FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Doporučovací systém pro bankovní aplikaci
Student:	Bc. Miloš Popovič
Vedoucí:	Ing. Jaroslav Kuchař, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Webové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2024/2025

Pokyny pro vypracování

Cílem práce je vytvoření doporučovacího systému v podobě mikroslužby pro firemní bankovní klienty, kteří vyžadují individuální výběr bankovních produktů a přizpůsobené poplatky za jejich využití na základě intenzivněji využívaných služeb, zvýšeného počtu transakcí, dosažení konkurenceschopné nabídky a podobně. Systém dokáže na základě specifikace klienta doporučovat vybrané produkty za konkrétní poplatky na základě porovnávání s podobnými klienty.

- Proveďte rešerši problematiky doporučovacích systémů a proveďte analýzu existujících řešení pro účely bankovní aplikace.
- Analyzujte firemní bankovní doménu, data a procesy o klientech a bankovních produktech.
- Navrhněte a implementujte doporučovací systém nad vybranou datovou sadou.
- Nastudujte si architektonický vzor architektury mikroskužeb a podle něj navrhněte a naimplementujte komponentu, která bude schopna interagovat s ostatními komponentami v bance.
- Výsledné řešení řádně otestujte, vyhodnoťte kvalitu doporučovaných výstupů formou experimentu a diskutujte vhodnost svého řešení.

Elektronicky schválil/a Ing. Jaroslav Kuchař, Ph.D. dne 14. září 2023 v Praze.

Diplomová práca

DOPORUČOVACÍ SYSTÉM PRO BANKOVNÍ APLIKACI

Bc. Miloš Popovič

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedúci: Ing. Jaroslav Kuchař, Ph.D.
10. januára 2024

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam výpisov kódu

Chcel by som sa poďakovať Ing. Jaroslavovi Kuchařovi, Ph.D., vedúcemu tejto práce, za jeho odbornú pomoc, vynaložený čas a objektívnu kritiku počas celej doby tvorenia práce.

Vyhlásenie

Vyhlasujem, že som predloženú prácu vypracoval samostatne a že som uviedol všetky použité informačné zdroje v súlade s Metodickým pokynom o dodržiavaní etických princípov pri príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Zb., autorského zákona, v znení neskorších predpisov, najmä skutočnosť, že České vysoké učení technické v Prahe má právo na uzatvorenie licenčnej zmluvy o použití tejto práce ako školského diela podľa § 60 ods. 1 citovaného zákona.

V Prahe dňa 10. januára 2024

Abstrakt

Cieľom tejto práce je návrh a implementácia odporučacieho systému v podobe mikroslužby zameraného na firemných bankových klientov. V teoretickej časti práce je zanalyzovaná bankova doména, dáta a procesy, je zrealizovaná rešerš nad odporúčacími systémami kolaboratívneho filtrovania a je popísaná architektúra mikroslužieb. V praktickej časti práce sú vykonané experimenty nad zvolenými algoritmami, je navrhnutá architektúra mikroslužieb pre aplikáciu na navrhovanie a odporúčanie produktov, a je naimplementovaná mikroslužba pre odporúčanie. Výsledkom experimentov je hybridný algoritmus v podobe zhlukovania aplikovanom na analytické údaje klientov a aplikovania podobnostných funkcií používateľsky založeného filtrovania na údaje o produktoch a poplatkoch. Ďalej bolo navrhnutých päť nových mikroslužieb, ktoré boli úspešne zakompované medzi ostatné bankové systémy. Na záver bola úspešne naimplementovaná mikroslužba pre odporúčanie pomocou asynchrónneho princípu, sprostredkovateľa správ a webového aplikačného rámca pre jazyk Python.

Klíčová slova odporúčací systém, architektúra mikroslužieb, kolaboratívne filtrovanie, firemné bankové produkty, Python

Abstract

The goal of this work is to design and implement a recommender system in the form of a microservice aimed at corporate banking clients. The theoretical part analyzes the banking domain, data and processes, conducts research on collaborative filtering recommendation systems, and describes the architecture of microservices. In the practical part, experiments are conducted with selected algorithms, the architecture of microservices for an application for designing and recommending products is proposed, and a recommendation microservice is implemented. The result of the experiments is a hybrid algorithm in the form of clustering applied to clients' analytical data and the application of similarity functions of user-based filtering to product and fee data. Furthermore, five new microservices were designed and successfully integrated among other banking systems. Finally, a microservice for recommendation was successfully implemented using an asynchronous principle, a message broker and a Python web application framework.

Keywords recommender system, microservices architecture, collaborative filtering, corporate banking products, Python

Úvod

V modernom svete podnikania je rola bánk nepostrádateľná a formuje základ všetkých finančných operácií. Pre prosperitu firmy je nevyhnutné zapojenie bankových transakcií a využívanie bankových služieb. Aby banka adekvátne uspokojila potreby rôznorodých klientel, musí ponúkať širokú škálu produktov navrhnutých tak, aby každý produkt napĺňal jedinečné požiadavky zákazníkov. V súčasnosti banky rozširujú svoju úlohu nad rámec bežných poskytovateľov finančných produktov a ich cieľom je, aby pôsobili ako firemní poradcovia a pomáhali firmám zvýšiť ich prosperitu a rast prostredníctvom finančného poradenstva.

Aby banky mohli efektívne fungovať ako finanční poradcovia, je nevyhnutná schopnosť analyzovať rozsiahle množstvo údajov o klientoch, čo je kľúčové pre pochopenie jedinečných finančných potrieb a situácie každého klienta. Okrem analýzy dát je rovnako dôležitá schopnosť extrakcie a interpretácie získaných dát pre ich následné zmysluplné využitie. Na základe týchto požiadaviek vznikla potreba vytvorenia odporúčacieho systému pre firemných bankových klientov, ktorý dokáže odporúčiť jedinečnú skladbu produktov vhodných pre konkrétneho klienta a individuálnu cenu za dané produkty tak, aby bola poskytnutá bezkonkurenčná ponuka pre klienta, ale zároveň aby bola pre banku zachovaná udržateľná profitabilita z daného klienta. Samotné odporúčania budú slúžiť bankérom ako pomôcka pri navrhovaní individuálnych klientských nastavení.

Hlavným cieľom tejto práce je návrh a implementácia popísaného odporúčacieho systému. Pre jeho vytvorenie je nutné vykonať rešerš nad problematikou odporúčacích systémov, na základe ktorej je možné vybrať vhodné modely pre aplikovanie na tejto problematike. Okrem toho je nutné zanalyzovať bankovú doménu, aby bolo možné navrhnúť algoritmus s kontextom danej domény. Následne je možné vykonať transformáciu klientských dát a vykonať experimenty pre získanie optimálneho algoritmu pre odporúčanie produktov. Ďalším hlavným cieľom práce je navrhnúť a naimplementovať mikroslužbu pre odporúčanie produktov v kontexte ostatných služieb v rámci domény tak, aby bola s nimi schopná interagovať. Pre dosiahnutie tohto cieľa je nutné naštudovanie bankových procesov a aplikácii, a definovanie požiadaviek na aplikáciu pre odporúčanie a navrhovanie nastavení produktov klinta. Pre dosiahnutie kvalitného architektonického návrhu mikroslužieb je taktiež potrebné si preštudovať daný architektonický vzor mikroslužieb.

Prvá kapitola je venovaná popisu bankovej domény, klientských dát, bankových procesov a analýze požiadaviek na aplikáciu. V ďalšej kapitole je vykonaná rešerš nad odporúčacími systémami kolaboratívneho filtrovania a technikám merania a vyhodnotenia algoritmov. Posledná kapitola teoretickej časti sa venuje architektúre mikroslužieb, jej požiadavkám, charakteristikám a vzorom. V štvrtej kapitole je na základe získaných znalostí vykonaná príprava klientských dát a následne sú vykonané experimenty nad konkrétnymi odporúčacími algoritmami, po ktorých nasleduje vyhodnotenie a výber vhodného algoritmu. V ďalšej kapitole je navrhnutá architektúra mikroslužieb s diskusiou nad novovzniknutými službami. Na záver je naimplementovaná samotná mikroslužba pre odporúčanie s vybraným algoritmom a v kontexte ostatných bankových systémov a následným vyhodnotením návrhu a implementácie.

Kapitola 1
Doména

Táto kapitola je venovaná popisu bankovej firemnej domény, ktorej získaný kontext je nevyhnutný pre správne fungovanie odporúčacích algoritmov a kvalitný návrh architektúry systému. Prvá časť je venovaná popisu dát, ktoré slúžia ako vstup a výstup odporúčacieho systému. Následne sú popísané existujúce bankové procesy, na základe ktorých je navrhnutá architektúra a naimplementovaná služba pre odporúčanie. Na záver sú zhrnuté požiadavky pre aplikáciu na odporúčanie a nastavovanie produktov, ktoré reflektuje navrhnutá architektúra v praktickej časti práce.

1.1 Dáta

Pre návrh kvalitného odporúčacieho systému je potrebné mať zdokumentované dáta, ktoré budú vstupom a výstupom pre odporúčacie algoritmy. Táto časť sa venuje analýze produktov a poplatkov za produkty, ktoré budú vstupom a výstupom odporúčacieho systému. Ako vstup pre odporúčací algoritmus poslúžia taktiež dáta z analytického úložiska pre získanie kontextu klienta, čomu je venovaná ďalšia sekcia. Dôležitou súčasťou kontextu bankovej domény je taktiež analýza klientov a ich segmentácia, ktorej je venovaná posledná časť podkapitoly.

1.1.1 Produkty a poplatky

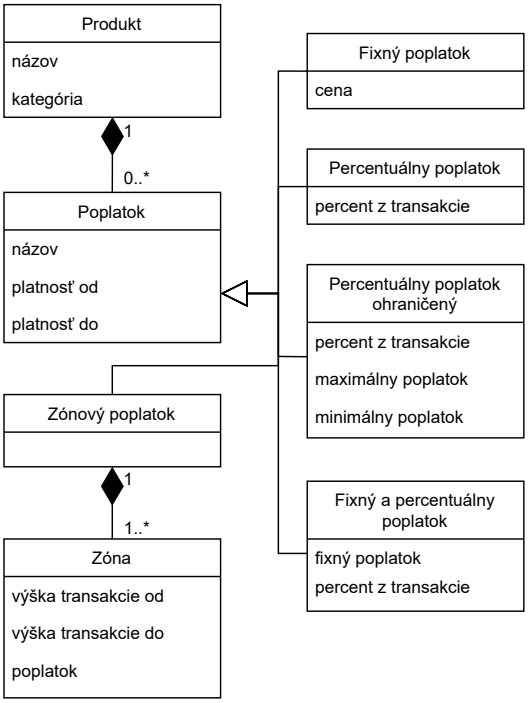
Táto časť sa venuje analýze produktov, ktoré budú klientom odporúčané za individuálnych podmienok po splnení konkrétnych kritérií. Individuálne podmienky sa najčastejšie prejavujú na znížení alebo zvýšení poplatkov za využívané produkty. Pri odporúčaní teda bude kladený dôraz na nastavovanie cien za využívané produkty. Tieto poplatky je podľa cenníkov pre firemných klientov [1, 2, 3] možné zväčša rozdeliť do nasledujúcich kategórií:

- vedenie účtu (založenie účtov, správa účtov, výpisy, kreditné a debetné operácie, cudzomenové účty);
- bezhotovostný platobný styk (prichádzajúce a odchádzajúce platby, trvalé platby, potvrdenia o platbe, inkasá, prioritné úhrady);
- hotovostné operácie (výber hotovosti, vklad bankoviek, zvozy hotovosti);
- priame bankovníctvo (internet banking, pripojenie k API, MultiCash[1], certifikáty, SMS);

[1] Medzinárodný multibankový systém pre obsluhu účtov, ktoré sú vedené v rôznych bankách na území Českej republiky aj v zahraničí

- bankové záruky, akreditíva (návrh, avízovanie, zmena, uplatnenie);
- platobné karty (debetné a kreditné karty, výber hotovosti z bankomatu, platby u obchodníka, úvery na kreditných kartách);
- úvery (žiadosti o úver, poskytnutie úverov, správa úverov);
- cenné papiere, sporenia a investície (obstaranie, na burze alebo finančných trhoch, správa aktív, predaj).

Jeden z príkladov, kde je žiaduce vytvoriť individuálne podmienky pre klienta je, že klient vlastní desiatky až stovky účtov, a teda nie je žiaduce, aby klient platil tisíce korún mesačne za vedenie účtov. Ďalším z príkladov môže byť, že klient vlastní platobný terminál s vysokým množstvom mesačných transakcií, a teda prijme ponuku banky, ktorá mu poskytne najlacnejší poplatok za transakciu, t. j. banky sú nútené poskytovať konkurencieschopné podmienky. Je žiaduce poznamenať, že v cenníkoch bánk nie je pri niektorých položkách uvedená cena a banky vopred očakávajú, že sa na poplatku dohodnú s klientom individuálne.



Obr. 1.1 Doménový model na základe zanalyzovaných údajov

Napriek tomu, že kategorizácia produktov a poplatkov za produkty a ich dôkladný popis môže patriť medzi užitočné metadáta pre odporúčací systém a môže zoptimalizovať niektoré algoritmy, dôležitým vstupom pre algoritmus je forma, akou je poplatok zadaný. Systém pre odporúčanie produktov musí byť schopný tieto informácie spracovať, transformovať do optimálnej podoby a prípadne normalizovať pre ďalšie použitie (pozri kapitolu 2). Na základe analyzovaných dát z cenníkov bankových produktov je možné kategorizovať typy poplatkov nasledovne:

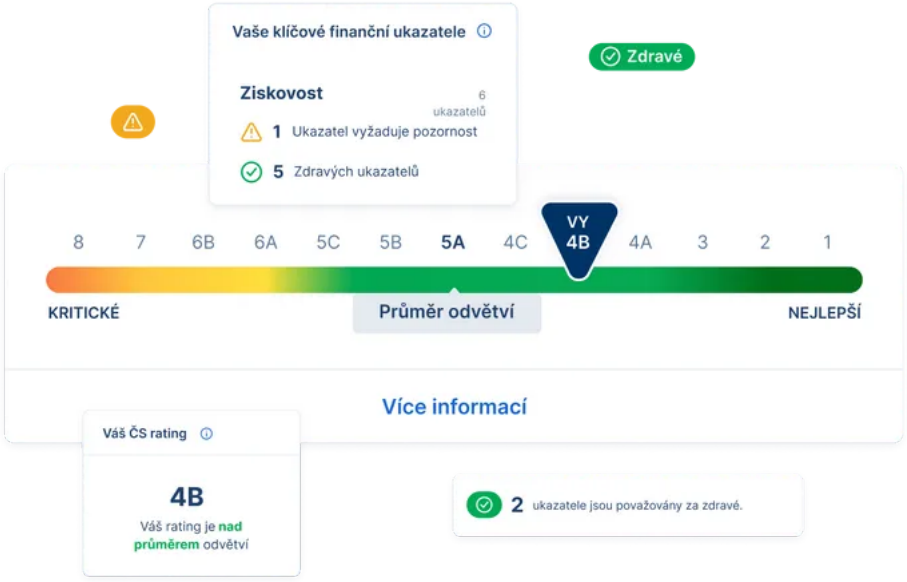
- fixný poplatok (napr. 180 CZK za vedenie účtu),
- percentuálny poplatok z transakcie (napr. 0.3% z nákupu podielového fondu),
- ohraničený percentuálny poplatok z transakcie (napr. spracovanie inkasa za 0.3% z transakcie, minimálne 1 000 CZK a maximálne 10 000 CZK),

- fixný poplatok kombinovaný s percentuálnym (napr. výber hotovosti z bankomatu v zahraničí za 150 CZK plus 0.5% z vyberanej čiastky),
- zónový poplatok, kde výška poplatku závisí na výške transakcie (napr. pri zvoze bankoviek je poplatok určený na základe objemu hotovosti, kde zóny sa delia na 0 CZK až 999 999 CZK, 1 000 000 CZK až 2 999 999 CZK alebo 3 000 000 CZK až 5 000 000 CZK).

Na obrázku 1.1 je na základe zistených informácií zobrazený možný bankový doménový model pre produkty a poplatky.

1.1.2 Dátový sklad

Dátový sklad (angl. Data Warehouse, DWH) je podniková dátová platforma používaná na analýzu a správu štruktúrovaných a pološtruktúrovaných dát z viacerých zdrojov dát, ako sú transakcie na mieste predaja, automatizácia marketingu, riadenie zákazníckych vzťahov a podobne [4]. Keďže je bankové prostredie zložené z mnohých rôznych systémov, dátový sklad je žiaducim riešením pre analyzovanie klientských dát a získavanie užitočných informácii, a teda pre bankovú doménu existujú aj predpripravené modely a riešenia pre implementáciu [5]. Ako príklad môže slúžiť [6], kde banka analyzuje dáta firemných klientov pre zistenie finančného zdravia firiem. Takéto analytické dáta môžu byť užitočné pre odporúčací systém, ktorý ich môže použiť na porovnávanie medzi klientmi. Podnikateľská klientská sféra môže byť rozdelená na segmenty na základe kritérií ako ekonomická aktivita, obrat spoločnosti, postavenie na trhu a vlastnícke vzťahy [7]. Tieto údaje sú teda jedny z najdôležitejších údajov, ktoré je žiaduce mať zahrnuté v analytickom úložisku a niektoré algoritmy môžu na základe týchto kritérií rozdeliť údaje na niekoľko skupín.



Obr. 1.2 Ukážka využitia zanalyzovaných dát firemných bankových klientov pre zistenie finančného zdravia firiem [6]

1.1.3 Klienti

Pre správne fungovanie odporúčacích algoritmov je potrebné nájsť atribúty, na základe ktorých je možné čo najpresnejšie rozdeliť firemných klientov do konkrétnych kategórií. Banky si udržiavajú

špecifické informácie o klientoch a následne ich rozdeľujú na viaceré segmenty, ktoré určujú, aké produkty môžu klientom ponúknuť a aké môžu klientovi poskytnúť cenové podmienky.

Jedno z najzákladnejších rozdelení klientov je rozdelenie podľa obratu klienta. Komerčná banka podľa [8] rozdeľuje klientov do nasledujúcich kategórií:

- malé podniky s obratom od 25 000 000 CZK do 60 000 000 CZK,
- stredné podniky s obratom od 60 000 000 CZK do 300 000 000 CZK,
- veľké podniky s obratom väčším než 300 000 000 CZK.

Okrem segmentácie podľa obratu existuje podľa [8] segmentácia podľa potreby ponuky finančných produktov banky. V tomto segmente sa produkty členia do kategórií ako firemné účty, leasing strojov, zariadení, informačných technológií, realít, fondy kolektívneho investovania, úvery, karty, záruky, trade financie, cash pooling, pomoc a poradenstvo, štruktúrované vklady, podielové fondy, obchodovanie s akciami, dlhopismi, emisnými povolenkami a podobne.

Z hľadiska segmentácie na základe údajov klienta, banka zbiera podľa [8] údaje o každom firemnom klientovi, ktoré môžu byť atraktívne pre odporúčací algoritmus, ako meno, adresa alebo sídlo spoločnosti, počet zamestnancov, mesačný alebo ročný obrat, predmet činnosti alebo úvery.

Celkovo je na českom trhu viac ako 500 000 firiem. S prihliadnutím na počet bánk a veľkosť jednotlivých bánk môže mať jedna dátová sada každej banky rádovo desiatky až stovky tisíc záznamov. Tento údaj je možné zohľadniť pri navrhovaní výpočetne náročnejších algoritmov.

1.2 Proces vyjednávania cenových podmienok

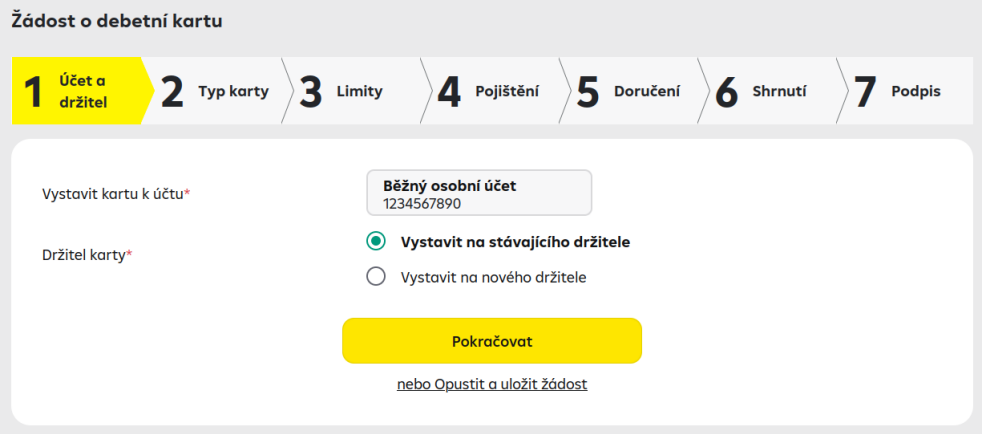
Každý klient, ktorý chce požiadať banku o individuálne nastavenie používaných produktov a cien za produkty, musí podstúpiť proces vyjednávania a schvaľovania. Táto podkapitola je venovaná analýze procesnému bankovému systému a krokom, ktoré klient podstupuje v danom procese. Tieto údaje je nutné zanalyzovať najmä kvôli návrhu architektúry mikroslužieb, ktorej systémy musia byť schopné spolupracovať s ostatnými systémami v bankovom prostredí.

1.2.1 Systém pre správu procesov

Každá banka potrebuje pre správne fungovanie množstvo procesov, ktoré slúžia na obsluhu klienta, pomáhajú bankérom v ich práci a zaisťujú auditovateľnosť operácií. Aby banky zvládli tieto procesy vyriešiť efektívne, väčšinou na to potrebujú róbustný systém skladajúci sa z viacerých komponentov. V tejto časti sú zanalyzované niektoré bankové systémy [9, 10] na základe vypozorovaných funkcionalít z používania aplikácií.

Pre účely tejto práce sa predpokladá, že aplikácie pre správu procesov v analyzovaných bankách sú postavené na základe architektúry mikroslužieb. Tieto aplikácie musia pre svoje fungovanie pozostávať minimálne z nasledujúcich služieb:

- Služba pre správu žiadostí. Táto služba obsahuje zakladanie žiadosti, posun medzi jednotlivými stavmi žiadosti a riešenie chybovej obsluhy žiadosti. Tento proces sa dá chápať aj ako prechod medzi stavmi v deterministickom konečnom automate. Služba vysiela správy ostatným systémom pri každom prechode a zároveň prijíma správy v prípade, že úloha v konkrétnom stave bola odbavená a žiadosť sa môže presunúť do ďalšieho stavu.
- Služba pre ponuku produktov. Táto služba je zodpovedná za poskytnutie ponuky produktov pre konkrétneho klienta v danom kontexte. Napríklad v ukážke na obrázku 1.3 sa môže jednať o poskytnutie účtov klienta a zoznam kartových produktov, ktoré je možné poskytnúť klientovi vrátane konfigurácie karty ako sú limity a podobne.



