dostupné také z: https://www.lumenci.com/post/ai-powered-search-and-recommendation-system.
17. BRIDEAU, Ryan. *Precision@k: The Overlooked Metric for Fraud and Lead Scoring Models*. 2021. Dostupné také z: https://towardsdatascience.com/precision-k-the-overlooked-metric-for-fraud-and-lead-scoring-models-fabad2893c01#.
18. PAUL, Odhiambo. *How to Create Django Docker Images*. 2021. Dostupné také z: https://www.section.io/engineering-education/django-docker/.
19. ANDREAS PELME, contributors. *Configuring Django settings*. 2022. Dostupné také z: https://pytest-django.readthedocs.io/en/latest/configuring_django.html#the-environment-variable-django-settings-module.
20. *Authentication*. 2022. Dostupné také z: https://www.django-rest-framework.org/api-guide/authentication/#django-rest-auth-dj-rest-auth.
21. *Django REST Frameork Doc*. 2022. Dostupné také z: https://www.django-rest-framework.org/#development.
22. *Dockerfile reference*. 2022. Dostupné také z: https://docs.docker.com/engine/reference/builder/.
23. *Migrations*. 2022. Dostupné také z: https://docs.djangoproject.com/en/4.0/topics/migrations/.
24. *Minimum viable product*. 2022. Dostupné také z: https://cs.wikipedia.org/wiki/Minimum_viable_product.
25. *Quickstart: Compose and Django*. 2022. Dostupné také z: https://docs.docker.com/samples/django/.
26. W3TECHS. *Overview*. 2022. Dostupné také z: https://pandas.pydata.org/docs/getting_started/overview.html.
27. RUBY, Leonard Richardson; Sam. *RESTful Web Services*. O'Reilly Media, Inc., [b.r.]. ISBN 9780596529260. Dostupné také z: https://archive.org/details/restfulwebservic00rich_0.

Obsah přiloženého média

- `thesis.pdf` ..... text práce ve formátu PDF
- `readme.txt` ..... poznámky ke spuštění
- `latex` ..... zdrojová forma práce ve formátu LaTeX
- `src` ..... zdrojové kódy
- `resources` ..... podklady k práci
  - `notebooks` ..... IPython notebooky