Obr. 1.3 Ukážka bankového systému pre správu procesov [10]

- Služba pre správu údajov klienta. V žiadostiach je často nutné vyplniť adresu alebo iné údaje pre daný produkt.
- Služba pre notifikovanie. Klientovi môže byť odoslaná správa rôznymi kanálmi, že žiadosť bola schválená alebo je potrebná ďalšia kooperácia. Notifikovanie môže byť taktiež potrebné pre bankéra, od ktorého je žiadané vykonať niektoré kroky v žiadosti, prípadne notifikovanie zákazníckej podpory, že v procese nastal problém.
- Služba pre generovanie dokumentácie. Banka generuje množstvo dokumentov, ktoré je žiaduce udržiavať na jednom mieste, verzovať a zachovávať jednodný formát v rámci celej banky. Služba je teda zodpovedná za vyplnenie šablóny údajmi zo žiadosti a vygenerovanie konkrétnych dokumentov.
- Služba pre podpisovanie dokumentov. V súčasnosti existuje viacero spôsobov, ktorými je možné dokumenty podpísať, ako elektronický podpis v internetovom bankovníctve alebo pomocou kvalifikovaného certifikátu, alebo ručne na pobočke. Každá z týchto možností vyžaduje rôzne úkony a je taktiež žiaduce spravovať históriu podpisovania na jednom mieste.
- Služba pre riadenie vzťahov so zákazníkmi (angl. Customer Relationship Management, CRM). Táto služba môže zahŕňať zákaznícku podporu, správu predaja a marketingu.
- Ďalšie služby pre obsluhu konkrétnych procesov. Napríklad v ukážke na obrázku 1.3 sa môže jednať o spracovanie žiadosti po podpísaní dokumentácie vrátane objednania karty a jej odoslania.

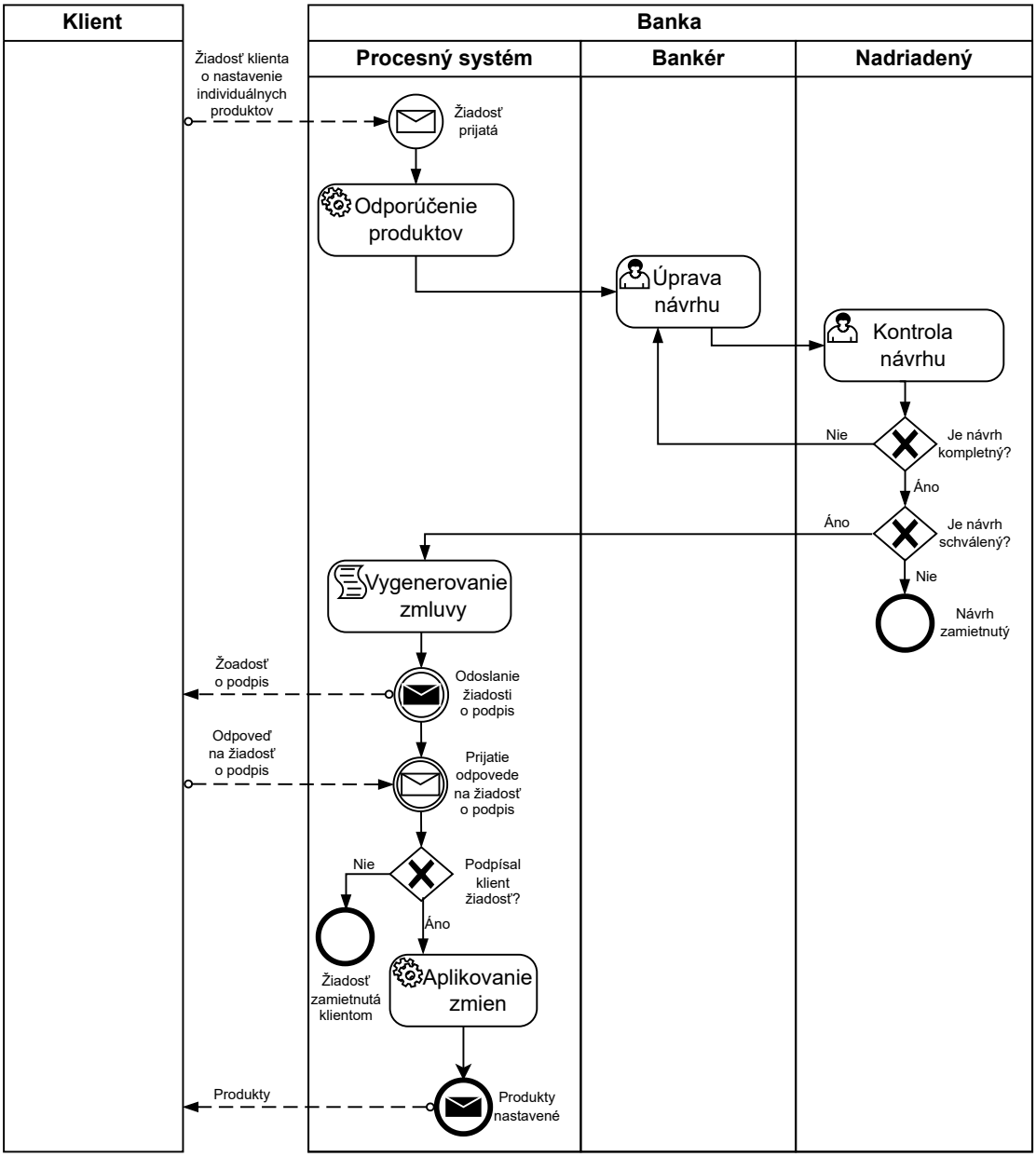
Tieto zdieľané služby môžu byť následne ďalej prepoužité v iných systémoch a môžu byť jednoducho rozšírené o ďalšie služby, ktoré budú obsluhovať nové procesy v banke, ako to je navrhnuté v kapitole 5.

1.2.2 Kroky žiadosti

Ako bolo zmienené v časti 1.2.1, každý proces je možné chápať ako deterministický konečný automat, kde stavy symbolizujú jednotlivé úkony, ktoré je potrebné vykonať pre úspešné dokončenie žiadosti. V tejto časti sú popísané potrebné kroky pre úspešné dokončenie žiadosti o vyjednanie individuálnych cenových podmienok.

Po tom, čo si klient zažiadal o vytvorenie individuálnej ponuky produktov, je vygenerovaný návrh produktov pomocou odporúčacieho systému. Vytvorenie systému pre tento krok je cieľom

tejto práce. Následne musí byť návrh odoslaný bankérovi, ktorý vykoná prípadné zmeny v návrhu. V tomto kroku by bolo žiaduce zaznamenať, aké zmeny bankér v návrhu vykonal a na základe tejto spätnej väzby vylepšiť systém. Následne, návrh typicky podlieha schváleniu nadriadeným a prebiehajú prípadné úpravy na základe podnetov. Po tom, čo je návrh schválený, je nutné, tak ako v každom procese, schválenie a podpísanie žiadosti s navrhnutým nastavením klientom. Následne je možné zmeny aplikovať v interných systémoch na základe zvolených kritérií. Celý proces je znázornený v BPMN diagrame na obrázku 1.4.



Obr. 1.4 BPMN diagram pre proces vyjednávania cenových podmienok

1.3 Analýza požiadaviek aplikácie

Pre úspešné vytvorenie návrhu a implementácie aplikácie je potrebné vykonať analýzu požiadaviek pre definovanie cieľov a ohraničení systému. V tejto časti budú zanalyzované najdôležitejšie funkčné a nefunkčné požiadavky pre aplikáciu na získavanie a úpravu cenových podmienok klienta.

1.3.1 Funkčné požiadavky

Cieľom funkčných požiadaviek je špecifikovať, aké operácie a funkcie by mala aplikácia vykonávať. Tieto požiadavky definujú očakávané správanie systému a sú kľúčovým nástrojom na zabezpečenie toho, že softvér spĺňa potreby a očakávania jeho užívateľov a zainteresovaných strán. V tejto časti sú popísané hlavné funkčné požiadavky kladené na túto aplikáciu.

F1 – Získanie cenníka Systém bude schopný vrátiť aktuálny zoznam bankových produktov v hierarchickej podobe podľa typu produktu a zobraziť názov produktu a poplatok za daný produkt v jeho forme. Systém bude schopný vrátiť časť alebo celý cenník na základe filtrovania alebo privilégií používateľa. Systém bude vraciať metadáta daného produktu slúžiace pre ďalšiu konfiguráciu a filtráciu.

F2 – Úprava cenníka Systém bude schopný upraviť názvy, poradie, metadáta a skladbu produktov a ich poplatkov na základe vstupu používateľa. Systém overí validitu vstupného dokumentu v podobe obmedzení kladených na dané poplatky, správny číselný formát alebo nepovolenia nepovolených konfigurácií pre daný produkt. Systém umožní nahrať novú verziu cenníka s platnosťou pre aktuálny dátum alebo budúci dátum. V prípade, že pre konkrétny dátum bude nahraných viac verzií cenníka, bude použitá vždy posledná platná nahraná verzia.

F3 – Získanie údajov o klientovi Systém bude schopný získať si údaje o konkrétnom klientovi a zobraziť produkty s kontextom daného klienta. Systém zobrazí predvolene len tie produkty, ktoré klient používa a skryje tie produkty, ktoré danému klientovi nie je možné poskytnúť.

F4 – Získanie aktuálnej konfigurácie klienta Systém bude schopný získať poplatky, ktoré klient hradí, a ich čiastku vo formáte podľa daného poplatku a to z rôznych bankových systémov pre získanie kompletnej konfigurácie produktov podľa cenníkovej ponuky. Systém musí byť schopný vyhodnotiť, či sa jedná o individuálnu konfiguráciu každého poplatku na základe porovnania s cenníkovou cenou.

F5 – Vygenerovanie výpisu pre klienta Systém bude schopný vygenerovať výpis vo formáte XLSL v podobe vhodnej pre predanie klientovi. Výpis bude obsahovať základné informácie o klientovi a o jeho účtoch, a obdobie, pre ktorý je výpis vygenerovaný. Výpis bude obsahovať kompletné informácie o produtoch, ktoré klient využíva a poplatky, ktoré za ne hradí. Výpis bude umožňovať filtrovanie podľa názvu a identifikátora produktu.

F6 – Správa konfigurácie kliena Systém bude umožňovať založenie a správu žiadosti pre individuálne nastavenie poplatkovania pre konkrétneho klienta. Žiadosť bude pozostávať z krokov pre návrh, schvaľovanie a aplikovanie individuálneho nastavenia. Žiadosť bude možné delegovať inému bankérovi, zrušiť alebo zamietnuť.

F7 – Návrh nastavení Systém bude ponúkať rozhranie pre navrhovanie nových individuálnych nastavení pre klienta, v rámci ktorých bude možné zadať hodnotu vo formáte, ktorý vyžaduje daný poplatok. Systém bude kontrolovať obmedzenia kladené na daný poplatok a neumožní uložiť chybný návrh. Návrh je možné upraviť, odstrániť alebo vrátiť do hodnoty podľa cenníka. V rámci auditovateľnosti zmien bude po každom prechode v žiadosti alebo

po zmene používateľa vytvorená nová verzia návrhu na základe pôvodnej verzie, ktorú bude možné ďalej modifikovať.

F8 – Historické žiadosti Systém bude umožňovať náhľad do starých žiadostí, ktoré boli aplikované alebo zamietnuté. Systém bude zobrazovať cenník platný k dátumu ukončenia žiadosti a bude zobrazovať poplatky hradené klientom k rovnakému dátumu. Analogicky bude systém schopný zobraziť návrh nastavenia platnému ku konkrétnemu staršiemu stavu aktuálnej alebo ukončenej žiadosti.

F9 – Notifikovanie používateľov Systém bude schopný informovať používateľov v prípade, že sa žiadosť dostala do chybového stavu v rámci automatického spracovania žiadosti. Systém bude notifikovať bankéra, ktorý navrhoval zmeny v prípade nutnosti modifikácie alebo doplnenia informácii, a aplikačnú podporu v prípade technickej chyby. Systém bude taktiež notifikovať klienta v prípade zamietnutia alebo schválenia žiadosti.

F10 – Získanie spätnej väzby Systém umožní používateľom pomocou formulára odoslať spätnú väzbu v štandardnej pobobe celej banky s hodnotením na škále od 1 do 5 a nepovinným poľom pre poznámku.

F11 – Odporúčanie produktov Systém bude schopný poskytnúť odporúčania produktov a poplatkov za produkty pomocou najlepšej odporúčacej metódy na základe kapitoly 4.

1.3.2 Nefunkčné požiadavky

Nefunkčné požiadavky sú zamerané na charakteristiky a vlastnosti aplikácie, ktoré nesúvisia priamo s jej konkrétnymi funkciami, ale ovplyvňujú jej celkový výkon, bezpečnosť, spoľahlivosť a použiteľnosť. V tejto časti sú popísané hlavné ciele nefunkčných požiadaviek kladené na túto aplikáciu.

N1 – Doba odozvy pre získanie cenníka Systém načíta a zobrazí cenník produktov do maximálnej doby odozvy 0.5 sekundy od začatia požiadavky. Tento čas odozvy sa vzťahuje na 95% požiadaviek za normálnych prevádzkových podmienok.

N2 – Integrácia systému Systém bude prepoužívať existujúce komponenty a mikroslužby dostupné v bankovom prostredí vrátane integrácie nového typu procesu do procesného systému, využitia zdieľaných grafických komponentov a služieb pre získanie informácií o klientovi, poskytnutie spätnej väzby, atď.

N3 – Doba odozvy pre získanie klientských poplatkov Systém načíta a zobrazí klientské poplatky do maximálnej doby odozvy 2 sekundy od začatia požiadavky. Tento čas odozvy sa vzťahuje na 95% požiadaviek za normálnych prevádzkových podmienok. Systém bude prispôsobený pre časté operácie načítavania z databázy.

N4 – Úprava návrhu nastavenia Systém bude implementovať mechanizmus automatického ukladania pri úprave návrhu nastavenia. Zmeny vykonané používateľmi počas návrhu by sa mali automaticky a trvalo ukladať po každej zmene s maximálnym oneskorením 3 sekundy za bežnej prevádzky v 95% prípadov. Používatelia by mali zaznamenať minimálne prerušenie počas automatického ukladania a systém musí poskytovať vizuálne podnety alebo upozornenia, ktoré indikujú úspešné dokončenie operácie ukladania.

Kapitola 2

Úvod do odporúčacích systémov

Pre vytvorenie systému pre odporúčanie, je nutné mať pokrytý teoretický základ do danej problematiky. V tejto kapitole sú rozobrané základné aspekty odporúčacích systémov. Časť 2.1 analyzuje základné typy odporúčacích systémov a následne je v časti 2.1 podrobne analyzovaný typ kolaboratívneho filtrovania, ktorý je ďalej implementovaný v praktickej časti. Následne sú v časti 2.3 popísané metriky potrebné pre vyhodnotenie odporúčaní. Pre odporúčanie produktov je nutné taktiež získať a spracovať dáta, čomu sa venuje časť 2.4. Pre kvalitné odporúčenie produktov je nutné brať do úvahy problémy súvisiace s odporúčacími systémami, popísané v časti 2.5. Záver kapitoly 2.6 je venovaný existujúcim riešeniam pre odporúčanie produktov v bankovom sektore.

Systémy pre odporúčanie sú softvérové nástroje a techniky, ktoré umožňujú návrh údajov, ktoré by mohli byť užitočné pre užívateľa [11]. Dizajn takýchto odporúčacích systémov závisí na doméne a konkrétnych charakteristikách dostupných dát [12]. Ako príklad odporúčacích systémov môže slúžiť odporúčanie príspevkov na sociálnych sieťach, videí a hudby v streamovacích službách alebo odporúčenie produktov v e-shope na základe predchádzajúceho prehliadania, profilu užívateľa alebo jeho preferencií. Odporúčacie systémy sú primárne určené jednotlivcom, ktorým chýba dostatočná osobná skúsenosť alebo schopnosť vyhodnotiť potenciálne prevažujúci počet alternatívnych údajov, ktoré môže doména ponúknuť [13]. V najjednoduchšej podobe je výsledok personalizovaných odporúčaní zobrazený ako zotriedený list údajov [11].

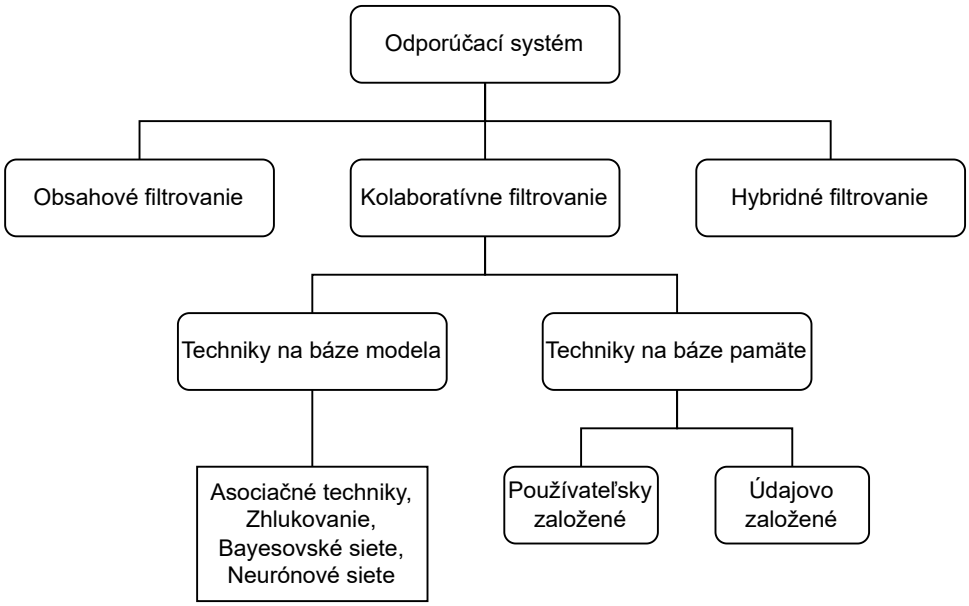
2.1 Typy odporúčacích systémov

Podľa doporučovacej techniky existuje niekoľko typov systémov. Medzi najznámejšie typy patrí obsahové filtrovanie (angl. Content-based filtering), kolaboratívne filtrovanie (angl. Collaborative filtering) a hybridné filtrovanie (angl. Hybrid filtering) [14]. Medzi ďalšie typy odporúčaní patrí napríklad znalostné filtrovanie (angl. Knowledge-based filtering), demografické filtrovanie alebo komunitné filtrovanie (angl. Community-based filtering) [11]. V tejto časti budú zanalizované základné typy a bude vybraný optimálny typ pre ďalšie použitie v práci.

Obsahové filtrovanie Pomocou daného typu sú odporúčané údaje, ktoré sú podobné tým, ktoré sa používateľovi páčili v minulosti. Podobnosť medzi údajmi sa vypočítava na základe charakteristík spojených s porovnávanými údajmi. Následne sú používateľovi poskytnuté údaje, ktoré sú najviac podobné pozitívne ohodnoteným údajom. Napríklad, ak používateľ pozitívne ohodnotil film patriaci do žánru komédie, systém sa môže naučiť odporúčať ďalšie filmy z tohto žánru. [11] Algoritmus je závislý na doméne a kladie dôraz na analýzu atribútov položiek s cieľom generovať predikcie [15].

Kolaboratívne filtrovanie Systémy fungujúce na princípe kolaboratívneho filtrovania zhromažďujú spätnú väzbu používateľov vo forme hodnotenia údajov v danej doméne a zisťujú podobnosti v spôsobe hodnotenia medzi rôznymi používateľmi, aby odporúčili údaj. Kolaboratívne filtrovanie je založené na predpoklade, že používatelia, ktorí sa zhodli v minulosti, sa zhodnú aj v budúcnosti a budú mať radi podobné druhy údajov, ako mali radi v minulosti. [16]. Algoritmy kolaboratívneho filtrovania sú rozdelené na báze pamäte (angl. Memory-based filtering) a na báze modelu (angl. Model-based filtering). Pamäťové algoritmy používajú celú sadu údajov na výpočet odporúčaní, a to buď pomocou porovnávania používateľsky založeného filtrovania (angl. User-based filtering), ktoré vyhľadáva podobnosti na základe porovnávania údajov medzi používateľmi, alebo filtrovania na báze údajov (angl. Item-based filtering), kde sa využíva podobnosť medzi jednotlivými položkami, namiesto používateľských podobností. Modelovo založené algoritmy namiesto použitia celej sady údajov generujú model, na základe ktorého sa vytvárajú odporúčania. [14]

Hybridné filtrovanie Technika hybridného filtrovania kombinuje rôzne odporúčacie techniky s cieľom dosiahnutia lepšej optimalizácie systému tým, že obchádza obmedzenia a problémy spojené s konkrétnym typom algoritmu [17]. Myšlienka hybridného filtrovania spočíva v tom, že kombinácia algoritmov poskytne presnejšie a efektívnejšie odporúčania než jeden algoritmus, pretože nevýhody jedného algoritmu môžu byť prekonané iným algoritmom. Základné stratégie kombinovania viacerých algoritmov je váženie, kde každému algoritmu sa určí váha odporúčania; prepínanie, kde sa využije konkrétny algoritmus na základe momentálnej potreby alebo skladanie, kde odporúčania jedného algoritmu prehodnocuje ďalší algoritmus. [15]



Obr. 2.1 Rozdelenie typov odporúčacích systémov [15]

Na základe povahy dát analyzovaných v kapitole 1 je možné usúdiť, že odporúčania je nutné generovať na základe užívateľských vlastností, a teda je najvhodnejšie použiť kolaboratívne filtrovanie, ktoré funguje na základe porovnávania rôznych užívateľov. Odporúčanie na základe obsahu je v tomto prípade nevhodné použiť, keďže nie je žiaduce klientovi poskytovať ďalšie zľavy na základe predchádzajúcich nastavení, ale cieľom je skôr nastaviť optimálne nastavenie produktov pre klienta a prípadne protivážiť niektoré výhody istými protivýhodami. V ďalších častiach práce sa teda pracuje s algoritmami kolaboratívneho filtrovania.

2.2 Kolaboratívne filtrovanie

Táto časť analyzuje detailne techniku kolaboratívneho filtrovania predstavenej v časti 2.1. Z techník založených na báze pamäte je popísaná technika používateľsky založeného filtrovania, ktorá je užitočná pre porovnávanie na základe podobnosti medzi užívateľmi a dokáže určiť vyvážené nastavenie pre každého používateľa. Naopak, údajovo založené filtrovanie hľadá poplatky podobné iným poplatkom a stráca sa kontext použitia poplatku u konkrétneho používateľa v spojení s inými jeho poplatkami, čo je v tomto prípade nežiaduce. Z techník na báze modela sú vybrané asociačné pravidlá, zhlukovanie a rozhodovacie stromy.

2.2.1 Používateľsky založené filtrovanie

Používateľsky založené filtrovanie s využitím celej dátovej sady uloženej v pamäti funguje na princípe hľadania podobností medzi užívateľmi, na základe ktorého sa vyselektujú používatelia s najvyššou zhodou a pomocou váženej kombinácie položiek, vytvorenej na základe rebríčka zhody používateľov, sa vytvorí zoznam odporúčaných údajov pre daného používateľa. Väčšinu prístupov je možné zgeneralizovať do nasledujúcich krokov:

1. Vytvorenie poradia užívateľov U na základe podobnosti s užívateľom, pre ktorého sa odporúčajú údaje. Pobobnosť sa počíta pomocou jednej z podobnostných funkcií (angl. similarity functions).
2. Výber k používateľov s najvyššou mierou podobnosti k používateľovi, pre ktorého sa odporúčajú údaje (taktiež nazývaní susedia).
3. Výpočet odporúčaných údajov na základe váženej kombinácie údajov susedov. Váha je vytvorená na základe výpočtu podobnostnej funkcie v kroku 1.[12]

Nasledujúca časť sa venuje podrobnej analýze podobnostných metód pre výpočet v kroku 1 a následne je rozoberaný výpočet odporúčaných údajov, potrebný pre krok 3.

2.2.1.1 Podobnostné funkcie

Jedným z najdôležitejších návrhových rozhodnutí pri vytváraní používateľsky založeného kolaboratívneho filtrovania je výber podobnostnej funkcie. Výber najvhodnejšej podobnostnej funkcie závisí na konkrétnej doméne a jej použití. V tejto časti sú popísané najznámejšie podobnostné funkcie – kosínusová podobnosť, pearsonova korelácia a euklidovská vzdialenosť.

Kosínusová podobnosť Technika zobrazuje používateľa ako vektor hodnotení údajov medzi používateľom a vektormi ostatných používateľov, na základe ktorých sa generuje odporúčanie. Kosínus medzi dvoma vektormi reprezentujúcich dvoch používateľov indikuje podobnosť používateľov medzi sebou. Hodnoty blížiace sa k 1 indikujú silnú koreláciu medzi dvoma údajmi, hodnoty blížiace sa k 0 neindikujú žiadnu koreláciu. [18] Rozdiel tejto techniky oproti pearsonovej korelácii je, že neberie do úvahy priemerné hodnotenie rôznych používateľov [19]. Toto správanie sa vo väčšine prípadov môže javiť ako negatívne. Podľa [20] je vhodné použiť túto techniku v prípade, že dáta sú riedke, čo znamená, že používatelia majú mnoho údajov neohodnotených. Kosínusovú podobnosť medzi používateľmi u a u' je možné vyjadriť podľa nasledujúceho vzorca:

$$s(u,u') = cos\measuredangle(r_u, r_{u'}) = \frac{r_u \cdot r_{u'}}{\|r_u\| \times \|r_{u'}\|} = \frac{\sum_{i \in I} r_{u,i} \cdot r_{u',i}}{\sqrt{\sum_{i \in I} r_{u,i}^2} \cdot \sqrt{\sum_{i \in I} r_{u',i}^2}} \tag{2.1}$$

kde I je množina všetkých údajov, ktoré je možné hodnotiť a $r_{u,i}$ je hodnotenie údaju i používateľom u. [12, 19]

Pearsonova korelácia Zatiaľ čo kosínusová podobnosť je závislá na vektorovom priestore založenom na lineárnej algebre, pearsonová korelácia využíva štatistický prístup, kde je meraná miera, v akej dve premenné spolu lineárne súvisia [21]. Empirické výskumy podľa [16] ukázali, že pearsonova korelácia všeobecne produkuje lepšie výsledky ako kosínusová podobnosť. Korelácia vracia, hodnotí v rozmedzí od -1 do 1, kde hodnoty blížiace sa k 1 indikujú silnú pozitívnu koreláciu, zatiaľ čo hodnoty blížiace sa k -1 indikujú silnú negatívnu koreláciu. Podobnosť medzi používateľmi u a u' je možné vyjadriť pomocou nasledujúceho vzorca:

$$s(u,u') = \frac{\sum_{i \in I_{uu'}} (r_{ui} - \bar{r_u}) \cdot (r_{u'i} - \bar{r_{u'}})}{\sqrt{\sum_{i \in I_{uu'}} (r_{ui} - \bar{r_u})^2} \cdot \sqrt{\sum_{i \in I_{uu'}} (r_{u'i} - \bar{r_{u'}})^2}} \tag{2.2}$$

kde $I_{uu'}$ vyjadruje všetky údaje hodnotené oboma používateľmi, r_{ui} vyjadruje hodnotenie údaju i používateľom u a $\bar{r_u}$ vyjadruje priemerné hodnotenie položiek používateľa u. Pearsonova korelácia taktiež eliminuje rozdiely vzniknuté na základe rôznych hodnotiacich škál používateľov pomocou zohľadnenia priemerného ohodnotenia používateľa. [18] Podľa [20] je túto techniku vhodné použiť v prípade, že dáta sú predmetom zaujatosti alebo rôznych hodnotiacich škál používateľov.

Euklidovská vzdialenosť Za najjednoduchšiu a najviac používanú mieru vzdialenosti je považovaná Euklidovská vzdialenosť. Menšie euklidovské vzdialenosti naznačujú väčšiu podobnosť, zatiaľ čo väčšie vzdialenosti znamenajú väčšiu odlišnosť medzi používateľmi alebo položkami. Táto vzdialenosť je ale citlivá na škálu hodnotení rôznych používateľov, je menej efektívna pri výpočte údajov s mnoho neohodnotenými údajmi a v mnohorozmerných vektorových priestoroch má tendenciu stať sa viac uniformnejšou, čím sa stáva menej zmysluplnou. [22, 23] Podľa [20] je túto vzdialenosť vhodné použiť v prípade, že dáta nie sú príliš riedke a je potrebné zachytiť vysoký rozsah hodnôt údajov. Euklidovskú vzdialenosť pre používateľov u a u' je možné vyjadriť pomocou vzorca:

$$s(u,u') = d(r_u, r_{u'}) = \sqrt{\sum_{i \in I} (r_{ui} - r_{u'i})^2} \tag{2.3}$$

kde I je množina všetkých údajov, ktoré je možné hodnotiť a $r_{u,i}$ je hodnotenie údaju i používateľom u.

Na základe dát analyzovaných v kapitole 1 je možné usúdiť, že ako vstup do podobnostných funkcií musia byť zahrnuté taktiež údaje zahrňujúce finančný objem klientov, aby bolo možné odporučiť ceny aj na základe veľkosti klienta (napr. odlíšenie korporátov od malých firiem). To sa odlišuje od typickej škály hodnotení, ako je to v iných doménach, kde normalizácia škál hodnotení vylepšuje výpočet a žiadne údaje tým nie sú stratené. V tejto doméne je teda žiaduce zohľadniť rôzne škály niektorých údajov, aby bolo možné používateľov porovnávať presnejšie. Okrem pearsonovej korelácie, ktorá všeobecne produkuje lepšie výsledky, je žiaduce pri konkrétnych údajoch použiť taktiež kosínusovú podobnosť alebo euklidovskú vzdialenosť. Odporúčací systém sa teda stáva hybridným a cieľom praktickej časti práce je taktiež nájsť optimálnu kombináciu váh a údajov pre konkrétne podobnostné funkcie tak, aby systém produkoval čo najlepšie odporúčania.

2.2.1.2 Výpočet odporúčaných údajov

Po vypočítaní susedov $N \subseteq U$ pre používateľa u je vytvoriť odporúčanie pre údaj i pomocou váženého priemeru hodnotení susedov, najčastejšie pomocou nasledujúceho vzorca:

$$p_{u,i} = \bar{r}_u + \frac{\sum_{u' \in N} s(u,u') \cdot (r_{u',i} - \bar{r}_{u'})}{\sum_{u' \in N} |s(u,u')|} \tag{2.4}$$

$s(u, u')$ je váha ohodnotenia vytvorená pomocou podobnostnej funkcie a $\bar{r}_u$ je priemerné hodnotenie údajov používateľa u. Odčítanie priemerného hodnotenia susedných užívateľov $\bar{r}_{u'}$ vo vzorci 2.4 napomáha kompenzovať rozdiely v používateľských hodnotiacich škálach, keďže niektorí užívatelia majú tendenciu dávať vyššie hodnotenia ako iní používatelia. [24]

	Údaj 1	Údaj 2	Údaj 3	Údaj 4
Používateľ A	4	?	3	5
Používateľ B	?	5	4	?
Používateľ C	5	4	2	?
Používateľ D	2	4	?	3
Používateľ E	3	4	5	?

Tabuľka 2.1 Príklad na výpočet odporúčaných údajov podľa [24]. Údaje majú hodnotenia na škále od 1 do 5. Používa sa pearsonovská podobnostná funkcia, susedstvo s veľkosťou 2 a výpočet odporúčacích údajov pomocou vzorca 2.4. Pri výpočte odporúčania údaja 4 pre používateľa C sú dostupné ohodnotenia jedine od používateľov A a D, a teda $s(C, A) = 0.832$ a $s(C, D) = -0.515$. Platí teda:

$$\begin{aligned} p_{C,4} &= \bar{r}_C + \frac{s(C,A) \cdot (r_{A,4} - \bar{r}_A) + s(C,D) \cdot (r_{D,4} - \bar{r}_D)}{|s(C,A)| + |s(C,D)|} \\ &= 3.667 + \frac{0.832 \cdot (5-4) + -0.515 \cdot (2-3)}{0.832 + 0.515} \\ &= 4.667 \end{aligned}$$

2.2.2 Techniky na báze modela

Techniky na báze modela dokážu rýchlo odporučiť množinu údajov na základe vopred vypočítaného modelu a produkujú výsledky podobné pamäťovo založeným technikám. Modelovo založené techniky analyzujú maticu pozostávajúcu z dát o používateľoch a ich hodnoteniach na identifikáciu závislosti medzi údajmi. Tieto techniky taktiež riešia problém riedkosti údajov asociovaný s odporúčacími systémami. Medzi najznámejšie techniky patria asociačné pravidlá, zhlukovanie alebo rozhodovacie stromy. [21]

2.2.2.1 Asociačné pravidlá

Získavanie asociačných pravidiel pozostáva z hľadania pravidiel, ktoré predpokladajú výskyt údaju na základe výskytoch iných údajov v jednej transakcii. Transakciou sa môže uvažovať napríklad množina pozitívne ohodnotených produktov užívateľa. Asociačné pravidlo je výrok v tvare $X \implies Y$, kde X a Y sú množiny údajov. Asociačné pravidlá majú tzv. *podporu* a *spoľahlivosť* (angl. *support* a *confidence*). Tieto vlastnosti je možné pre množinu údajov X a Y vyjadriť formálne:

$$podpora = \frac{počet\ transakcií\ obsahujúce\ X \cup Y}{celkový\ počet\ transakcií} \tag{2.5}$$

$$spoľahlivosť = \frac{počet\ transakcií\ obsahujúce\ X \cup Y}{počet\ transakcií\ obsahujúce\ X} \tag{2.6}$$

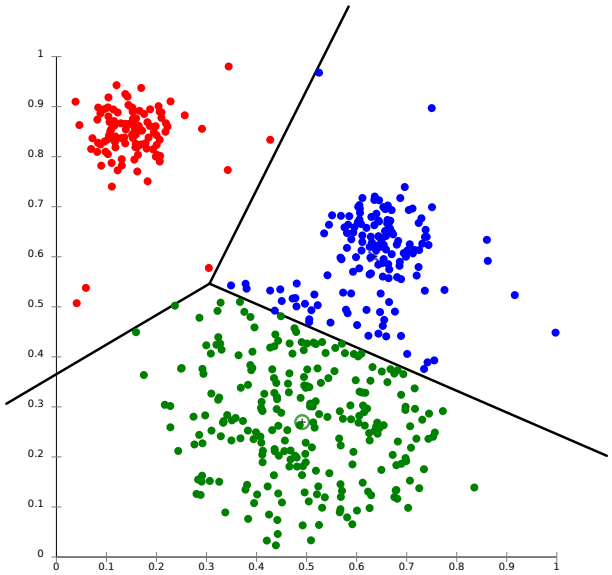
Cieľom algoritmu je nájsť všetky asociačné pravidlá, ktoré majú podporu a spoľahlivosť vyššiu ako vopred stanovené minimá. Následne je nutné zotriediť údaje podľa spoľahlivosti pravidiel, ktoré dané údaje navrhovali a následne vrátiť prvých N údajov. Okrem toho existujú techniky na zredukovanie počtu transakcií a porovnávaní, ako je napríklad Apriori algoritmus. [11, 25]

	Údaj 1	Údaj 2	Údaj 3	Údaj 4	Údaj 5
Používateľ A	1	0	0	0	?
Používateľ B	1	0	0	1	1
Používateľ C	1	0	1	0	1
Používateľ D	0	0	0	1	1
Používateľ E	0	1	1	0	0

Tabuľka 2.2 Príklad na výpočet asociačného pravidla podľa [25]. Otázka znie, či je vhodné odporúčiť údaj 5 používateľovi A. Na základe údajov je možné určiť, že pravidlo s najvyššou spoľahlivosťou je predpoklad, že používateľovi, ktorému sa páčil údaj 1, sa bude páčiť údaj 5, a teda používateľovi A je vhodné odporúčiť tento údaj. Dané pravidlo má podporu 50% transakcií a spoľahlivosť je 100%.

2.2.2.2 Zhlukovanie

Táto technika poskytuje odlišný prístup k organizácii dát. Zhlukovanie je algoritmus založený na učení sa bez dozoru, čo znamená, že nepotrebuje žiadne trénovacie dáta. Cieľom algoritmu je na základe neoznačenej sady dát vytvorenie niekoľkých skupín (zhlukov) dát. [26] Po tom, čo boli zhluky sformované, je možné vytvoriť odporúčanie pre používateľa na základe priemernej hodnoty údaja v zhluku priradenom danému používateľovi [21]. Snahou algoritmu je vytvorenie čo najmenších možných vzdialeností medzi údajmi v jednom zhluku, zatiaľ čo vzdialenosť medzi údajmi z rôznych zhlukov by mala byť čo najväčšia. Ideálny zhlukovací algoritmus by mal brať do úvahy všetky možné varianty zhlukov a vybrať z nich najoptimálnejší variant, čo je NP-ťažký problém. [11] Ďalej je detailnejšie analyzovaná metóda k-means.



Obr. 2.2 Zhlukovanie pomocou algoritmu k-means na gaussovsky rozloženej dátovej sade [27]

Metóda zhlukovania k-means rozdeľuje množinu dát veľkosti N do k disjunktných podmnožín C_j obsahujúce N_j údajov, ktoré sú k sebe podľa danej vzdialenostnej funkcie čo najbližšie. Každý zhluk je definovaný N_j členmi a jedným centroidom λ_j. Centroid je bod, ktorého súčet vzdiale-

ností od ostatných členov zhluku je minimalizovaný. Cieľom algoritmu je iteratívnym procesom zminimalizovať vzdialenosť vyjadrenú nasledujúcim vzorcom:

$$E = \sum_{1}^{k} \sum_{n \in C_j} dist(x_n, \lambda_j) \tag{2.7}$$

kde x_n je vektor reprezentujúci n-tý údaj, λ_j je centroid zhluku C_j a *dist* je vzdialenostná funkcia. Vzdialenostná funkcia je v praktickej časti vybraná analogickým spôsobom ako pri pamäťovo založených metódach (pozri 2.2.1.1). V každej iterácii prebieha výber nového centroidu pre zohľadnenie členov, ktoré boli v zhluku pridané a odobrané. Iterácia prebieha, až kým žiadny prvok nebude schopný zmeniť priradený zhluk, t.j. hodnota E nemôže byť ďalej minimalizovaná, respektíve kým sa nedosiahne nejaká podmienka na ukončenie pre vylepšenie efektívnosti. Napriek tomu, že tento algoritmus je jednoduchý a efektívny, má isté obmedzenia. Algoritmus predpokladá predchádzajúcu znalosť dát, aby bolo možné zvoliť vhodnú hodnotu k, vzor výsledných zhlukov je veľmi citlivý na výbere počiatočných centroidov a algoritmus môže produkovať prázdne zhluky. [11, 26]

```
function KMeansCluster(x_1, ..., x_n, k)
    A[1], ..., A[N] ← initial cluster assignment
    repeat
        change ← false
        for i = 1 to N do
            k̂ ← arg min_k dist(x_i, λ_k)
            if A[i] ≠ k̂ then
                A[i] ← k̂
                change ← true
            end if
        end for
    until change ≠ false
    return A[1], ..., A[N]
end function
```

Výpis kódu 2.1 Zhlukovanie pomocou algoritmu k-means [26]

2.2.2.3 Rozhodovacie stromy

Rozhodovacie stromy sú klasifikátory zamerané na cieľový atribút v podobe stromovej štruktúry. Údaje na klasifikáciu sú zložené z atribútov (angl. features) a ich cieľovej hodnoty. Uzly rozhodovacieho stromu môžu byť *rozhodovacie uzly*, tvoriace vnútorné uzly stromu, kde každý uzol predstavuje jeden atribút, ktorého hodnota určí vetvu podstromu, ktorý bude ďalej aplikovaný. Listy indikujú hodnotu cieľového atribútu. [11]

Jeden z mnoho algoritmov pre rozhodovacie stromy je algoritmus ID3[1], ktorý využíva princíp hladného algoritmu (angl. greedy search) zhora nadol. Podstata ID3 algoritmu je v každom kroku vybrať najlepší atribút pre rozdelenie stromu pomocou informačného zisku (vzájomná informácia, angl. information gain). [28] Informačný zisk atribútu a pre množinu údajov S je možné definovať ako:

$$IG(S, a) = H(S) - H(S|a) \tag{2.8}$$

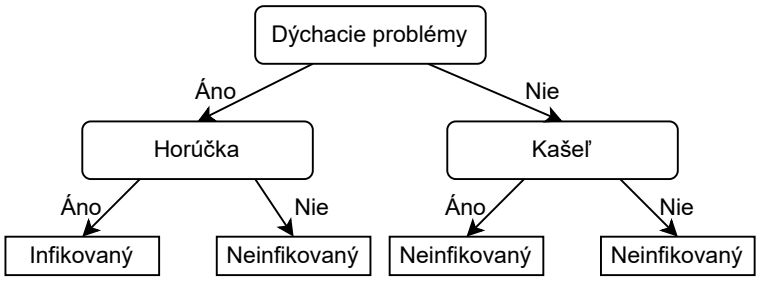
[1]Iterative Dichotomiser 3, t.j. iteratívna dychotómia, čo znamená, že algoritmus opakovane rozdeľuje atribút do dvoch alebo viacerých skupín v každom kroku

kde $H(S)$ je entrópia cieľového atribútu množiny S a $H(S|a)$ je podmienená entrópia pri množine hodnôt atribútu a:

$$H(S|a) = \sum_{\substack{v \in \text{hodnoty} \\ \text{atribútu } a}} \frac{|S_a(v)|}{|S|} \cdot H(S_a(v)) \tag{2.9}$$

$$H(S) = -\sum_{\substack{v \in \text{hodnoty} \\ \text{cieľového} \\ \text{atribútu } S}} p(v) \cdot \log_2 p(v) \tag{2.10}$$

kde $S_a(v)$ je podmnožina údajov S, ktorých atribút a má hodnotu v a $p(v)$ je pravdepodobnosť výskytu hodnoty v v hodnotách cieľového atribútu množiny údajov S. Cieľom je teda v každej iterácii nájsť atribút s najvyšším informačným ziskom.



Obr. 2.3 Ukážka rozhodovacieho stromu na sade dát o infekčnosti pacientov a príznakov podľa [28]. Cieľový atribút je teda infekčnosť pacienta a rozhodovacie atribúty sú pacientove symptómy. Hodnoty každého atribútu sú binárne (áno/nie). Na príklade je možné pozorovať nevýhodu algoritmu ID3 v podobe opakujúcich sa hodnôt listov v pravom podstrome, kde uzol s daným symptómom je redundantný a nepriniesol žiadnu hodnotu.

Výhodou rozhodovacích stromov je jednoduchosť na zostrojenie, rýchlosť a produkujú množinu pravidiel jednoduchých na vysvetlenie, čím sa stávajú transparentnými oproti iným technikám. Na druhej strane je presnosť systémov založených na rozhodovacích stromoch nižšia ako pri odporúčaniach založených na priemernom hodnotení. Rozhodovacie stromy fungujú najlepšie s nižším počtom atribútov a často sú kombinované s inou technikou pre zvýšenie jej presnosti, ako napríklad s asociačnými pravidlami, kde rozhodovací strom vyfiltruje potenciálnych užívateľov pre odporúčenie. [11, 25]

2.3 Metriky pre vyhodnotenie odporúčaní

Pre správne vyhodnotenie odporúčaných údajov je nutné si vybrať vhodné metriky pre konkrétnu doménu a systém. Medzi dôležité aspekty domény analyzovanej v kapitole 1 patrí statické množstvo údajov na odporúčanie, t.j. banka ponúka konkrétny počet produktov, a variabilitu hodnotiacich škál, čo znamená, že každý produkt môže mať rôznu výšku poplatkov. Rozdiel v tomto odporúčacom systéme je aj to, že sú zaujímavé aj negatívne odporúčania, t.j. produkty, ktoré ostávajú nastavené klientovi za cenníkovú cenu, prípadne je cena oproti cenníku ešte nevýhodnejšia na protiváženie výhodnej ponuky za iný produkt. Naďalej však ostávajú z hľadiska odporúčaní najdôležitejšie tie produkty, ktorých cena bude zľavnená.

Podľa [25] existuje niekoľko typov metrík ako presnosť predikcií, presnosť klasifikácií, presnosť poradia alebo pokrytie. Na základe typu domény a vybraných odporúčacích systémov bude ďalej pracované s metrikami na základe presnosti predikcií a presnosti klasifikácií.

2.3.1 Presnosť predikcií

Medzi najznámejšiu metriku pre vyhodnotenie správneho predpovedania používateľských preferencií pre konkrétny údaj patrí priemerná absolútna odchýlka (angl. Mean Absolute Error, MAE):

$$MAE = \sqrt{\frac{1}{|\tau|} \cdot \sum_{(u,i)\in\tau} |\hat{r}_{ui} - r_{ui}|} \tag{2.11}$$

kde τ je množina dvojíc predpovedaných údajov pre používateľov, $\hat{r}_{ui}$ je odporúčaná hodnota a r_{ui} je známa reálna hodnota údaja. Okrem toho existuje metrika normalizovanej verzie priemernej absolútnej odchýlky, ktorá berie do úvahy škálu odporúčacích hodnôt:

$$NMAE = \frac{MAE}{r_{max} - r_{min}} \tag{2.12}$$

kde r_{max} a r_{min} sú najvyššie a najnižšie hodnotenia údajov, čím sa hodnoty $NMAE$ normalizujú do intervalu $\langle 0, 1\rangle$. [25, 29]

2.3.2 Presnosť klasifikácií

V rámci vyhodnotenia odporúčaní je potrebné vedieť správnosť určenia zliav pre produkty pre klienta. K tomu je vhodné využiť metriku na presnosť klasifikácií, ktorá dokáže určiť, či systém správne klasifikuje produkty. K tejto klasifikácii je potrebné dáta rozdeliť do skupín správne odporúčaných produktov (angl. True-Positive, TP), nesprávne odporúčaných produktov (angl. False-Positive, FP), správne neodporúčaných produktov (angl. False-Positive, FP) a nesprávne neodporúčaných produktov (angl. False-Negative, FN).

	Odporúčané	Nedporúčané
Použité	Správne odporúčané (TP)	Nesprávne neodporúčané (FN)
Nepoužité	Nesprávne odporúčané (FP)	Správne neodporúčané (TN)

Tabuľka 2.3 Klasifikácia možného výsledku odporúčania položky používateľovi [11]

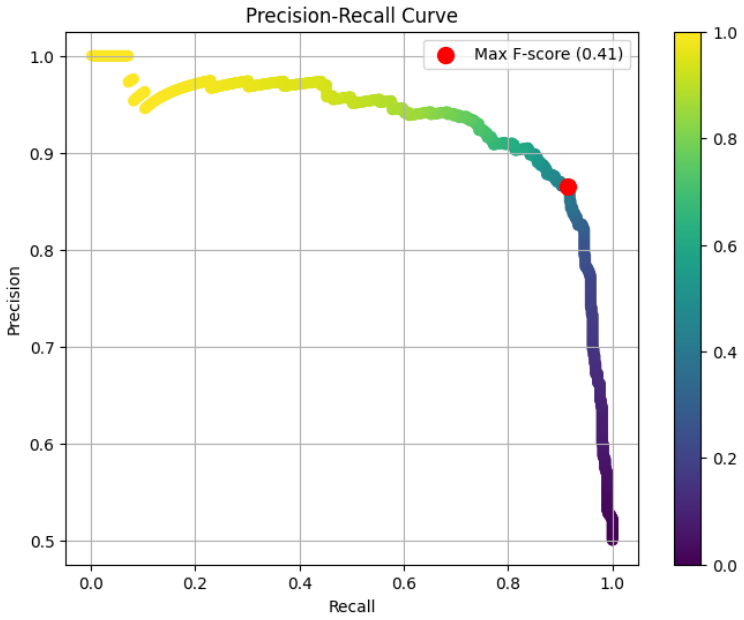
Na základe uvedeného rozdelenia je následne možné zhodnotiť metriky založené na presnosti a na úplnosti (angl. Recall) odporúčaných údajov:

$$Presnosť = \frac{|TP|}{|TP \cup FP|} \tag{2.13}$$

$$Úplnosť = \frac{|TP|}{|TP \cup FN|} \tag{2.14}$$

Presnosť predstavuje počet správnych predikcií z celkového počtu predikcií, zatiaľ čo úplnosť predstavuje počet správnych predikcií z celkového možného počtu správnych odporúčaní. Zvyčajne je možné pozorovať výmenu úspešnosti presnosti a úplnosti so zvyšujúcim sa počtom údajov. Typicky sa úplnosť zvyšuje so zvyšujúcim sa počtom odporúčaných údajov, keďže pravdepodobnosť nájdenia žiaducich údajov v testovacej sade údajov stúpa, ale na úkor presnosti. Z toho dôvodu vznikla metrika *F-measure*, ktorá dokáže vyhodnocovať systém viac nezávislo na konkrétnej aplikácii: [25, 11]

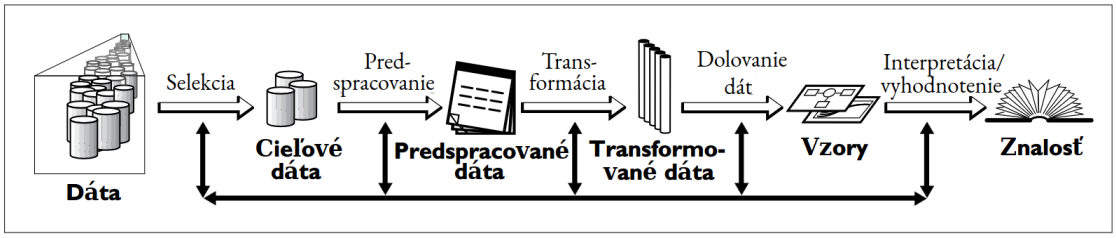
$$F\text{-}measure = \frac{2 \cdot Presnosť \cdot Úplnosť}{Presnosť + Úplnosť} \tag{2.15}$$



Obr. 2.4 Krivka presnosti a úplnosti so zaznačenou optimálnou hodnotou F-measure [30]. So zvyšujúcou úplnosťou sa typicky stráca presnosť.

2.4 Získavanie dát pre odporúčanie

V predchádzjúcich častiach je popísané, ako je možné získavať a ohodnocovať odporúčania na základe poskytnutých dát. Hoci táto aktivita je primárnym cieľom tejto práce, pre získanie kvalitných dát je v rámci procesu odporúčania venovať adekvátnu pozornosť získavaniu dát, ich príprave a spracovaniu. Slepá aplikácia metód pre dolovanie dát je nebezpečná činnosť vedúca k odhaleniu nezmyselných vzorcov [31]. Táto časť sa venuje procesu získavania dát z databázy a krokom vedúcim k úspešnému odporúčaniu údajov.



Obr. 2.5 Prehľad krokov k získavaniu znalostí z databáz [31]

Podľa [31] sa proces získavania znalostí z databáz môže definovať ako netriviálny proces identifikácie platných, nových, potenciálne užitočných a konečne pochopiteľných vzorov v dátach. Tento proces je znázornený na obrázku 2.5. Proces je interaktívny a iteratívny a môže byť zhrnutý do nasledujúcich krokov:

1. Naštudovanie aplikačnej domény, vrátane relevantných predchádzajúcich doménových znalostí a cieľov aplikácie. Tento krok je vykonaný v rámci kapitoly 1.
2. Vytvorenie cieľovej množiny dát, zahŕňujúce výber dátovej sady alebo podmnožiny údajov alebo vzoriek, na základe ktorých bude prebiehať zisťovanie.

3. Čistenie a predspracovanie dát, ako odstránenie nežiaducich dát alebo odľahlých hodnôt; zber potrebných informácii pre návrh alebo zváženie nežiaducich údajov; určenie stratégií pre zvládnutie problémov súvisiacimi s chýbajúcimi údajmi; branie do úvahy informácie podliehajúce časovým zmenám a známe zmeny v dátach; riešenie problémov súvisiacich s databázovým systémom, ako mapovanie dátových typov a schém vrátane chýbajúcich hodnôt a nezmámych typov.

4. Redukcia dát a projekcia. Tento krok zahŕňa nájdenie užitočných vlastností na reprezentáciu údajov závisiacich na cieli a použitie redukcie rozmerov alebo tranformačných metód na zníženie počtu použitých premenných alebo nájdenie invariantu na reprezentáciu dát. Podľa [11] je veľký počet riedkych údajov problém pre odporúčacie systémy, ktoré sú závislé na porovnávacích funkciách (pozri 2.2.1.1), ako sú techniky kolaboratívneho filtrovania založené na báze pamäte alebo napríklad zhlukovanie. Medzi najznámejšie techniky na zníženie dimenzionality patrí analýza hlavných komponentov (angl. Principal component analysis, PCA) alebo singulárny rozklad (angl. Singular Value Decomposition) založený na princípe faktorizácii matíc.

5. Zvolenie účelu dolovania dát a zvolenie algoritmu pre dolovanie. Tieto kroky sú podrobne vykonané v predchádzajúcich častiach kapitoly.

6. Dolovanie dát, zahrňujúce hľadanie žiaducich vzorov v konktétnej reprezentačnej forme.

7. Interpretácia vrátane vysvetlenia nájdených vzorov a možnosťou vrátiť sa späť do ktoréhokoľvek predchádzajúceho kroku, vizualicácia extrahovaných vzorov, odstránenie redundandných alebo irelevantných vzorov a pretlmočenie užitočných vzorov používateľom.

8. Použitie objavených znalostí, zahŕňajúce začlenenie znalostí do produkčných systémov, podniknutie ďalších krokov na základe nových znalostí, zdokumentovanie a zreportovanie zaujímavých vzorov.

Zatiaľ čo v tejto časti sú zanalyzované znalosti potrebné pre odporúčanie bankových produktov, v praktickej časti je vykonaná aplikácia jednej iterácie týchto krokov v danej doméne pre problém odporúčania údajov vrátane extrakcie a transformácie dát z databázového systému, predspracovanie údajov a samotná exekúcia a vyhodnotenie.

2.5 Problémy odporúčacích systémov

Pri navrhovaní odporúčacích systémov sa typicky objavujú problémy, ktoré môžu znížiť použiteľnosť a výkonnosť algoritmov odporúčania. Táto časť sa venuje najčastejším problémom, ktoré môžu nastať pri navrhovaní tohto systému podľa [32].

Riedkosť dát Tento problém už v práci je zmienený v súvislosti s vzdialenostnými funkciami a algoritmami, ktoré ich používajú. Zvyčajne je ponuka údajov pre odporúčanie príliš vysoká a prekryv údajov medzi dvoma používateľmi je príliš nízky alebo žiadny. Taktiež, aj keď je priemerný počet hodnotení na používateľa vysoký, hodnotenia sú rozdelené na údaje veľmi nerovnomerne a mnoho údajov môže mať len niekoľko hodnotení (jedná sa o tzv. zákon nepriamej úmernosti, angl. Power Law).

Rozmanitosť vs. presnosť Pri snahe odporúčiť spoľahlivý údaj konkrétnemu užívateľovi je typicky najefektívnejšie použiť údaje, ktoré sú populárne a sú vysoko hodnotené. Takéto údaje majú ale veľmi nízku pridanú hodnotu, pretože sú jednoduché na nájdenie. Dobrý zoznam odporúčaných údajov by mal preto obsahovať údaje, ktoré majú nízku pravdepodobnosť nájdenia samotným používateľom. Možné riešenie tohto problému je priame vylepšenie zoznamu odporúčaných údajov o diverzifikované položky a použitie hybridných odporúčacích systémov.

Čas Zatiaľ čo väčšina používateľov má záujmy výrazne líšiace sa na základe časového obdobia, mnoho odporúčacích systémov faktor času zanedbávajú. V bankovom priemysle sa tento fakt môže prejaviť napríklad obdobím recesie a inflácie, kedy sa priority zákazníkov môžu zmeniť. Je teda žiaduce pri odporúčaní brať ohľad na novšie vytvorené záznamy, ktoré môžu zmenu požiadaviek zákazníkov indikovať.

Vyhodnotenie odporúčaní Zatiaľ čo existuje mnoho metrík pre vyhodnotenie zoznamu odporúčaných údajov, otázkou stále zostáva ako zvoliť najlepšiu metriku pre danú situáciu. Porovnávanie rôznych odporúčacích algoritmov je problematické, keďže rôzne algoritmy môžu riešiť rôzne úlohy. Celkovú spokojnosť užívateľov je náročné merať na základe dát a používateľské výskumy sú žiaducim zdrojom spätnej väzby.

Používateľské rozhranie Bolo preukázané, že pre zvýšenie akceptácie odporúčaných údajov je dôležitá transparentnosť údajov. Používatelia oceňujú, keď je zjavné, prečo im bol údaj odporúčený. Na druhej strane, musí byť rozsiahly zoznam odporúčaní prezentovaný v jednoduchej podobe.

Okrem vyššie zmienených problémov existujú vo všeobecnosti aj niektoré ďalšie známe problémy odporúčacích systémov, ktoré sú v tomto prípade irelevantné. Napríklad problém chladného štartu (angl. Cold Start), kde je nedostatočný počet informácií pri nových zákazníkoch, nemôže nastať, keďže banka povinne zbiera informácie o každom klientovi napríklad pomocou tzv. AML opatrení. Problém so škálovateľnosťou dát nie je problém typický v danej doméne, keďže počet firemných klientov je relatívne stabilný a nízky oproti iným doménam. Odporúčací systém pre bankovú aplikáciu je taktiež odolný voči útokom snažiacich sa o manipuláciu dát a odporúčaní, keďže sa jedná o interný systém, kde údaje zadávajú jedine pracovníci banky.

2.6 Odporúčacie systémy v bankovníctve

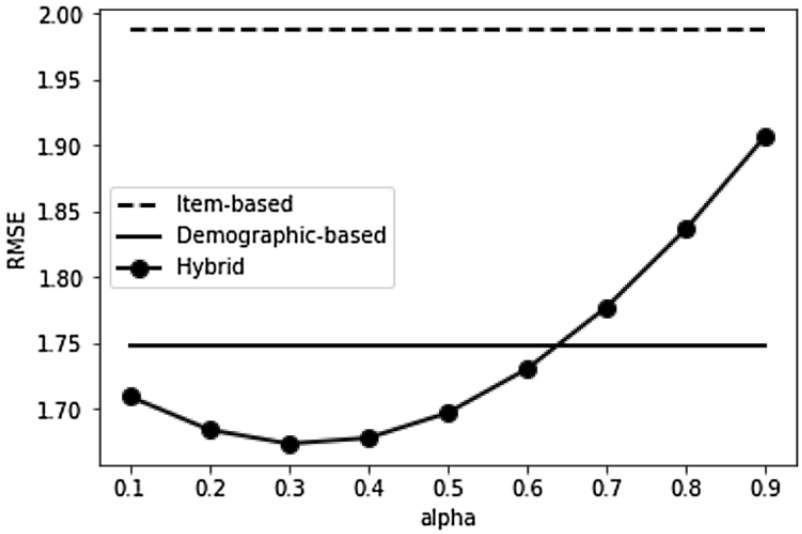
Táto časť je venovaná analýze existujúcich riešení pre problém odporúčania produktov v bankovníctve a vo finančnom sektore. Na základe vybraných textov sú popísané zvolené riešenia autorov a sú vystihnuté najdôležitejšie aspekty, ktoré je možné využiť v ďalších častiach práce.

Autori v [33] demonštrovali užitočnosť odporúčacích systémov v bankovom sektore. Bola analyzovaná dátová sada európskej banky obsahujúca investície 200 000 zákazníkov, 1 200 000 transakcií za 12 mesiacov a rôzne typy produktov. Všetky dáta boli zahešované a anonymizované a tabuľka transakcií obsahovala dátum vykonania, užívateľské dáta ako identifikátory klienta, pobočky a účtu, a dáta o produkte ako typ imania, mena transakcie, priemyselný sektor a skupina, hodnotenie a obchodný kanál). Autori pracovali bez informácie o výške transakcie alebo o výške majetku klienta a vytvorili model pre odporúčania na základe binárnej informácie, či bol alebo nebol daný produkt zakúpený. Autori pracovali v 300-dimenzionálnom vektorovom priestore a vyskúšali 3 algoritmy kolaboratívneho filtrovania. Pre testovanie boli použité techniky odstránenia jednej náhodnej transakcie zo zakúpených produktov pre každého používateľa (angl. leave-one-out), odstránenie poslednej transakcie pre každého používateľa (angl. leave-last-out) a odstránenie náhodne 20% transakcií. Pre zlepšenie kvality odporúčania sa autori pokúšali odstrániť n najpopulárnejších produktov z testovacej sady dát. Pre $n = 0$ zistili, že systém funguje spoľahlivo, zatiaľ čo pre $n = 50$ funguje systém nespoľahlivo, a teda je vhodné odporúčať populárne produkty, ktoré ale nesúvisia so záujmami daného používateľa.

V texte [34] sa autori zamerali na použitie odporúčacích systémov vo finančnom sektore vrátane bankovníctva, akciových trhov alebo poistenia. Odporúčacie systémy v tomto odvetví pomáhajú zákazníkovi v rozhodovaní pri výbere najlepšieho produktu alebo bankérovi pri rozhodovaní, či poskytne zákazníkovi daný produkt. Bolo zistené, že pri aplikovaní algoritmov môžu byť použité okrem samotných hodnotení produktov aj metadáta o používateľovi a samotných produktoch, ako napríklad vek a pohlavie používateľov alebo typ a cena produktov. Pri odporúčaní akcií na burze pomáhajú odporúčacie systémy používateľovi rozhodnúť sa, či je vhodné zachovať

si alebo predať akcie. Tieto odporúčania sú vykonané na základe analyzovania daných akcií a ich potencionality, čo znamená, že tieto systémy sú závislé na vonkajších okolnostiach a musia byť schopné zohľadniť tieto vplyvy. Napríklad, autori [35] vytvorili odporúčací systém založený na kolaboratívnom filtrovaní, ktorý skúmal závislosť medzi pocitmi investorov na základe textov publikovaných v platforme pre investorov Guba a medzi trendmi na akciových trhoch v Číne. Na záver autori diskutujú výhody a nevýhody odporúčacích techník, kde kolaboratívne filtrovanie založené na báze pamäte má výhody v podobe jednoduchosti pridania nového produktu, ale má problémy s riedkosťou dát. Na druhej strane, kolaboratívne filtrovanie na báze modela je podľa autorov užitočná technika pre zvládnutie riedkosti dát, ale je výpočetne náročná.

V článku [36] je kladený dôraz na použitie hybridného odporúčacieho systému pre predaj produktov v bankovom prostredí. Autori používajú vážený hybridný odporúčací systém rozdelený na demografické a transakčné dáta. Ako demografické dáta boli použité anonymizované číslo zákazníka, vek, pohlavie, rodinný stav a profesia. Pre nájdenie podobnosti medzi zákazníkmi bol použitý zhlukovací algoritmus k-means, kde textové dáta boli prevedené do numerickej podoby a počet zhlukov bol určený pomocou tzv. lakťovej metódy[2]. Medzi transakčné dáta boli zaradené anonymizované údaje o čísle účtu, čísle zákazníka, čísle produktu, typu transakcie, množstve transakcie, dátume transakcie, status transakcie a menu transakcie. Pre porovnanie týchto dát bola použitá technika kolaboratívneho filtrovania založená na porovnávaní produktov. Ako ideálna váha hybridného odporúčacieho systému bola vyhodnotená 0.3 pre údajovo založené kolaboratívne filtrovanie a 0.7 pre zhlukovanie používateľov na základe demografických údajov. Ako ideálna hodnota počtu odporúčaní bola zvolená 4, hoci hodnota sa môže individuálne líšiť na základe počtu dostupných produktov. Podľa autorov tento hybridný princíp dokáže prekonať problém chladného štartu a na základe demografických údajov dokáže vygenerovať odporúčania aj pre nových zákazníkov.



Obr. 2.6 Ukážka nastavovania váh pre hybridný odporúčací systém. Najnižšia chybovosť je dosiahnuná pri váhe 0.3 pre údajovo založené kolaboratívne filtrovanie a 0.7 pre zhlukovanie používateľov na základe demografických údajov. [36]

Na základe analyzovaných riešení je možné v ďalšej časti práce využiť poznatky pre implementáciu odporúčacích systémov v bankovej oblasti. V riešeniach bolo popísané, ako je možné

[2]angl. Elbow Method, je heuristická metóda založená na výpočte rozptylu medzi každým zhlukom pre rôzne hodnoty k algoritmu k-means a nájdenia bodu (elbow), kde pridanie nového zhluku výrazne neznižuje rozptyl. Táto hodnota je považovaná za odporúčaný počet zhlukov pre algoritmus.

rozdeliť množinu testovacích dát, ako je možné transformovať dáta, ako vhodne vypočítať parametre pre odporúčací systém, potrebu metadát a demografických údajov pri výpočte podobnosti medzi užívateľmi a vyriešenia problému chladného štartu, a dôležitosť hybridizácie riešenia pre dosiahnutie lepších výsledkov.

Kapitola 3
Architektúra mikroslužieb

Táto kapitola je venovaná analýze architektúry mikroslužieb. Prvá časť je venovaná podmienkam, ktoré je nutné splniť predtým, ako je možné naimplementovať túto architektúru. Ďalej sú popísané charakteristiky tejto architektúry a na záver sú spomenuté najčastejšie vzory a antivzory spojené s touto architektúrou.

Architektúra mikroslužieb bola navrhnutá s cieľom rozdeliť architekrútu aplikácií na nezávislo nasaditeľné služby, ktoré je možné rýchlo nasadiť na ľubovoľný infraštruktúrny zdroj podľa potreby. Mikroslužby sú nezávisle nasaditeľné, zvyčajne podporované nástrojom na nasadzovanie a orchestráciu, napríklad v cloude, čo im umožňuje časté a nezávislé nasadzovanie podľa potreby, obzvlášť ak majú byť poskytované ako PaaS[1]. [37]

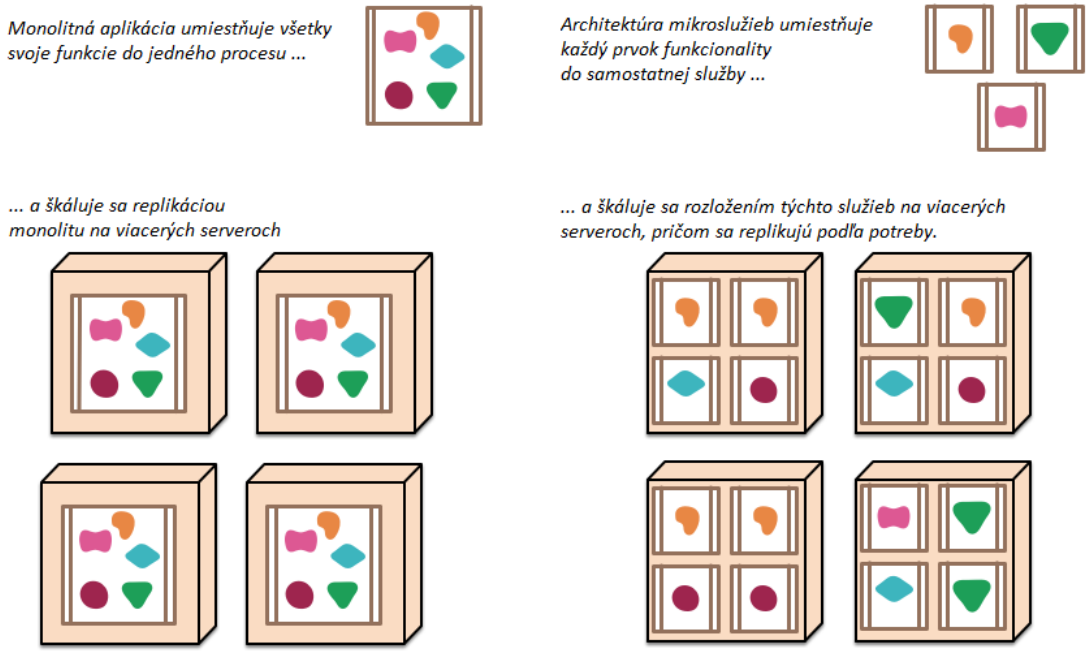
Zatiaľ čo je monolitická architekrúra vzor, kde sú všetky komponenty a moduly súčasťou jednej aplikácie, úzko prepojené a vzájomne závislé a formované do jednej nerozdeliteľnej jednotky, architektúra mikroslužieb uľahčuje údržbu, použiteľnosť, škálovateľnosť, dostupnosť a automatické nasadzovanie aplikácie. Hlavnými výhodami architektúry mikroslužieb sú technologická heterogénnosť, kde každá mikroslužba používa rôzne technológie pre dosiahnutie žiaduceho cieľa a výkonu; odolnosť voči výpadku celého systému, kde výpadok jednej mikroslužby neovplyvní ostatné; škálovateľnosť oproti monolitickej aplikácii je dostupnejšia vďaka možnosti konfigurácie jednej služby namiesto celého systému; uľahčené nasadzovanie bez ovplyvnenia celého systému alebo minimalizovanie počtu ľudí pracujúcich na konkrétnej časti kódu, čím je uľahčená organizácia tímu. [38]

3.1 Prerekvizity pre implementáciu mikroslužieb

K tomu, aby bolo možné začať používať architektúru mikroslužieb, je potrebné, aby systém spĺňal niektoré požiadavky, aby bolo možné zabezpečiť jeho riadny chod a údržbu. [40] V prvom rade je potrebné zabezpečiť kontinuálnu integráciu a kontinuálne nasadenie (angl. Continuous Integration, Continuous Deployment) pomocou zreťazeného spracovania (angl. pipeline). Tento krok zabezpečí automatické zostavovanie aplikácie, spustenie automatizovaných testov, kontrolu kvality kódu a skontrolovanie zraniteľnosti kódu. Následne je zostavený Docker obraz, ktorý je automaticky nahraný do registra Docker obrazov, z ktorého je aplikácia následne automaticky nasadená na konkrétne prostredie.

Keďže sa z podstaty architektúry nachádzajú rôzne funkcionality v odlišných kontajneroch, ktorých inštancie môžu kedykoľvek vznikať a zanikať, je dôležité udržiavať logy aplikácií na cen-

[1] Platforma ako služba (PaaS) je cloudový vývojový model, ktorý poskytuje kompletnú softvérovú platformu a infraštruktúru pre vývoj, testovanie a nasadzovanie aplikácií bez potreby správy hardvéru alebo operačného systému.



Obr. 3.1 Porovnanie monolitickej architerkúry a architerkúry mikroslužieb podľa [39]

tralizovanom mieste. To môže byť zabezpečené pomocou distribuovanej streamovacej platformy, ktorá odošle logy jednotlivých inšancií do analytického úložiska pre správu logov, akým je napríklad nástroj Elastic. Okrem toho je podstatné okrem samotného zberu logov, monitoring aplikácie a varovania, bez ktorých nie je možné identifikovať príčiny chýb. Ďalej je potrebná schopnosť dohľadať chyby k ich základu, keďže podstata problému sa môže nachádzať v kombinácii spolupráce viacero mikroslužieb.

Ďalšou podmienkou pre možnosť implementácie architektúry je schopnosť rozdeliť rozhranie na interné a externé pomocou tzv. API brány. Táto brána zabezpečí jediný vstupný bod pre volanie služby a efektívnu a spoľahlivú komunikáciu medzi službami a externými zdrojmi. Ďalšou možnosťou je vytvorenie separátneho API pre každý typ klienta (angl. Backend for Frontend, BFF).

3.2 Charakteristika architektúry

V tejto časti sú prebrané základné charakteristiky architektúry mikroslužieb. Keďže neexistuje jednotná definícia tejto architektúry, v tejto časti sú popísané vybrané princípy podľa [39].

Komponentizácia prostredníctvom služieb V softvérovom priemysle je snahou dlhodobo budovať systémy pomocou spájania komponentov, ako je to v reálnom svete. Komponent je možné definovať ako jednotku softvéru, ktorú je možné nezávisle nahradiť a povýšiť. Hoci architektúra mikroslužieb používa knižnice, jej cieľom pre vytvorenie softvérových komponentov je pomocou rozčlenenia na samostatné služby. Zatiaľ čo komponentizácia prostredníctvom knižníc použitých v programe sú volané pomocou funkcií uložených v pamäti, komponentizácia prostredníctvom služieb využíva nezávislé komponenty komunikujúce pomocou mechanizmov ako sú napríklad webové služby.

Jednou z hlavných výhod tohto prístupu je možnosť nezávislého nasadzovania služieb. Oproti monolitickej aplikácii, kde pri zmene jednej funkcionality dochádza k prenasadeniu celej apli-

kácii, pri dekompovaných mikroslužbách je nutné prenasadiť jedine služby, kde došlo k zmenám. Ďalšou výhodou toh to prístupu je explicitnejšie aplikačné rozhranie. Zatiaľ čo väčšina programovacích jazykov nemá dobrý mechanizmus pre definovanie explicitného rozhrania, dokumentácia a disciplína sú jediné prostriedky pre zachovanie zapuzdrenia komponentov. Mikroslužby pomáhajú zabraňovať tomuto problému pomocou explicitného mechanizmu pre volanie ostatných komponentov. Na druhej strane, tento prístup je nákladnejší oproti volaniu funkcie uloženej v pamäti programu. Okrem toho je zmena priradenia zodpovednosti medzi komponentmi náročnejsí proces kvôli potrebe presunu funkcionalít.

Zameranie sa na obchodné schopnosti Pri rozdeľovaní veľkej aplikácie na menšie časti sa manažment väčšinou zameriava na technické vrstvy aplikácie, čo vedie k rozdeleniu na tímy zamerané na používateľské rozhranie, server a databázu. Pri takomto rozdelení môže aj jednoduchá zmena viesť k predĺženiu času dodávky projektu a schvaľovania rozpočtu. Na druhej strane je možné zamerať sa na logiku aplikácie bez ohľadu na technologickú vrstvu aplikácie. Toto je príklad aplikácie Conwayovho zákona, ktorý poukazuje na to, že každá organizácia, ktorá navrhuje systém, vytvorí návrh, ktorého štruktúra je kópiou komunikačnej štruktúry organizácie.

Architektúra mikroskužieb sa zameriava na rozdelenie služieb podľa obchodných schopností. Takéto služby zahŕňajú širokú škálu implementácie softvéru pre danú obchodnú oblasť, vrátane používateľského rozhrania, servera, databázy a podobne. V dôsledku toho majú tímy viacero úloh a potrebujú ovládať viacero zručností potrebných na vývoj od používateľskej skúsenosti po databázu a riadenie projektov.

Zameranie sa na produkt Zvyčajne je vývoj pri snahe vyvinúť aplikáciu vnímaný ako projektový model, kde je cieľom dodať nejaký softvér, ktorý sa potom považuje za dokončený. Po dokončení je softvér upravovaný iba pre účely údržby a spravovania, a projektový tím, ktorý ho vytvoril, je rozpustený. Pri aplikácii architektúry mikroslužieb je tendencia vyhýbať sa tomuto modelu a namiesto toho je uprednostnená predstava, že tím by mal byť vlastníkom projektu počas celej doby trvania projektu. Vďaka tomu sú vývojári v dennom kontakte s tým ako sa ich softvér správa v produkčnom prostredí, čo napomáha k zvýšeniu kontaktu medzi používateľmi aplikácie, keďže musia prevziať aspoň časť podpory.

Takéto zameranie na produkt je prepojené so zameraním sa na obchodné schopnosti, kde namiesto toho, aby sa na softvér hľadelo ako na sadu funkcionalít, ktoré je potrebné dokončiť, je tu trvalá spojitosť s otázkou, ako môže softvér pomôcť svojim používateľom zlepšiť obchodné možnosti.

Chytré koncové body a jednoduché spracovanie Pri budovaní komunikačnej štruktúry medzi rôznymi procesmi existuje množstvo riešení a prístupov, ktoré kladú dôraz na samotný komunikačný mechanizmus, akým je napríklad riešenie Enterprise Service Bus, ktoré zvyčajne zahŕňajú sofistikované metódy na smerovanie správ, transformácie a aplikáciu obchodných pravidiel. Pri aplikácii architektúry mikroslužieb je kladený dôraz na čo najvyššiu oddelenosť a kohéznosť služieb. Každá služba má svoju doménovú logiku a správajú sa viac ako filtre podobné tým v Unixových operačných systémoch, kde je prijatá požiadavka, aplikovaná vhodná logika a vrátená odpoveď. Preto je pre takéto spracovanie vhodný jednoduchý protokol ako napríklad REST namiesto iných komplexných riešení. Okrem toho je druhým riešením využitie odosielania správ pomocou jednoduchej zbernice správ (angl. message bus), akým je napríklad implementácia RabbitMQ, ktorej cieľom je poskytnutie jednoduché a spoľahlivé riešenie pre správu asynchrónnych správ.

Decentralizované riadenie Jedným z následkov centralizovanej správy je tendencia štandardizovania technológií na jedinú platformu. Tento prístup je nevyhovujúci, keďže na každý problém je vhodný iný prístup. Rozdelenie monolitickej aplikácie prináša možnosť voľby platformy pre budovanie nového systému. Napríklad, pre vygenerovanie jednoduchej stránky je

možné použiť technológiu Node.js alebo pre vybudovanie komponentu schopného reagovať v krátkom čase je možné použiť jazyk C++. Ďalej je možné využiť napríklad rôzne databázové systémy, ktoré najviac vyhovujú pre vyriešenie daného problému.

Okrem toho možné je využitie rôznych prístupov a štandardov pri riešení podobných problémov. Namiesto striktne daných štandardov pre implementáciu je možné využiť zdieľané zdrojové kódy v rámci organizácie alebo zverejnené ako open-source. Zdieľaním týchto spoločných riešení sú ostatní vývojári motivovaní vyriešiť podobný problém podobným spôsobom s možnosťou využitia odlišného prístupu na vyriešenie konkrétnych problémov. Zdieľané knižnice zvyčajne riešia typické problémy ukladania dát, komunikáciu v rámci interných procesov alebo automatizáciu infraštruktúry.

Decentralizovaná správa dát Na najabstraktnejšej úrovni si je možné decentralizovanú správu dát predstaviť tak, že konceptuálny model sveta sa medzi rôznymi systémami líši. Tento problém nastáva pri integrácii naprieč veľkými systémami, kde sa napríklad pohľad zákazníka môže líšiť od pohľadu aplikačnej podpory, čím môže vznikať rozdielnosť dostupných atribútov alebo ich sémantika sa môže líšiť. Tento problém môže vznikať aj vnútri aplikácie, ktorá je rozdelená na viacero komponentov. Pre tento účel môže slúžiť doménovo riadený návrh (angl. Domain-Driven Design), ktorý oddeľuje komplexnú doménu do viacerých ohraničených kontextov a mapuje väzby medzi nimi.

Okrem decentralizácie na úrovni konceptuálneho modela, architektúra mikroslužieb taktiež decentralizuje rozhodnutia ohľadom ukladania dát. Zatiaľ čo monolitické aplikácie uprednostňujú jediný logický databázový celok pre ukladanie dát a podnikové riešenia uprednostňujú jediný databázový systém naprieč rôznymi aplikáciami kvôli licenčným dôvodom, architektúra mikroslužieb ponecháva každej službe správu dát na databáze danej služby. Databázy jednotlivých služieb sa môžu líšiť inštanciou, databázovou technológiou alebo úplne rozdielnym databázovým systémom.

Ošetrenie chýb Dôsledok využívania služieb ako komponenty je, že aplikácie musia byť navrnhuté tak, aby boli schopné tolerovať zlyhanie služieb. Ľubovoľná služba môže zlyhať v dôsledku nedostupnosti jej závislosti a klient musí byť schopný odpovedať čo najprívetivejšie. Keďže môžu služby kedykoľvek zlyhať, je dôležité detekovať zlyhania rýchlo a ak to je možné, automaticky obnoviť službu.

Evolučný návrh Evolučný návrh je praktika vyvíjania systému prirodzeným spôsobom pridávaním minimálneho množstva kódu pre vyhovenie obchodných potrieb iteratívnym a postupným prístupom, nepretržitou zmenou štruktúry kódu pre optimalizáciu zmien, čím je umožnená konštantná rýchlosť vývoja v dlhších časových úsekoch [41]. Kedykoľvek pri rozdeľovaní systému do komponentov nastáva problém rozhodovania, akým spôsobom rozdeliť aplikáciu. Kľúčovými vlastnosťami komponentov sú možnosť nezávislej výmeny a možnosť povýšenia komponentu, čo znamená, že sú v kóde hľadané také miesta pre prepísanie komponentu, aby boli pri zmene čo najmenej obmedzení ostatní spolupracovníci.

Dôraz na možnosť nezávislej výmeny komponentu je špeciálnym prípadom princípu modulárneho návrhu, ktorého cieľom je vytvárať modulárnosť na základe pozmeniteľnosti – zachovať časti, ktoré sa menia v rovnaký čas v rovnakom module. Časti systému, ktoré sa menia zriedka, by mali byť v iných službách ako tie, ktoré v čase podstupujú veľa zmien. Ak sa opakovane menia navzájom dve služby, mali by byť zlúčené.

3.3 Architektonické vzory

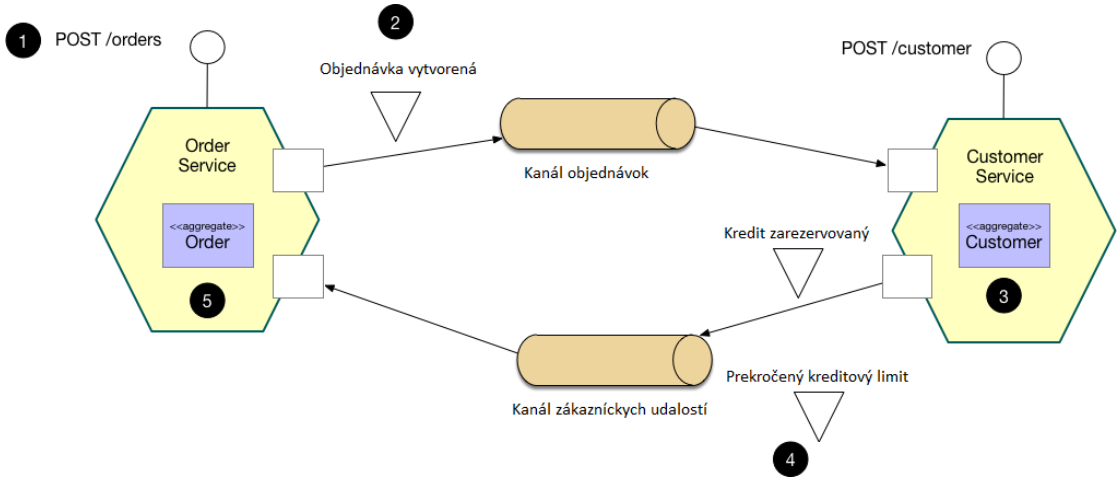
V tejto časti tú prebrané najdôležitejšie vzory súvisiace s architektúrou mikrosužieb. V tejto časti sú prebrané vzory súvisiace najmä s dátovou manipuláciou. V predchádzajúcich častiach už sú

spomenuté niektoré vzory ako samostatná databáza pre každú službu alebo API brána. Na záver sú prebrané najčastejšie antivzory vznikajúte pri navrhovaní mikroslužieb.

3.3.1 Saga

Pri aplikovaní architektúry mikroslužieb má typicky každá služba vlastnú databázu. Niektoré biznisové transakcie sa ale rozpínajú medzi viacerými službami. Pre zachovanie ACID vlastností biznisovej transakcie naprieč viacerými službami slúži vzor sága.

Sága je sekvencia lokálnych transakcií, po ktorých je aktualizovaná lokálna databáza a je vyslaná správa, ktorá spustí ďalšiu lokálnu transakciu v inej službe. V prípade, že je porušené biznisové pravidlo, sága vykoná sériu kompenzačných operácii, pomocou ktorých navráti stav databáz do pôvodného stavu pred vykonaním lokálnych transakcií. [42]



Obr. 3.2 Príklad použitia vzoru Saga podľa [42]. Objednávka je vytvorená v `Order Service`, na základe ktorej je potrebné zákazníkovi zarezervovať kredit za objednávku v `Customer Service`. V prípade, že zákadník nemá dostatočný kredit, musí byť objednávka zrušená.

3.3.2 Event Sourcing

Typické operácie služby sú úprava databázových záznamov a odosielanie správ sprostredkovateľovi správ. Databázové záznamy sú upravené atomicky a správy ostatným službám o zmene sú odoslané pre zabránenie dátových nekonzistencií a chýb. Pre potvrdenie prijatia správ službami nie je žiaduce použiť 2PC[2], ale bez toho nie je garantované, že správa bola doručená alebo že systém nezlyhal pred odoslaním správy.

Riešením tohto problému je vzor Event Sourcing, ktorý ukladá stav biznisovej entity ako sekvenciu udalostí v podobe meniacich sa stavov. V prípade, že sa stav biznisovej entity zmení, je vložená nová udalosť do zoznamu udalostí. Keďže sa jedná o ukladanie záznamu do úložiska udalostí, jedná sa o atomickú operáciu. Aplikácia môže následne obnoviť aktuálny stav entity pomocou zopakovania všetkých udalostí z úložiska. Toto úložisko sa chová taktiež ako sprostredkovateľ správ, ktorý doručuje správy všetkým odoberateľom v prípade zmeny stavu entity. [43]

[2]Two-phase Commit Protocol je distribuovaný protokol pre koordináciu transakcií medzi viacero databázovými systémami, ktorý zaisťuje buď spoločné prevedenie alebo návrat do pôvodného stavu v prípade zlyhania

3.3.3 CQRS

Výsedkom architektonického návrhu architekrúty mikroslužieb a použitia vzoru samostatnej databázy pre každú službu je, že spojenie dát z rôznych služieb už nie je priamočiare. Taktiež, pri použití vzoru ako Event Sourcing, údaje už nie sú naďalej jednoducho získateľné. Riešením tohto problému je využitie vzoru pre oddelenie zodpovednosti dopytovacích príkazov (angl. Command Query Responsibility Segregation), na základe ktorého je vytvorený databázový pohľad, ktorý slúži jedine na čítanie dát. Aplikácia udržiava aktuálny stav repliky odoberaním doménových udalostí vyprodukované službou, ktorá je vlastníkom dát. [44]

3.3.4 Circuit Breaker

Služby musia pri spracovaní požiadavky niekedy spolupracovať. V prípade, že jedna služba synchrónne volá druhú službu, je tu vždy možnosť, že daná služba je nedostupná alebo prejavuje takú vysokú latenciu, že je nepoužiteľná. V takomto prípade sú v pôvodnej službe zaberané zdroje akými sú napríklad vlákna, zatiaľ čo sa čaká na odpoveď, čo môže viesť k vyčerpaniu zdrojov a zapríčineniu nemožnosti spracovať ďalšie požiadavky. Zlyhanie jednej služby teda môže kaskádovo vyvolať zlyhanie ostatných služieb naprieč aplikáciou.

Riešením tohto problému je volanie služby pomocou proxy, ktorá v prípade, že počet po sebe idúcich zlyhaní presiahne istú úroveň, sa aktivuje tzv. istič, ktorý po istú časovú dobu okamžite preruší volanie cieľovej služby a vráti chybový stav. Po uplynutí časového limitu je umožnený prechod obmedzenému počtu požiadaviek a v prípade úspechu je obnovený normálny chod aplikácie. [45]

3.3.5 Antivzory

Okrem vzorov popísaných vyššie existujú aj vzory, ktorým je žiaduce sa vyvarovať. Táto časť popisuje najzraniteľnejšie chyby pri návrhu architektúry podľa [46]. Ako najzraniteľnejšia chyba pri implementácii služby je zakódovanie koncových bodov. Služby, ktoré majú zakódované koncové body, čelia problémom v prípade, že sa zmení adresa. Riešením tohto problému je zavedenie prístupu zisťovania služieb (angl. Service Discovery). Ďalším najzraniteľnejším problémom je tzv. chybný zárez, ktorý spočíva v nesprávnej separácii záujmov, čo vedie k zvýšenej komplexite pri rozdeľovaní dát. Správnym riešením je mať jasne zanalyzované biznisové procesy a zdroje a rozdeliť služby na základe biznisových záujmov. Ďalším závažným problémom je cyklická závislosť medzi službami. Tá spôsobuje závažné problémy pri údržbe systému a prepoužiteľnosti. Pre odstránenie tohto problému je žiaduce zanalyzovať cykly na základe ich tvaru a aplikácie a prípadne použiť vzor API brány (pozri časť 3.1). Každé aplikačné rozhranie by malo byť sémanticky verzované, aby boli ostatné služby oboznámené s tým, či používajú novú alebo starú verziu API, a či je potrebné adaptovať sa na novú verziu API. V prípade chýbajúceho verzovania môžu konzumenti čeliť výzvam ako sú nekonzistencia dát alebo zmena významu. Posledným antivzorom uvedeným v tomto texte je zdieľané úložisko. V prípade, že viacero služieb využíva rovnakú databázu, dochádza k vysokej väzbe a znižuje nezávislosť tímu a služieb. Možnými riešeniami tohto problému sú použitie nezávislých databáz pre každú službu, jedna zdieľaná databáza s privátnymi tabuľkami alebo privátna databázová schéma pre každú službu.

Kapitola 4

Návrh a implementácia odporúčacieho algoritmu

Táto kapitola je venovaná výberu vhodného odporúčacieho algoritmu pre výber produktov v bankovom prostredí. Prvá časť je venovaná získavaniu dát, ich filtrovaniu a transformovaniu do potrebnej podoby. Ďalej je navrhnutý model pre meranie a vyhodnotenie úspešnosti jednotlivých algoritmov a sú vykonané samotné experimenty. Na záver je vyhodnotená celková úspešnosť algoritmov a sú uvedené odporúčania pre ďalšie kroky do budúcnosti pre rozšírenie a vylepšenie odporúčaní.

4.1 Príprava dát

Po tom, čo bola naštudovaná aplikačná doména, je nutné pred začatím hľadania vzorov v dátovej sade nutné predspracovať a vyčistiť dáta, aby bolo možné vykonať čo najpresnejšie meranie. Táto časť opisuje, ako je možné získať dáta z analytického úložiska a dáta o produktoch a ich poplatkoch. Následne sú z daných údajov odfiltrované nežiaduce výsledky a potom sú dáta upravené do podoby potrebnej pre odporúčacie algoritmy.

4.1.1 Získanie dát

Pre vytvorenie odporúčaní je potrebné získať dáta o produktoch a poplatkoch, nachádzajúce sa v samostatnom dátovom zdroji a údaje o klientoch, ktoré sa nachádzajú v analytickom úložisku. Táto časť je venovaná získavaniu a zlučovaniu týchto dát do podoby použiteľnej pre odporúčacie algoritmy.

Databázový model o produktoch a poplatkoch klienta, ktorých doménový model je na obrázku 1.1, musí byť navrhnutý vo vysoko normalizovanom stave pre účely zachovania synchronicity a aktuálnosti dát. Z toho dôvodu musí byť každý typ poplatku umiestnený vo vlastnej relácii, kde každá n-tica reprezentuje jeden poplatok pre jedného klienta v konkrétnom časovom úseku.

Pre účely odporúčania je tento spôsob reprezentácie dát nevhodný, keďže väčšina odporúčacích algoritmov očakáva ako vstup vektor číselných hodnôt. Z databázy je teda nutné získať reprezentáciu poplatkov v podobe atribútov jednej relácie namiesto niekoľkých n-tíc. Túto podobu je možné dosiahnúť pomocou SQL funkcií `MAX` a `DECODE` pri agregácii na základe čísla účtu. Výsledok tejto operácie sú n-tice skladajúce sa z čísla účtu a poplatkov spojenými s daným účtom, kde každý poplatok je reprezentovaný práve jedným atribútom.

```
SELECT /*+ PARALLEL(4)*/
  a.account_number AS account_number,
  MAX(DECODE(
    pmmf.fee_id,
    'FEE_ID1',
    CONCAT(pmmf.percent, '/', pmmf.min_amount, '/', pmmf.max_amount)
  )) AS "PMM_ID1"
  -- todo add max-decode for each fee individually
FROM percent_min_max_fee pmmf
JOIN account a ON pmmf.account_id = a.account_id
GROUP BY a.account_number;
```

Výpis kódu 4.1 Ukážka získavania poplatkov z databázy

Problém s týmto prístupom je v prípade, že poplatok sa skladá z viacerých parametrov, ako napríklad percentuálny poplatok s ohraničením (pozri časť 1.1.1). V takomto prípade budú dáta o danom poplatku uložené v podobe textového atribútu, kde jednotlivé údaje o poplatku budú oddelené konkrétnym oddeľovačom. Ďalším problémom s týmto prístupom je výkonnostná náročnosť týchto operácií. Keďže táto SQL operácia vytvára nové stĺpce na základe prehľadávania všetkých atribútov pôvodnej tabuľky, kde pre tieto atribúty nie sú vytvorené takmer žiadne indexy, jedná sa o vysoko nákladnú operáciu. Exekúcia pri paralelnom výpočte s použitím 4. vlákien trvá rádovo niekoľko desiatok minút pre každý typ poplatku. Je teda žiaduce vykonávať tieto operácie v čase, kedy databázový server nie je vyťažený ostatnými operáciami.

Analytické úložisko, z ktorého sú získavané dáta o klientoch a účtoch, je prispôsobené pre analytické operácie a žiadne väčšie komplikácie pri správnom využití indexov neboli zistené. Z dôvodu róbustného modelu a vysokého počtu záznamov je však nutné správne sformulovať SQL príkaz pre efektívne využitie zdrojov. Každý získaný záznam obsahuje číslo účtu pre možnosť zlúčenia dát s informáciami o poplatkoch klienta na účte.

Problém nastáva pri zlučovaní získaných údajov, keďže sa obe používané schémy nachádzajú na dvoch rôznych databázových serveroch. Tento problém je pre Oracle databázové systémy možné riešiť napríklad pomocou tzv. databázového linku, kde je možné vytvoriť spojenie do druhej databázy pomocou príkazu `CREATE DATABASE LINK`. Na túto operáciu je však nutné vlastniť špeciálne privilégiá a v rámci organizácie nie je žiaduce prepájať viacero databáz z hľadiska zachovania princípu nízkej spojitosti a vysokej súdržnosti dát. Riešenie pre zlúčenie týchto dvoch zdrojov bolo zvolené na aplikačnej úrovni. Obe dátové zdroje boli načítané do knižnice Pandas v jazyku Python, kde bolo následne vykonané zlúčenie na základe čísla účtu. Vzhľadom k veľkosti záznamov v rádoch desiatok tisíc záznamov (pozri časť 1.1.3) bolo toto riešenie umožnené a všetky údaje bolo možné vložiť do operačnej pamäte.

Výsledná dátová štruktúra danej knižnice teda obsahuje všetky dáta vrátane informácií o klientoch a poplatkoch. V oboch databázových schémach boli vykonávané spájania viacerých relácii s väzbou 1 ku N. Toto spájanie sa teda nutne musí prejaviť vo forme duplicity niektorých dát. Každá n-tica typicky obsahuje údaje spojenými s konkrétnym účtom vrátane atribútov viazaných na klienta. Typicky sa môže jednať napríklad o poplatok za používanie elektronického bankovníctva, ktoré je nezávislé od počtu účtov, alebo o demografické údaje viazané na klienta, ako je napríklad odvetvie. Pri implementácii odporúčacích algoritmov je nutné brať ohľad na túto duplicitu a pri odporúčaní produktov pre klienta je potrebné vynechať ostatné n-tice spojené s daným klientom. Pri implementácii vzdialenostných funkcií je treba brať do úvahy, že klienti so zvýšeným počtom účtov môžu ovplyvniť výledok tým, že duplicitné údaje budú mať zvýšenú váhu pri výpočte. Túto nerovnováhu je nutné pri aplikovaní funkcií odstrániť.

4.1.2 Filtrovanie

Pre dosiahnutie lepších výsledkov pre odporúčanie je dôležitá súčasť prípravy dátovej sady filtrovania údajov. Filtrovaním je možné odstrániť nežiaduce údaje, ktoré mohli vzniknúť technickými problémami alebo ľudskou chybou, odstrániť údaje, ktoré nie sú potrebné pre daný prípad použitia alebo znížiť výpočetnú záťaž.

Prvá aplikácia filtrovania dát bola zameraná na odstránenie nekompletných údajov alebo údajov, s ktorými nie je možné ďalej pracovať pre dôvody odporúčania. Každá firma vrátane bánk musí ponúkať svoje produkty za podmienky, aby bola naďalej v zisku. Z toho dôvodu je nutné, aby náklady na každého zákazníka, respektíve na produkty, ktoré používa, boli nižšie, ako sú výnosy. Pre každého klienta teda boli spočítané náklady a klienti, ktorí mali vyššie náklady ako výnosy, boli odfiltrovaní. Táto informácia každopádne neznamená, že klient je pre banku nezaujímavý alebo neziskový, ale na základe dostupných údajov to nie je možné určiť – klient môže využívať ďalšie produkty, ktoré nie sú súčasťou dátovej sady, ale je stále ziskový, hoci teda nie je možné ho pre účely odporúčania brať do úvahy. Ďalším dôvodom, prečo klient nemusí byť pre banku ziskový, môže byť na základe základných ekonomických princípov dôvod, že firma sa snaží vykompenzovať fixné náklady na produkt, ktorý je stratový. Informácie o takýchto produktoch nie sú riešiteľovi dostupné a takéto výnimky je nutné riešiť osobne. Systém teda musí vždy odporúčať údaje jedine na základe dát, o ktorých je známe, že sú profitabilné.

Ďalšia aplikácia filtrácie bola zameraná na produkty a poplatky, ktoré nie sú zaujímavé pre účely odporúčania. Jedná sa najmä o produkty, ktorým nie je možné nastaviť individuálne podmienky pre klienta alebo o produkty, ktoré boli nahradené inými produktami, resp. poplatkami, alebo produkty, ktoré sú používané alebo nastavované ojedinele. Napriek tomu, že je žiaduce, aby odporúčací algoritmus bol schopný odporúčať produkty s nižšou frekvenciou používania, produkty, ktoré majú nastavené jednotky až desiatky používateľov, je nutné riešiť osobitne a zvyčajne sa jedná o nepoužívané produkty. Okrem toho odstránenie týchto ojedinelých poplatkov z dátovej sady výrazne zredukovalo vektorovú rozmernosť, ktorej vysoká hodnota je hlavným problémom pri aplikovaní vzdialenostných funkcií.

Pre získanie kvalitných odporúčaní bolo taktiež potrebné odfiltrovať z množiny vstupných údajov účty, ktoré sú zatvorené, nepoužívané alebo účty, ktoré sú určené na špecifický účel a nie je im možné priradiť produkty z danej dátovej sady. Pre správne fungovanie odporúčacích algoritmov bol v dátovej sade ponechaný vždy len jeden účet pre klienta. Tým je zabezpečené, že vzdialenostné funkcie nebudú klásť neprimeranú váhu klientom s vyšším počtom účtov. Ďalej je predpokladané, že väčšina účtov jedného klienta má nastavené rovnaké podmienky, a teda je strata informácií o týchto účtoch zanedbateľná.

4.1.3 Transformácia

Okrem filtrácie nepotrebných a nežiaducich údajov je dôležitou súčasťou prípavy dátovej sady pre odporúčanie transformácia údajov pre potreby algoritmov. Jedna transformácia bola vykonaná už počas extrahovania dát priamo pomocou SQL príkazov. V tejto časti sú preskúmané ďalšie techniky pre skvalitnenie vstupnej množiny dát.

Algoritmy založené na porovnávacích funkciách vyžadujú ako vstup vektor číselných hodnôt. Z toho dôvodu je nutné priradiť všetkým textovým hodnotám číselný identifikátor. Pre túto transformáciu je však potrebné, aby boli hodnoty medzi sebou porovnateľné a identifikátory boli pridelené v hierarchickom poradí. Napríklad pri transformovaní typu klienta na základe jeho veľkosti (pozri časť 1.1.3) je možné prideliť identifikátory od malých podnikov s číselnou hodnotou 0 až po korporáty s číselnou hodnotou 1. Hodnoty na tejto škále sú porovnateľné a výsledok porovnávania je možné jednoducho interpretovať. Naopak, údaje ako identifikátory odvetvia nie je možné medzi sebou porovnávať, keďže odvetvia podnikania nemajú žiadnu hierarchickú štruktúru a výsledok takéhoto porovnania by nebolo možné interpretovať. Jedným z riešení takýchto údajov by bolo vytvorenie nového stĺpca pre každý jeden identifikátor, kde by bola obsahom pravdivostná

hodnota, či je daný používateľ identifikovaný s daným odvetvím podnikania. Takéto riešenie by však výrazne zvýšilo vektorovú rozmernosť a mohlo by viesť k zhoršenej kvalite výstupov.

Ďalším transformačným krokom vstupnej množiny dát je normalizácia hodnôt. Niektoré odporúčacie algoritmy môžu byť citlivé na extrémne hodnoty, ktoré môžu spôsobiť nesprávne začlenenie užívateľov do skupín alebo znížiť podobnosti medzi užívateľmi. Z toho dôvodu je žiaduce odstrániť extrémne hodnoty. Pre identifikáciu minimálnych a maximálnych akceptovateľných hodnôt bol pre každý údaj vytvorený histogram, v ktorom minimálne a maximálne hodnoty museli byť nastavené aspoň na konkrétnom percente užívateľov, aby boli prijaté za akceptovateľné minimá a maximá. Všetky extrémne hodnoty boli následne nastavené na hraničné hodnoty (angl. clipping).

Keďže je kosínusová podobnosť založená na geometrickej interpretácii hodnôt, je dôležité, aby všetky údaje mali rovnakú škálu hodnôt. Preto je potrebné všetky údaje normalizovať. Predpokladá sa, že sú všetky údaje uniformne rozložené, a preto bola pre túto dátovú sadu zvolená lineárna transformácia:

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{4.1}$$

Výsledné hodnoty sú na škále 0 až 1. Táto transformácia neovplyvní výsledky pri použití Pearsonovej korelácie. Údaje, ako napríklad obrat spoločnosti, nie je žiaduce normalizovať a pri zisťovaní veľkosti klienta na základe obratu je žiaduce použiť Euklidovskú vzdialenosť nenormalizovaných hodnôt.

Poplatky analyzované v časti 1.1.1 sú rozdelené do viacerých kategórií. Zatiaľ čo poplatky s fixnou alebo percentuálnou hodnotou je jednoduché reprezentovať pomocou jedinej číselnej hodnoty, ostatné poplatky sa skladajú z viacerých číselných hodnôt. Keďže je reprezentácia týchto poplatkov pre vstup do porovnávacích funkcií komplexným problémom, boli tieto údaje transformované na pravdivostnú hodnotu značiacu, či bolo klientovi udelné pre daný poplatok individuálne nastavenie. Tento princíp bol taktiež zvolený v ostatných riešeniach, ktoré ponúkajú bankové produkty (pozri časť 2.6).

Anonymizované číslo účtu	Analytické dáta								
	Údaje o klientovi						Údaje o účte		
	Obrat	Norm. veľkosť klienta	Počet účtov	Norm. počet účtov	ID odvetvia	...	Typ účtu	Zostatok na účte	...
1234657890	10 000 000	0.25	10	0.01	123456	...	123	1 000 000	...
...	...						...		

Anonymizované číslo účtu	Poplatky							
	Poplatky viazané na klienta				Poplatky viazané na účet			
	Vedenie internet. bank.	Odoslanie SMS	Vygenerovanie certifikátu	...	Vedenie účtu	Kreditná operácia	Debetná operácia	...
1234657890	1	0	0	...	1	1	0	...
...	...				...			

Tabuľka 4.1 Ukážka výslednej podoby dátovej sady. Analytické údaje sú transformované do číselných hodnôt a normalizované, ak to je možné. Poplatky majú značky 1 alebo 0, indikujúce individuálne nastavenie podmienok bez konkrétnej hodnoty. Atribúty v tabuľke sú ukážkové na základe analýzy z kapitol 1 a 2, a reflektujú štruktúru tabuľky. Ukážkové atribúty sa nemusia zhodovať so skutočnými aktibútmi použitej dátovej sady.

4.2 Meranie

V tejto časti práce sú vykonané samotné experimenty pre zistenie najvyhovujúcejšej odporúčacej techniky pre túto doménu. Najskôr je uvedený model, pomocou ktorého sú implementované všetky merania a vyhodnocovania. Následne sú vykonané experimenty nad pamäťovo a modelovo založenými odporúčacími technikami kolaboratívneho filtrovania.

4.2.1 Model

Pred samotným meraním je nevyhnutné stanoviť jednotné rozhranie pre všetky algoritmy. Toto rozhranie by malo umožňovať generovanie konzistentného výstupu na základe rovnakých vstupných dát, čo umožní následné hodnotenie výstupov. V tejto časti je popísaný model pre odporúčanie produknov znázornený na obrázku 4.1.

Základné dátové triedy slúžiace na prenos a uchovávanie vygenerovaných odporúčaní pre používateľov sú `UserRecommendations` a `FeeRecommendation`. Tieto triedy slúžia ako výstup odporúčacích algoritmov a následne sú poskytnuté používateľovi v transformovanej podobe a sú taktiež použité pre ohodnotenie daných techník. V každom odporúčacom algoritme slúži ako vstup premenná s informáciami o užívateľovi, pre ktorého budú generované odporúčania. Ďalej je potrebné použiť vzorku tzv. trénovacích dát, na základe ktorých bude generované odporúčanie. V tomto prípade boli dáta rozdelené na trénovacie a testovacie z originálnej množiny dát v pomere 4 k 1. Vstupné používateľské dáta typu `UserData` majú formu rovnakú ako to je popísané v tabuľke 4.1. Každý algoritmus teda vygeneruje odporúčania pre každého testovacieho používateľa, čo slúži ako vstup pre vyhodnotenie úspešnosti daných algoritmov.

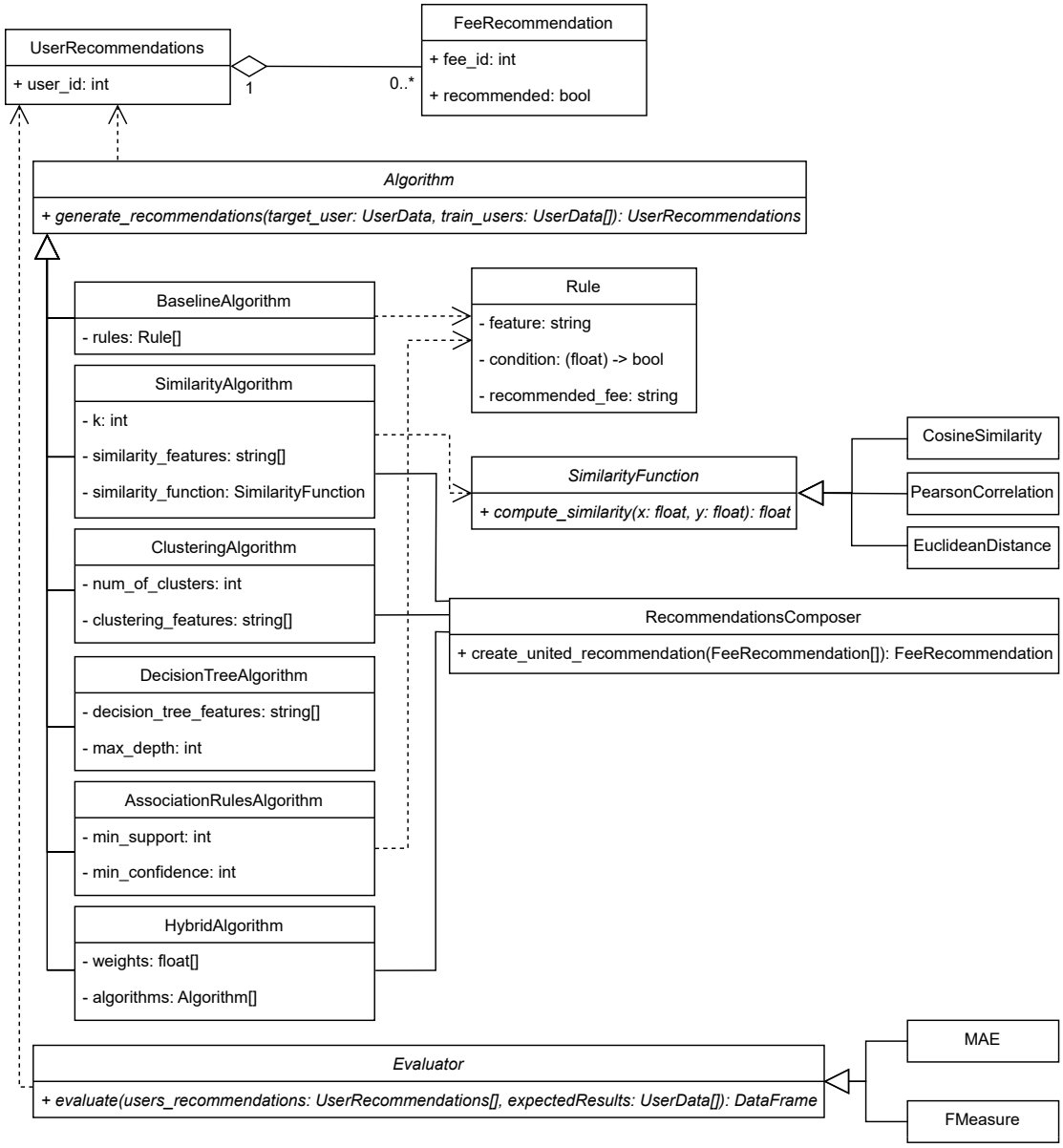
Zatiaľ čo algoritmy ako asociačné pravidlá alebo rozhodovacie stromy vracajú konkrétny výsledok v podobe, či sa má daný produkt odporučiť alebo nie, ostatné algoritmy vracajú ako výstup zotriedený zoznam užívateľov vhodných pre odporúčenie. V prípade podobnostných funkcií je tento zoznam generovaný na základe podobností medzi užívateľmi. Podobnostné funkcie `SimilarityFunction` vracajú v tomto prípade vždy normalizované hodnoty, ktoré následne slúžia ako váhy pri skladaní odporúčaní. V prípade zhlukovania sa jedná o zhluk užívateľov, ktorí majú v rámci jedneho zhluku medzi sebou rovnakú váhu pre odporúčanie danému užívateľovi. Pre zlúčenie týchto odporúčaní je použitá trieda `RecommendationsComposer`, ktorá pre každý poplatok vytvorí vážený priemer z daných odporúčaní a vráti výsledok, ktorý je bližší k jednej z možných hodnôt.

Pre vyhodnotenie sú použité metriky presnosti predikcií a presnosti klasifikácií. Ako vstup pre meranie kvality odporúčaní sú všetci používatelia z testovacej množiny a ich odporúčania pre každý poplatok. Keďže majú v tomto prípade všetky poplatky rovnakú škálu hodnôt, t.j. 0 až 1, je možné použiť nenormalizované verzie metrík. Metódy vracajú ako výstup atribúty typy `DataFrame` z Python knižnice `pandas`, aby bolo možné ďalej manipulovať s výsledkami meraní.

4.2.2 Algoritmy

Po tom, čo je definované jednotné rozhranie pre všetky odporúčacie techniky, je možné zistiť kvalitu daných odporúčaní pre každý algoritmus s použitím rôznych konfigurácií algoritmov. V modeli na obrázku 4.1 sú pri každom algoritme znázornené konfigurovateľné položky, ktoré budú predmetom merania.

Pre určenie referenčných bodov, na základe ktorých je možné určiť kvalitu ostatných odporúčacích techník, je najskôr vytvorený základný algoritmus pozostávajúci zo základných znalostí o probléme. Následne sú otestované techniky kolaboratívneho filtrovania na báze pamäte a následne sú použité techniky na báze modela. Na záver sú použité hybridné algoritmy na základe váženia pozostavájuce z dvoch alebo troch modelov.



Obr. 4.1 Diagram tried pre vygenerovanie a vyhodnotenie odporúčaní

4.2.2.1 Základný algoritmus

Pre overenie a vyhodnotenie správnosti fungovania odporúčacích výsledkov je potrebné si stanoviť základné kritéria, ktoré budú slúžiť ako referenčný bod pre vyhodnotenie efektívnosti algoritmu a pomôžu kvantifikovať, do akej miery dokážu tieto algoritmy optimalizovať danú úlohu.

Pre tento referenčný bod bol vytvorený primitívny algoritmus pozostávajúci z jednoduchých porovnávaní používateľských atribútov. V prípade, že daný atribút dosiahol konkrétnu hodnotu, bude používateľovi pridelená konkrétna zľava. Tieto atribúty boli vybrané na základe znalosti domény a na základe najfrekventovanejšie používaných algoritmov. Ako príklad môže slúžiť prípad, kedy má používateľ mnoho účtov, a dáva teda zmysel vytvoriť používateľovi individuálnu cenu pre vedenie účtov. Jedná sa teda o manuálne naimplementované základné asociačné pravidlá.

Po vykonaní testov na množine testovacích používateľov bolo zistené, že priemerná absolútna

chybovosť (MAE) dosahuje hodnoty 0.4932 a metrika F-measure dosahuje skóre 0.0160. Jedná sa teda o veľmi nekvalitné výsledky, čo značí, že manuálna implementácia odporúčacich pravidiel nie je dostačujúca a je žiaduce pre tento účel vytvoriť komplexné odporúčacie techniky.

4.2.2.2 Techniky na báze pamäte

Táto časť je venovaná výpočtu modelov založených na užívateľsky založenom kolaboratívnom filtrovaní s použitím vzdialenostných funkcií. Kosínusová podobnosť a pearsonova korelácia budú využité pre údaje o poplatkoch, ktoré sú normalizované a dostupné v jednotnej forme. Ako vstup je určený vektor zložený zo všetkých poplatkov, ktoré sa zachovali po filtračnom procese. Euklidovská vzdialenosť je určená najmä pre zistenie veľkosti klienta, a teda bude zameraná na vstupy z analytickej časti dátovej sady.

Podstatou merania je každého testovacieho používateľa porovnať s množinou používateľov určených na trénovanie. Zložitosť tohto procesu je $O(n^2)$, čo s prihliadnutím na veľkosť dátovej sady pozostáva z rádovo miliónov operácií a exekúcia tohto procesu trvá niekoľko desiatok minút. Pre urýchlenie tohto procesu bol použitý fond vlákien (angl. thread pool) s využitím knižnice `multiprocessing` určenej pre jazyk Python.

4.2.2.2.1 Poplatky

Pri zvyčajnej aplikácií vzdialenostných funkcií pre účely odporúčania produktov je typicky niekoľko alebo väčšina produktov používateľmi neohodnotených a tieto produkty sa typicky stávajú predmetom odporúčaní. V prípade poplatkov platí, že klient musí mať nastavené nejaké hodnoty takmer vždy – ak klient nemá zjednané individuálne poplatkovanie, je použitá cenníková cena produktu. V tomto prípade odporúčania bankových produktov teda nie je možné použiť prístup zamerania sa na nedefinované hodnoty.

Pre odporúčanie produktov je možné v tomto prípade použiť produkty, pre ktoré nie je nastavená žiadna zľava. Z toho dôvodu je možné tieto techniky aplikovať jedine pre existujúceho klienta, ktorý už má aplikované niektoré zľavy. Pre účel testovania je potrebné určiť, či algoritmus dokáže správne predpovedať zľavnené aj nezľavnené produkty. Pre tento účel bolo náhodne odstránených 25 percent nastavení a cieľom testovania bolo určiť hodnody pre tieto hodnoty.

Obe techniky vracajú ako výstup zotriedený zoznam najpodobnejších používateľov. Počet používateľov, na základe ktorých sa vytvára odporúčanie, môže ovplyvniť výslednú kvalitu odporúčania, a preto bol tento parameter záujmom merania. Výsledky meraní je možné pozorovať v tabuľkách 4.2 a 4.3. Po aplikovaní viacerých hodnôt bolo zistené, že najlepšie výsledky oboch funkcií je možné dosiahnuť pri použití 5. až 10. najlepších odporúčaní.

Počet odporúčaní	MAE	Presnosť	Úplnosť	F-measure
3	0.1804	0.7467	0.7201	0.7332
5	0.1740	0.7477	0.7976	0.7718
10	0.1767	0.7566	0.7844	0.7703
15	0.1815	0.7400	0.7231	0.7314
20	0.1867	0.7360	0.7216	0.7287
30	0.1869	0.7492	0.7051	0.7265
50	0.1892	0.7449	0.7009	0.7223

Tabuľka 4.2 Výsledok meraní pri použití kosínusovej podobnosti pre poplatky s využitím prvých n najlepsích odporúčaní

Z meraní je možné pozorovať, že obe vzdialenostné funkcie dosahujú výrazne nižšiu chybovosť a vyššiu kvalitu výstupov oproti primitívnemu riešeniu. Aplikácia kosínusovej vzdialenosti má priemernú absolútnu chybovosť 0.1740 a aplikácia pearsonovej korelácie má priemernú absolútnu

Počet odporúčaní	MAE	Presnosť	Úplnosť	F-measure
3	0.1968	0.6849	0.7134	0.6989
5	0.1981	0.6998	0.7459	0.7221
10	0.1996	0.7100	0.7364	0.7230
15	0.2012	0.7021	0.6895	0.6957
20	0.2048	0.6980	0.6792	0.6885
30	0.2077	0.7088	0.6602	0.6836
50	0.2433	0.7221	0.6617	0.6905

Tabuľka 4.3 Výsledok meraní pri použití pearsonovej korelácie pre poplatky s využitím prvých n najlepsích odporúčaní

chybovosť 0.1981. Vo väčšine aplikácií dosahuje porovnávanie na základe pearsonovej korelácie lepšie výsledky oproti porovnávaniu pomocou kosínusovej vzdialenosti. Hoci v tomto prípade tento rozdiel nie je príliš výrazný, vysvetlenie tohto javu je možné vysvetliť jednoduchosťou vstupných údajov. Keďže vektory môžu obsahovať jedine hodnoty 0 alebo 1, kde väčšina z hodnôt je nulových, je možné lepšie uplatniť geometrickú reprezentáciu hodnôt oproti štatistickej reprezentácii, ktorá skúma lineárnu závislosť hodnôt.

4.2.2.2.2 Veľkosť klienta

Väčšina odporúčacích techník sa pri odporúčaní údajov snaží eliminovať rozdiely vzniknuté na základe rôznych hodnotiacich škál každého používateľa. Táto stratégia môže byť užitočná napríklad pri hodnotení filmov, kde niektorí používatelia môžu udeľovať nižšie priemerné ohodnotenia a naopak. V bankovej doméne je žiaduce uvážiť taktiež veľkosť klienta, ktorá môže mať zásadnú rolu pri udeľovaní individuálnych cenových podmienok za produkty, keďže klienti s vyšším obratom musia nutne generovať vyšší zisk a je teda o ne vyšší záujem zo strany bánk.

Identifikátory použitých atribútov	MAE	Presnosť	Úplnosť	F-measure
1, 2, 3, 4, 5, 6	0.2286	0.6539	0.5707	0.6095
1, 3, 4, 5, 6	0.2255	0.6595	0.5707	0.6119
1, 2, 5, 6	0.2268	0.6714	0.5569	0.6088
1, 5, 6	0.2275	0.6672	0.5597	0.6088
1, 2, 3, 4	0.2322	0.6486	0.5696	0.6065
3, 4, 5, 6	0.2311	0.6520	0.5682	0.6072
1, 2, 5, 6	0.2268	0.6714	0.5569	0.6088

Tabuľka 4.4 Výsledok meraní pri použití euklidovskej vzdialenosti pre analytické dáta s použitím rôznych údajov z analytickej časti dátovej sady.

Pre objekt merania boli zvolené atribúty, ktoré reflektujú veľkosť toku financií, ako napríklad celkový obrat spoločnosti alebo súčet prichádzajúcich alebo odchádzajúcich platieb. Tieto atribúty boli použité ako vstupné hodnoty do funkcie v rôznych kombináciách, ako to je zobrazené na výsledku meraní v tabuľke 4.4. Z uvedených hodnôt je možné usúdiť, že samotná kombinácia atribútov nemá zásadný vplyv na kvalitu výstupu. Dôvod tohto výsledku spočíva v tom, že dané atribúty sú priamo úmerné, kde pri raste hodnoty atribútu reprezentujúceho finančný objem klienta musí nutne rásť aj iný atribút podobného charakteru a naopak. Celková priemerná absolútna chybovosť a kvalita výstupov je výrazne horšia ako pri použití vzdialenostných funkcií pre poplatky, hoci výsledky sú taktiež kvalitnejšie oproti základnej implementácii. Samostatné

použitie tejto vzdialenostnej funkcie je nevyhovujúce a v ďalších fázach merania bude skúmané, či táto technika dokáže vylepšiť kvalitu výstupu v kombinácii s inou technikou.

4.2.2.3 Techniky na báze modela

Po aplikácií pamäťovo založených techník kolaboratívneho filtrovania primárne na poplatkovú časť dátovej sady sú predmetom merania techniky, ktoré sú založené na báze modela. Tieto techniky budú primárne aplikované na analytickú časť dátovej sady, ako to bolo použité v podobných riešeniach v bankovej doméne. V tejto práci budú analyzované modely založené na asociačných pravidlách, zhlukovaní a rozhodovacích stromoch.

4.2.2.3.1 Asociačné pravidlá

Pre implementáciu asociačných pravidiel boli použité dáta z analytickej časti ako predpoklady z analytickej časti dátovej sady. Hodnoty v každých atribútoch boli rozdelené do niekoľkých intervalov v závislosti od škály hodnôt daného atribútu a na základe analytickej znalosti o daných údajoch. Ako záver každého pravidla bol identifikátor poplatku, ktorému je možné priradiť individuálne cenové nastavenie v prípade, že daný používateľ spĺňa daný predikát, čo znamená, že konkrétny atribút sa nachádza v danej škále hodnôt.

K tomu, aby bolo možné vytvoriť pravidlo, musí istý počet používateľov s individuálnym nastavením cien spĺňať danú podmienku. Na základe merania uvedeného v tabuľke 4.5 je možné pozorovať, že pri zvolení príliš nízkej minimálnej spoľahlivosti je zachytená väčšina prípadov pre udelenie individuálnych cenových podmienok, hoci mnoho z nich je udelených nesprávne. Na druhej strane, pri nastavení príliš striktnej minimálnej spoľahlivosti je počet nesprávne udelených individuálnych cenových podmienok nízky, ale nie sú zachytené všetky potrebné prípady, kedy je žiaduce udeliť dané individuálne nastavenie. Ako optimálna hodnota minimálnej spoľahlivosti bola zvolená hodnota 60 percent. Pri tejto konfigurácii dosahujú asociačné pravidlá vyššiu úspešnosť ako porovnávanie na základe euklidovskej vzdialenosti, hoci výsledky nedosahujú kvalitu porovnávacích funkcií použitých na atribúty zamerané na poplatky.

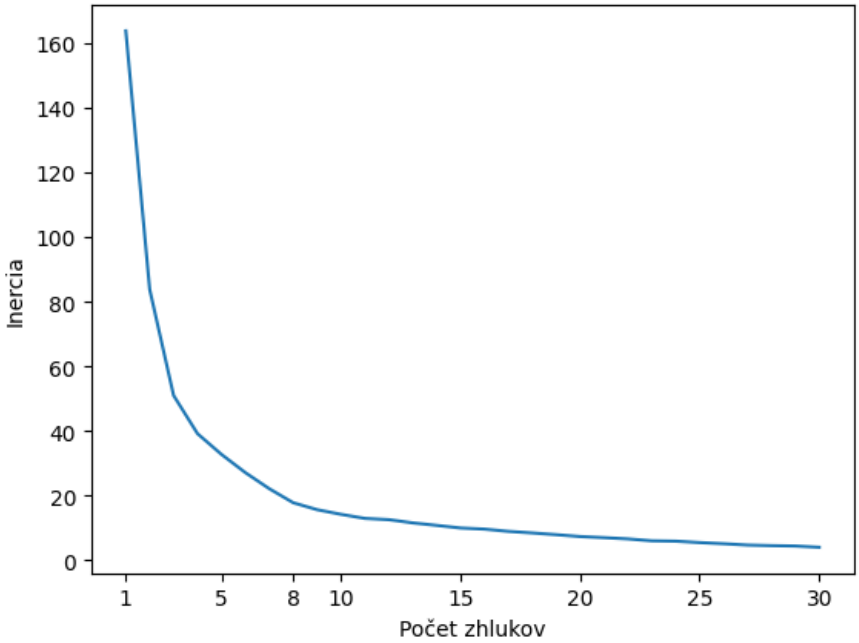
Minimálna spoľahlivosť	MAE	Presnosť	Úplnosť	F-measure
0.1	0.5943	0.1069	0.9643	0.1924
0.2	0.4347	0.1651	0.8536	0.2766
0.3	0.3304	0.2850	0.8032	0.4207
0.4	0.2837	0.4008	0.7513	0.5227
0.5	0.2696	0.5861	0.6967	0.6366
0.6	0.2486	0.6528	0.6376	0.6451
0.7	0.3168	0.7136	0.4831	0.5761
0.8	0.4053	0.7532	0.4004	0.5228
0.9	0.4588	0.8034	0.3569	0.4942

Tabuľka 4.5 Výsledok meraní pri použití asociačných pravidiel pre analytické dáta ako predpoklady a poplatky ako závery

4.2.2.3.2 Zhlukovanie

Pre odporúčanie individuálnych nastavení na poplatky bolo aplikované zhlukovanie na rôzne analytické atribúty pomocou algoritmu k-means. Ako vzdialenostná funkcia pre porovnávanie jednotlivých bodov bola použitá euklidovská vzdialenosť, ako je to typické pri aplikovaní tohto algotitmu. Väčšina hodnôt použitých ako vstup pre funkciu je však normalizovaná. Pre aplikáciu bola použitá knižnica `sklearn`.

Pre správne fungovanie algoritmu je v prvom rade potrebné určiť počet zhlukov. Tento počet bol zistený pomocou lakťovej metódy (angl. Elbow method) s konštantným nastavením použitých atribútov. Výsledok merania je možné pozorovať na obrázku 4.2. Vzhľadom na veľkosť dátovej sady a počet použitých atribútov bol určený ako ideálny počet zhlukov osem, ktorý bol vyhodnotený ako bod zlomu grafu.



Obr. 4.2 Nájdenie ideálneho počtu zhlukov pre algoritmus *k*-means pomocou lakťovej metódy (Elbow method)

Počet použitých atribútov	Identifikátory použitých atribútov	MAE	Presnosť	Úplnosť	F-measure
17	9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25	0.2579	0.6452	0.5811	0.6115
9	9, 10, 11, 12, 13, 19, 20, 23, 25	0.1935	0.7120	0.6947	0.7032
8	9, 11, 12, 13, 19, 20, 23, 25	0.1977	0.7096	0.6905	0.6999
6	9, 11, 12, 13, 23, 25	0.2095	0.7002	0.6787	0.6892
5	9, 12, 13, 23, 25	0.2105	0.6999	0.6756	0.6875
4	9, 12, 13, 25	0.2347	0.6656	0.6506	0.6580
3	12, 13, 25	0.2447	0.6345	0.6049	0.6194

Tabuľka 4.6 Výsledok meraní pri použití zhlukovania pre analytické dáta na základe rôznych kombinácií vstupných atribútov

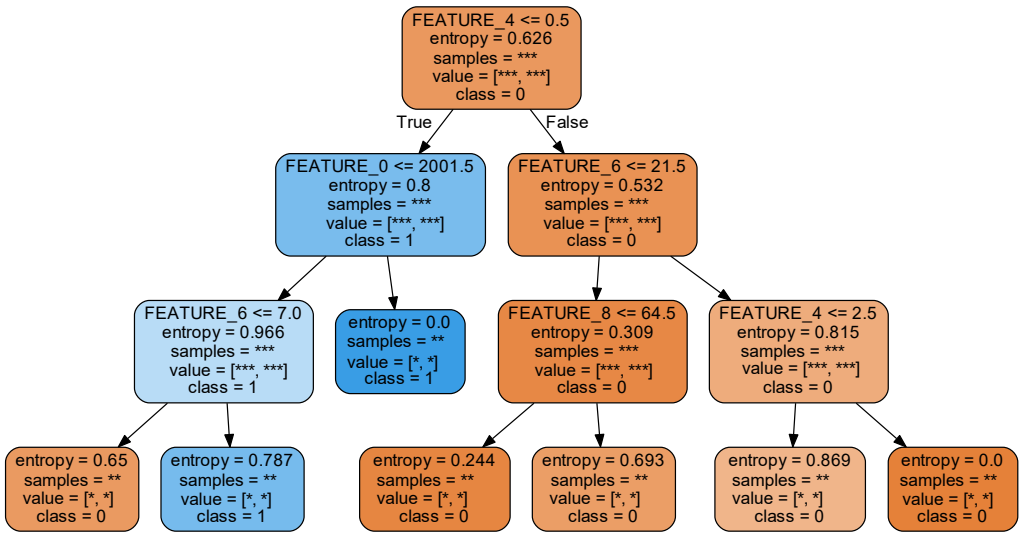
Po tom, čo bol určený počet zhlukov, je možné vyhodnotiť úspešnosť daného algoritmu. Ako objekt pre meranie boli opäť použité rôzne kombinácie analytických atribútov. Výsledok merania je možné pozorovať v tabuľke 4.6. Ako ideálny počet atribútov pre vytvorenie zhlukov je 9 atribútov. Výsledné odporúčania boli vytvorené na základe kombinácie poplatkov ostatných používateľov v rovnakom zhluku. Keďže je v danom zhluku každý používateľ pre účely odporúčania rovnocenný, boli skombinované odporúčania na základe všetkých používateľov na základe

rovnakej váhy pre každé odporúčanie. Priemerná absolútna chybovosť a kvalita výstupov tohto algoritmu je veľmi dobrá a hodnoty dosahujú podobné hodnoty ako vzdialenostné funkcie aplikované na poplatky.

4.2.2.3.3 Rozhodovacie stromy

Pre implementáciu rozhodovacích stromov boli ako vnútorné uzly stromu, t.j. testy pre atribúty, použité dáta z analytickej časti dátovej sady. Listy stromu sú reprezentované vo forme odpovede, či daný poplatok má byť odporučený pre individuálne nastavenie alebo nie. Pre každý poplatok bol teda vytvorený individuálny rozhodovací strom, na základe ktorého je možné vytvoriť odporúčanie každému používateľovi. Oproti zhlukovaniu poskytujú rozhodovacie stromy jediný výstup, a teda nie je nutné odporúčania kombinovať.

Rozhodovacie stromy boli opäť implementované pomocou knižnice `sklearn` určenej pre jazyk Python. Bola zvolená implementácia na základe entrópie daných vlastností. Cieľom merania bolo zistiť ideálnu maximálnu hĺbku stromu. Výsledok meraní je znázornený v tabuľke 4.7. Ako ideálna maximálna hĺbka stromu bola zvolená hodnota 3. Výsledná priemerná absolútna chybovosť a kvalita výstupov však nie je dostačujúca, keďže technika zhlukovania ponúka výrazne kvalitnejšie spracovanie analytických údajov dátovej sady.



Obr. 4.3 Vizualizácia rozhodovacieho stromu s maximálnou hĺbhou 3 a anonymizovanými údajmi. Rozhodnutie o odporúčaní daného produktu pozostáva z entrópie atribútov určených na analytické spracovanie.

4.2.2.4 Hybridné modely

Na základe analýzy odporúčacích systémov aplikovaných v bankovom sektore (pozri časť 2.6) je pre dosiahnutie vysokej kvality výstupov žiaduce implementovať hybridné modely pre odporúčanie. V predchádzajúcich meraniach boli implementované odporúčacie algoritmy pre analytické údaje o klientoch a údaje o poplatkoch. V tejto časti sú analyzované hybridné modely na základe kombinácie týchto dvoch typov údajov a techník, a to konkrétne zhlukovanie analytických údajov skombinované s kosínusovou podobnosťou pre poplatkové údaje, a rozhodovacie stromy skombinované opäť s kosínusovou podobnosťou pre poplatkové údaje. Na záver je implementovaný

hybridný model na základe kombinácie zhlukovania a euklidovskej vzdialenosti pre analytické údaje s kosínusovou podobnosťou pre poplatkové údaje.

Maximálna hĺbka stromu	MAE	Presnosť	Úplnosť	F-measure
2	0.2968	0.6569	0.5710	0.6109
3	0.2954	0.6721	0.5589	0.6103
4	0.2993	0.6624	0.5612	0.6076
5	0.3035	0.6487	0.5670	0.6051
6	0.3057	0.6439	0.5679	0.6035
7	0.3077	0.6404	0.5676	0.6018
8	0.3156	0.6258	0.5676	0.5953
9	0.3195	0.6171	0.5693	0.5922
10	0.3290	0.6003	0.5684	0.5839
∞	0.3426	0.5715	0.5732	0.5724

Tabuľka 4.7 Výsledok meraní pri použití rozhodovacích stromov pre analytické dáta pre rôznu maximálnu hĺbku stromu

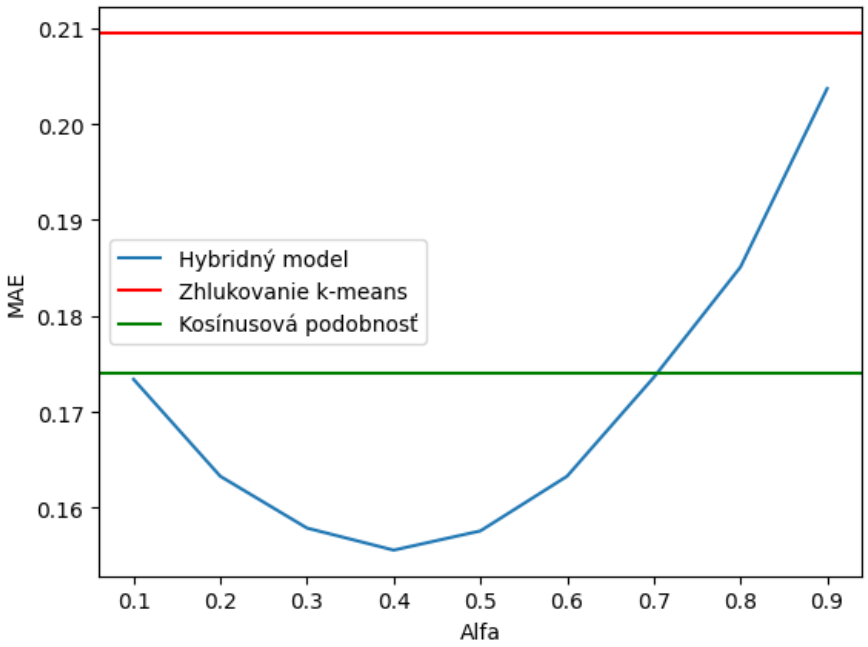
4.2.2.4.1 Zhlukovanie a kosínusová vzdialenosť

V predchádzajúcich experimentoch dosahovali údaje namerané pomocou zhlukovania a porovnávania na základe kosínusovej vzdialenosti maximálne hodnoty, a teda je predpokladané, že kombinácia týchto techník dokáže skvalitniť výstupy odporúčaní. Cieľom tohto merania je vytvorenie váženého hybridného systému pozostávajúceho zo všetkých odporúčaní vytvorených zo zhluku používateľov a doplnených chýbajúcich odporúčaní na základe podobnostnej funkcie.

V tejto časti je predmetom merania nájdenie ideálnej váhy pre obe techniky. Výsledky meraní sú znázornené v tabuľke 4.8 a v grafe na obrázku 4.4. Z nameraných hodnôt vyplýva, že pri nastavení váh 2 ku 3 sa výrazne zlepšila chybovosť a celkové výstupy odporúčaní.

Parameter alfa	Parameter beta	MAE	Presnosť	Úplnosť	F-measure
0.1	0.9	0.1734	0.7593	0.7994	0.7788
0.2	0.8	0.1633	0.7631	0.8031	0.7825
0.3	0.7	0.1579	0.7691	0.8193	0.7934
0.4	0.6	0.1556	0.7786	0.8248	0.8010
0.5	0.5	0.1576	0.7663	0.8036	0.7845
0.6	0.4	0.1633	0.7550	0.7705	0.7626
0.7	0.3	0.1736	0.7473	0.7563	0.7517
0.8	0.2	0.1851	0.7358	0.7287	0.7322
0.9	0.1	0.2037	0.7230	0.7001	0.7113

Tabuľka 4.8 Výsledok meraní pri použití hybridného modelu pomocou zhlukovania a porovnávania pomocou kosínusovej vzdialenosti. Parameter alfa určuje váhu zhlukovacieho algoritmu, parameter beta určuje váhu porovnávania pomocou kosínusovej vzdialenosti.



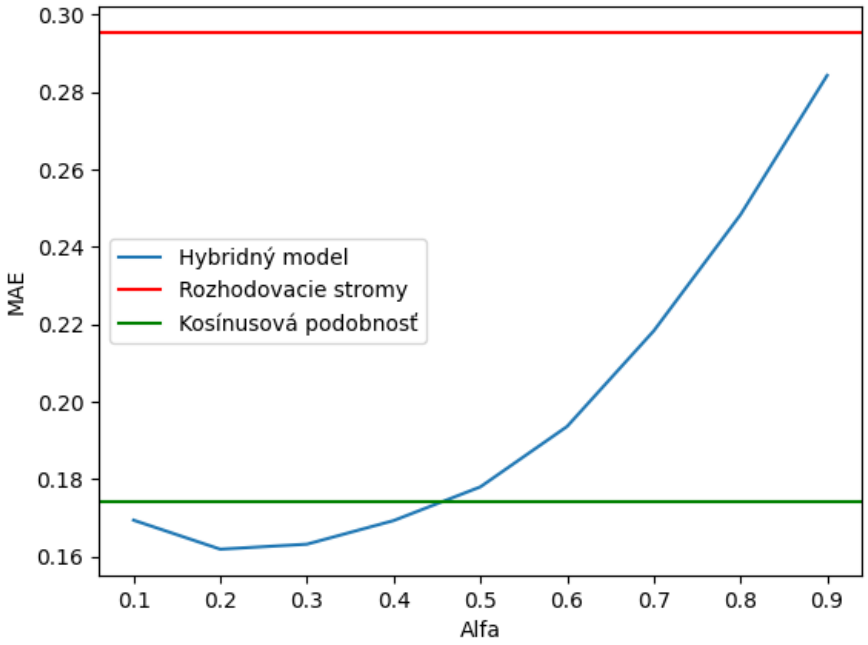
Obr. 4.4 Priemerná absolútna chybovosť pri použití hybridného modelu pomocou zhlukovania a porovnávania pomocou kosínusovej vzdialenosti. Parameter alfa určuje váhu zhlukovacieho algoritmu.

4.2.2.4.2 Rozhodovacie stromy a kosínusová podobnosť

Podľa analýzy v kapitole 2 boli očakávané menej kvalitné výsledky pri použití rozhodovacích stromov, keďže táto technika najlepšie funguje s použitím iných modelov a jej účelom je skvalitniť vstupné dáta použité v inom modeli. Preto je vhodné preskúmať hybridizáciu pomocou rozhodovacích stromov pre analytické údaje a porovnávania poplatkov pomocou kosínusovej vzdialenosti, ktorá mala doteraz najlepšie výsledky. Na základe meraní znázornených v tabuľke 4.9 a v grafe na obrázku 4.5. Algoritmus dosahuje najlepšie výsledky pri pomere váh rozhodovacích stromov ku kosínusovej vzdialenosti 1 k 4. Výsledky dosahujú mierne zlepšenie oproti jednotlivým modelom, ale napriek tomu dosahuje hybritný model pomocou zhlukovania a kosínusovej vzdialenosti lepšie výsledky.

Parameter alfa	Parameter beta	MAE	Presnosť	Úplnosť	F-measure
0.1	0.9	0.1694	0.7594	0.7532	0.7562
0.2	0.8	0.1619	0.7650	0.7879	0.7762
0.3	0.7	0.1632	0.7632	0.7624	0.7628
0.4	0.6	0.1693	0.7605	0.7403	0.7502
0.5	0.5	0.1780	0.7574	0.6943	0.7244
0.6	0.4	0.1936	0.7290	0.6681	0.6972
0.7	0.3	0.2183	0.7163	0.6315	0.6712
0.8	0.2	0.2483	0.6394	0.5961	0.6169
0.9	0.1	0.2843	0.5731	0.5751	0.5740

Tabuľka 4.9 Výsledok meraní pri použití hybridného modelu pomocou rozhodovacích stromov a porovnávania pomocou kosínusovej vzdialenosti. Parameter alfa určuje váhu algoritmu rozhodovacích stromov, parameter beta určuje váhu porovnávania pomocou kosínusovej vzdialenosti.



Obr. 4.5 Priemerná absolútna chybovosť pri použití hybridného modelu pomocou rozhodovacích stromov a porovnávania pomocou kosínusovej vzdialenosti. Parameter alfa určuje váhu rozhodovacích stromov.

4.2.2.4.3 Zhlukovanie, euklidovská vzdialenosť a kosínusová podobnosť

Na základe predchádzajúcich meraní dosahoval doposiaľ najlepšie výsledky hybridný model pozostávajúci zo zhlukovania a porovnávania na základe kosínusovej vzdialenosti. Cieľom tejto časti je pokus tento model rozšíriť o ďalšiu techniku – začlenenie veľkosti klienta pomocou euklidovskej vzdialenosti zameranej na konkrétne analytické atribúty, ktorá individuálne nedosahovala významné výsledky.

Pre objekt merania bola opäť použitá konfigurácia váh pre jednotlivé modely. Pre otestovanie jednotlivých kombinácií bolo celkovo vykonaných 36 meraní. Najlepšie výsledky dosahujú konfigurácie s najnižšou váhou udelenou porovnávaniu na základe euklidovskej vzdialenosti. Najlepšie výsledky boli dosiahnuté v pomere váh algoritmom zhlukovania ku euklidovskej vzdialenosti ku kosínusovej vzdialenosti 3 : 1 : 6. Priemerná absolútna chybovosť tejto konfigurácie dosahovala hodnoty 0.1580. Pridanie ďalšieho modelu teda nezlepšilo doposiaľ najlepšie namerané výsledky.

4.3 Vyhodnotenie

V tejto časti boli použité a zmerané algoritmy kolaboratívneho filtrovania zamerané zvlášť na analytickú a poplatkovú časť dátovej sady. Pre porovnávanie poplatkov medzi používateľmi boli použité porovnávacie funkcie na základe kosínusovej vzdialenosti a pearsonovej korelácie. Z nameraných hodnôt bolo zistené, že pre daný prípad generuje mierne kvalitnejšie výsledky porovnávanie pomocou kosínusovej vzdialenosti. Pre analytickú časť boli použité techniky na báze modela na základe asociačných pravidiel, zhlukovania a rozhodovacích stromov. Spomedzi týchto techník bola najefektívnejšia technika zhlukovania. Pre dosiahnutie lepších výsledkov boli aplikované hybridné modely, spomedzi ktorých dosahoval najlepší výsledok model zložený zo zhlukovania a

kosínusovej vzdialenosti. Tento model generuje doposiaľ najkvalitnejšie výsledky spomedzi všetkých meraných algoritmov, a preto bude použitý pre implementáciu v rámci mikroslužby v ďalšej časti práce. Všetky implementované modely vyprodukovali kvalitnejšie merania ako základný primitívny model, ktorý bol určený ako referenčný bod pre túto prácu.

Všetky merania dosiahli pomerne vysokú presnosť a úplnosť. Tento jav môže byť spôsobený tým, že väščina poplatkov je prirodzene nastavená na cenníkovú hodnotu, čím môžu byť výsledné odporúčania skreslené a odporúčanie nenavrhnúť žiadny poplatok za individuálnych podmienok môže mať relatívne vysokú úspešnosť. Na druhej strane, vygenerované odporúčania dosahujú stále kvalitnejšie výsledky ako odporúčanie založené na neodporúčaní žiadneho produktu. Pre budúce kroky je preto vhodné produkty a poplatky za produkty ďalej zredukovať a prípadne ich rozdeliť do kategórií pre parciálne odporúčania a pre každú kategóriu navrhnúť individuálnu odporúčaciu metódu. Pre zlepšenie odporúčaní je taktiež vhodné rozšíriť analytické dáta okrem všeobecných údajov o ďalšie hodnoty, ktoré majú spojitosť s danými poplatkami, vďaka čomu môže byť zvýšená kvalita rozhodovacích stromov. V neposlednom rade je nutné brať do úvahy náklady spojené na vývoj týchto algoritmov a zhodnotenie celkovej prospešnosti daného projektu.

Metóda	MAE	Presnosť	Úplnosť	F-measure
Kosínusová podobnosť	0.1740	0.7477	0.7976	0.7718
Pearsonova korelácia	0.1981	0.6998	0.7459	0.7221
Euklidovská vzdialenosť	0.2255	0.6595	0.5707	0.6119
Asociačné pravidlá	0.2486	0.6528	0.6376	0.6451
Zhlukovanie	0.1935	0.7120	0.6947	0.7032
Rozhodovacie stromy	0.2954	0.6721	0.5589	0.6103
Zhlukovanie a kosínusová podobnosť	0.1556	0.7786	0.8248	0.8010
Rozhodovacie stromy a kosínusová podobnosť	0.1619	0.7650	0.7879	0.7762
Zhlukovanie, kosínusová podobnosť a euklidovská vzdialenosť	0.1579	0.7632	0.8063	0.7841

Tabuľka 4.10 Zhrnutie nameraných hodnôt podľa metódy

Kapitola 5

Návrh architektúry aplikácie

Táto kapitola je venovaná návrhu mikroslužieb pre doménu zameranú na správu klientských poplatkov. Mikroslužby musia byť schopné komunikovať s ostatnými zdieľanými bankovými komponentmi pre dosiahnutie čo najvyššej prepoužiteľnosti a musia poskytovať funkčnosti pre viaceré používateľské aplikácie.

Jednou z obsluhovaných aplikácií je aplikácia pre zobrazenie klientskych nastavení. Táto aplikácia má informatívny charakter a poskytuje používateľom prehľad o poplatkoch klienta oproti cenníkovemu nastaveniu. Aplikácia pre správu bankových procesov zastrešuje všetky auditovateľné procesné úkony v banke. V rámci tejto aplikácie je vytvorený nový proces pre nastavenie klientských poplatkov, v rámci ktorého budú použité komponenty z prvej zmienenej aplikácie spolu s prvkami potrebnými pre nastavovanie nových podmienok. Nové komponenty bude taktiež používať reprezentatívna webová stránka banky slúžiaca na prehľad finančných služieb, v rámci ktorej je potrebné zobraziť cenník služieb.

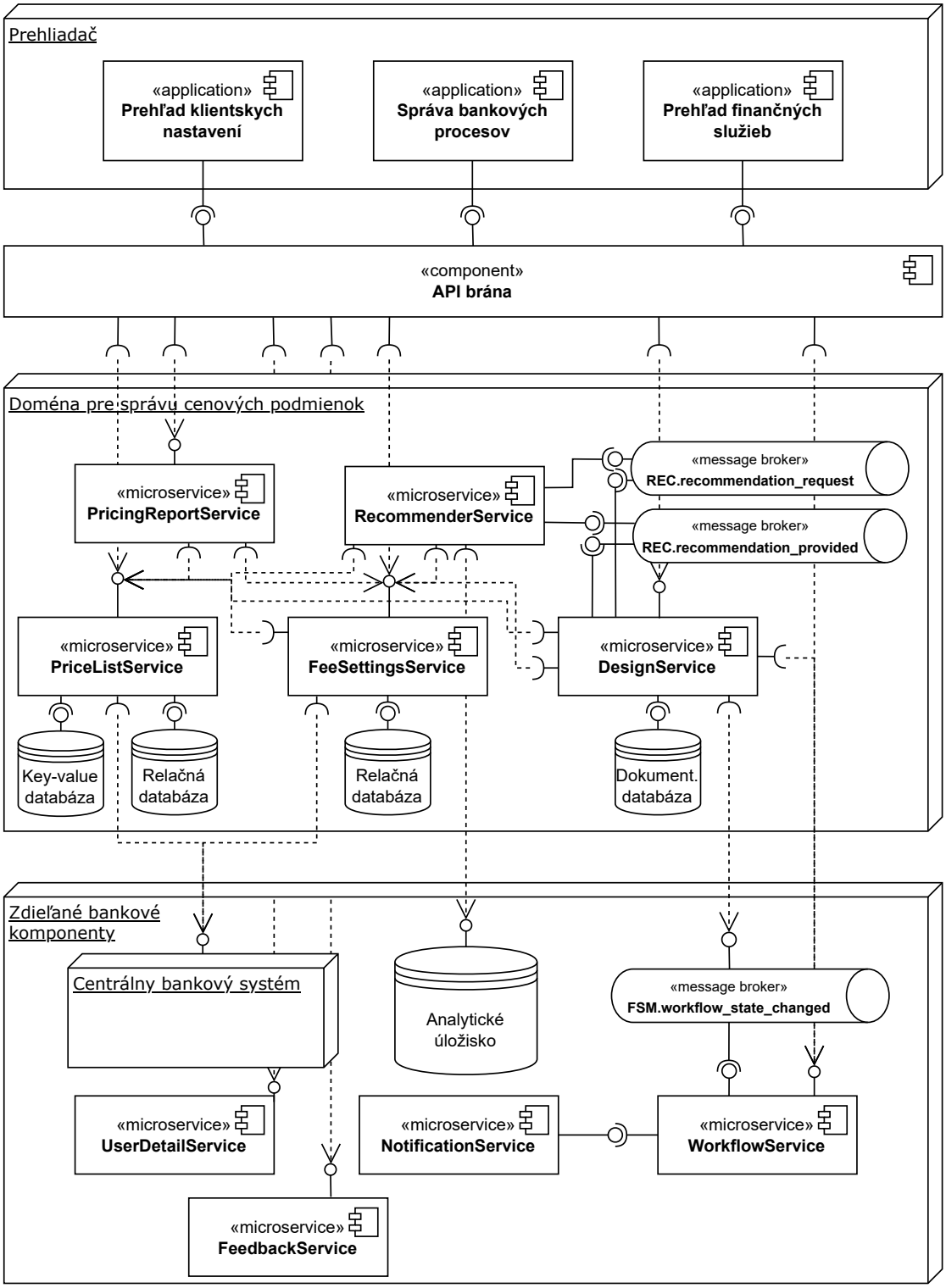
5.1 Mikroslužby

V tejto časti sú popísané vytvorené mikroslužby potrebné pre naplnenie všetkých funkčných aj nefunkčných požiadaviek. Okrem nových mikroslužieb sú využité aj niektoré zdieľané bankové mikroslužby, ktoré sú pripravené na použitie a je potrebné ich len začať konzumovať webovou aplikáciou alebo inou službou. Jedná sa typicky o mikroslužby zamerané na spoločné prípady užitia, ako je napríklad získanie spätnej väzby, detailu klienta alebo odosielanie notifikácií. Diagram zobrazujúci použité komponenty a závislosti medzi nimi je zobrazený na obrázku 5.1. Ďalšie časti tejto podkapitoly sú venované novým mikroslužbám navrhnutým v rámci tejto práce.

5.1.1 Služba pre získanie cenníka

Ako prvou navrhnutou mikroslužbou, ktorá je navrhnutá na základe biznisových vlastností aplikácie, je služba pre získanie cenníka. Táto služba má potenciál vysokej prepoužiteľnosti a rozšíriteľnosti aj v ostatných bankových systémoch, a preto má zmysel udržiavať túto funkcionalitu samostatne.

Služba je využívaná aplikáciou pre prehľad klientskych nastavení, aplikáciou pre správu bankových procesov, ale aj webom banky slúžiacom verejnosti, ktorý slúži ako prehľad finančných služieb. Keďže je táto služba dostupná verejne, je pri implementácii žiaduce klásť dôraz na autorizáciu používateľov a poskytovať detailné údaje o produktoch na základe rolí používateľa. Táto služba je taktiež používaná všetkými ostatnými službami v danej doméne, keďže práca s cenníkom



Obr. 5.1 Diagram komponentov použitých pri návrhu mikroslužieb. Navrhnuté mikroslužby sú zobrazené v bloku zapuzdrujúcom doménu pre správu cenových podmienok

	PriceListService	FeeSettingsService	UserDetailService	FeedbackService	DesignService	WorkflowService	RecommenderService
Aplikácia pre prehľad klientskych nastavení	×	×	×	×			
Aplikácia pre správu bankových procesov	×	×	×	×	×	×	×
Mikroslužba PricingReportService	×	×	×				
Aplikácia pre prehľad finančných služieb	×						

Tabuľka 5.1 Prehľad závislostí vybraných komponentov (riadky) na výbraných mikroslužbách z domény pre správu cenových podmienok a ďalších zdieľaných mikroslužbách (stĺpce)

je esenciálna súčasť ďalších operácií. Keďže sa jedná o komponent s vysokou previazanosťou s ostatnými komponentmi, je kritické, aby táto služba odpovedala rýchlo, bola odolná voči chybám a mala zabezpečenú spoľahlivú replikáciu. Na druhej strane sa jedná o službu s primárnym účelom načítavania údajov, a preto je riziko vyčerpania zdrojov servera alebo možnosť zablokovania databázy minimálna.

Pre zamedzenie duplikovania údajov a možnosti analýzy údajov medzi rôznymi verziami cenníka je žiaduce udržiavať databázu vo vysoko normalizovanom tvare, a to aj z dôvodu budúcej prepoužiteľnosti a rozšíriteľnosti. Pre ukladanie perzistentných dát bola z týchto dôvodov zvolená relačná databáza. Na základe vysokého podielu čítacích operácií je možné do databázy umiestniť zvýšený počet indexov pre urýchlenie načítavania.

Okrem získavania údajov z databázy je pre kompletné zostavenie cenníka potrebné získať údaje z centrálneho bankového systému, ktorého účelom je správa najdôležitejších bankových operácií. Keďže sa jedná o získavanie dát z viacero zdrojov, kde v budúcnosti je možné zakomponovanie ďalších zdrojov pre rozšírenie funkcionality, je potrebné navrhnúť systém s vysokým stupňom abstrakcie a programovacím jazykom so silnou statickou typovou kontrolou. Pre urýchlenie načítavania a z dôvodu nefrekventovanej modifikácie dát je žiaduce použiť kešovanie v podobe key-value databázy, kde kľúčom je dátum, ku ktorému je potrebné získať platnú verziu cenníka. Pri vkladaní hodnôt do databázy je vhodné určiť exspiračnú dobu záznamu pre efektívne využitie priestoru. Keďže je načastejšou operáciou získanie cenníka pre aktuálny dátum, systém bude automaticky pomocou naplánovanej úlohy každé ráno aktualizovať databázový záznam pre aktuálny dátum.

Skladbu cenníka zloženú z hierarchického usporiadania produktov, ich metadát a cien je potrebné aktualizovať, hoci v menej častých intervaloch. Z toho dôvodu je žiaduce, aby služba taktiež poskytovala možnosť vytvorenia novej verzie cenníka a overila obmedzenia, ktoré sú kladené na ceny, overila logické prepojenie medzi jednotlivými údajmi a uložila novú podobu pre budúci alebo aktuálny dátum. V prípade zmeny cenníka pre rovnaký deň, v ktorom bol aktualizovaný, je kritické pre správne fungovanie premazať keš údajov pre aktuálny dátum.

5.1.2 Služba pre získanie poplatkov klienta

Ďalšou mikroslužbou založenej na biznisových potrebách je služba pre získanie poplatkov klienta. Táto služba je zodpovedná za správu týchto poplatkov, zobrazovanie a synchronizáciu s kľúčovými systémami banky, ktoré sú zodpovedné napríklad za samotné účtovanie poplatkov. Táto služba má veľa spoločných charakteristík so službou pre získavanie cenníka – väčšina operácií je typu čítania, jednotlivé konfigurácie klientov sa modifikujú zriedka a na tejto službe sú okrem cenníkovej služby závislé všetky ostatné komponenty domény.

Podstatným rozdielom oproti službe pre získanie cenníka je typ dát, s ktorými táto služba manipuluje. Zatiaľ čo v druhej službe sa jedná prevažne o statické, jednoducho kešovateľné dáta, táto služba manipuluje s rôznymi klientami, ktorí sa môžu rýchlo striedať pri používaní aplikácie bankérom, a ktorých dáta je potrebné rýchlo získať z databázy a ostatných zdrojových systémov, a spracovať. Je ale predpoklad, že po získaní klientskych údajov bude nutné v krátkom časovom úseku tieto dáta získať opäť. Tomuto je potrebné prispôsobiť návrh systému a databázové kešovacie mechanizmy a indexy.

Táto služba nie je určená pre navrhovanie produktov. Tým je zabezpečené, že do databázy sa nebude často zapisovať a teda používatelia, ktorých cieľom je získanie dát pomocou jedného z komponentov (aplikácia pre prehľad klientských nastavení a mikroslužba pre vygenerovanie reportu klientských nastavení), nebudú postihnutí v prípade zahltenia databázy zápisovými operáciami, ako je to typické v prípade monolitickej architekrúry, a tieto komponenty ostanú plne funkčné. Služba poskytuje REST API pre úpravu klientských nastavení, ktoré je konzumované službou pre navrhovanie nastavení. Služba teda uloží finálny návrh, ktorý je schválený a pripravený na aplikáciu. Služba si následne uloží nové nastavenie a ponechá si taktiež historické nastavenie pre auditovateľnosť a zároveň aplikuje nastavenie do kľúčových bankových systémov v momente, kedy má nové nastavenie vojsť do platnosti. Takéto architektonické riešenie teda oddeľuje čítacie a zapisovacie operácie podobne ako to rieši vzor CQRS (pozri časť 3.3.3) s tým rozdielom, že databáza určená primárne na čítanie nie je len databázovým pohľadom, ale plnohodnotnou databázou, v ktorej sú uložené kľúčové dáta, kde modifikácia prebieha na základe volania API druhou službou. Závislosti konkrétnych komponentov na mikroslužnách sú zobrazené v tabuľke 5.1.

Táto mikroslužba je navrhnutá pre rôzne prípady použitia, a preto musí byť schopná produkovať odpovede s rôznou granularitou. Služba musí byť schopná vrátiť nastavenie poplatkov pre jeden konkrétny účet, čo je typické pre webové zobrazenie; nastavenie poplatkov na všetkých účtoch jedného klienta až po nastavenie poplatkov všetkých účtov viacerých klientov. HTTP odpovede teda môžu obsahovať jednotky až tisícky záznamov. K tomuto je nutné prispôsobiť štruktúru odpovede, kde je nutné mať možnosť vraciať záznamy po stránkach. Pri vykonávaní náročných požiadaviek je nutné mať možnosť použiť asynchrónne REST API[1], aby nenastalo prerušenie spojenia a zbytočne sa nevyužívali prostriedky. Z dôvodu častých vstupno-výstupných operácií je žiaduce pre túto mikroslužbu použitie technológie, ktorá poskytuje asynchrónne neblokujúce spracovanie operácií. Keďže sa rovnako vyžaduje silná statická typová kontrola, vhodným riešením je napríklad využitie Kotlin korutín. Keďže môže dôjsť k náhlej zmene využitia zdrojov inštancií nahor alebo nadol, je správna škálovateľnosť pre efektívne využitie zdrojov kľúčová pre túto mikroslužbu.

5.1.3 Služba pre navrhovanie nastavení klienta

V predchádzajúcich častiach sú popísané mikroslužby, ktorých primárnym účelom je získavanie údajov. V tejto časti je navrhnutá služba, ktorá slúži pre samotnú obsluhu navrhovania nových nastavení pre klientov. Cieľom tejto mikroslužby je teda obsluha žiadosti a komunikácia s mikroslužbou pre správu žiadostí (`WorkflowService`, pozri obr. 5.1) zodpovedajúcou za prechody medzi žiadosťami a komunikujúcou s aplikáciou pre správu žiadostí. Komunikácia prebiaha na základe správ, ktoré pomocou sprostredkovateľa správ rozosiela zdieľaná mikroslužba ostatným komponentom, ktoré sú prihlásené k odberu daných správ. Navrhnutá služba môže taktiež meniť stav alebo údaje žiadosti pomocou volania REST API.

Mikroslužba je navrhnutá tak, aby navrhovanie nových nastavení neovplyvnilo ostatné funkčnosti aplikácie. Z toho dôvodu je iniciátorom aplikovania novej zmeny táto služba, vďaka čomu je zachovaný princíp informačného experta, keďže táto služba konzumuje udalosť, ktorá informuje

[1] Asynchrónne REST API je typ webového rozhrania, ktoré umožňuje klientom odosielať požiadavky a pokračovať vo svojej činnosti, zatiaľ čo server spracováva tieto požiadavky na pozadí a vracia výsledky neskôr, často pomocou spätných volaní alebo iných mechanizmov na asynchrónnu komunikáciu.

o nutnosti aplikovania novej zmeny (`FSM.workflow_state_changed`, pozri obr. 5.1) a zároveň má táto služba prístup k návrhu, ktorý má vojsť do platnosti. Zároveň je týmto prístupom zabránená možnosť vzniknutia cyklickej závislosti, keďže táto služba potrebuje pre výpočet poplatkov a kontroly správnosti návrhu prístup k aktuálnemu nastaveniu klienta zo služby pre získanie poplatkov klienta.

Táto služba musí byť schopná spravovať návrh žiadosti, čo zahŕňa rýchle načítavanie, parciálne ukladanie a kopírovanie údajov. Aplikačná logika služby by mala zaistiť kontrolu správnosti návrhu, a to najmä s ohľadom na ukladanie cien v správnom formáte, kontrolu limitov a obmedzení týkajúce sa daných poplatkov. Keďže sa jedná o pracovný návrh nastavení, ktorý je určený jedine pre účel navrhovania, vysoká normalizovanosť perzistentných dát nie je potrebná a duplicita údajov je akceptovateľná a žiaduca kvôli auditovateľnosti zmien, kde by sa mal návrh duplikovať pri prechode do nového stavu alebo prevzatí žiadosti novým používateľom. Z týchto dôvodov bola ako najvhodnejší typ databázy pre tento účel vybraná dokumentová databáza. Údaje do tejto databázy sú uložené v prakticky nezmenej podobe oproti tomu, ako sú ponúkané ostatným aplikáciám, vďaka čomu je umožnené rýchle načítavanie a ukladanie údajov. Pri výbere tejto databázy je kritické zvoliť správnu granularitu a určiť, čo bude reprezentovať jeden databázový dokument. V prípade, že by dokument obsahoval veľké množstvo údajov, ako by sa to mohlo stať napríklad pri klientoch s veľkým množstvom účtov, bude spomalené načítavanie na všetkých vrstvách aplikácie a bude zbytočne konzumované vysoké množstvo prostriedkov. Preto je vhodné ako jednotku dokumentu považovať napríklad návrh nastavení pre jeden účet, v rámci ktorého sú navrhnuté poplatky len pre konkrétny účet. Pre efektívne ukladanie zmien vykonaných pri modifikácii návrhu je žiaduce parciálne aktualizovanie návrhu z klientskej aplikácie. Preto je vhodné pri implementácii REST API použiť okrem typických metód POST, PUT a DELETE pre manipuláciu s celým dokumentom aj metódu PATCH, v ktorej tele správy bude informácia o konkrétnom poplatku, ktorý sa má modifikovať a cesta, ako je k nemu možné v rámci dokumentu pristúpiť.

5.1.4 Služba pre generovanie reportu nastavení klienta

Zatiaľ čo sú webové aplikácie určené pre bankérov, je potrebné poskytnúť klientovi prehľad jeho nastavení vo forme, s ktorou môže ďalej manipulovať. Z toho dôvodu je vytvorená služba pre generovanie reportu vo formáte XLSL. Táto mikroslužba má teda podobné závislosti ako aplikácie pre prehľad klientských nastavení. Oproti webovej aplikácii musí byť táto mikroslužba schopná spracovať viacero účtov na základe jednej požiadavky. Preto sú požiadavky opäť spracované pomocou asynchrónneho REST API, kde počas doby čakania vracia webovej aplikácii progres generovania reportu.

Mikroslužba má vysokú väzbu na službu pre získanie poplatkov klienta. Okrem toho je generovanie reportu náročné na operačnú pamäť. Z toho dôvodu by bolo možné uvažovať o zlúčení tejto služby so spomínanou službou, ale z biznisového hľadiska, rovnako ako z dôvodu rozšíriteľnosti oboch služieb a zachovania princípu jedinej zodpovednosti mikroslužby má zmysel tieto funkčnosti spravovať v oddelených službách. Pre zachovanie stability mikroslužby a nezahltenia siete je však nutné správne nakonfigurovať stránkovanie pri získavaní informácií a horizontálnu škálovateľnosť mikroslužby s vysokou rezervou operačnej pamäte pri zakladaní novej inštancie, aby nedošlo k vyčerpaniu prostriedkov pôvodnej inštancie.

5.1.5 Služba pre odporúčanie produktov

Cieľom tejto služby je realizácia odporúčacieho systému navrhnutého v kapitole 4. Implementácia tejto služby je ďalej popísaná v kapitole 6. Z biznisového hľadiska tvorí táto služba samostatnú funkčnú jednotku a má teda zmysel, aby bola štruktúrovaná ako samostatná entita. Okrem toho je pre túto funkcionalitu vhodné použiť iné technologické riešenia oproti ostatným službám. Zatiaľ čo je v ostatných službách vhodné brať ohľad na typovú kontrolu, správne ošetrovanie chýb,

prácu so stabilnými a podporovanými aplikačnými rámcami a knižnicami, táto služba vyžaduje riešenia pre efektívny vývoj s jednoducho čitateľným zdrojovým kódom, s vysokou podporou knižníc určených pre strojové učenie a matematické výpočty.

Z používateľského hľadiska je potrebné, aby odporúčanie produktov, ktoré môže byť časovo náročné, neovplyvnilo chod aplikácie a jej čakaciu dobu pre načítavanie. Preto je vhodné riešenie použitie asynchrónneho prístupu získavania odporúčaní pomocou sprostredkovateľov správ. Mikroslužba konzumuje správy informujúce o používateľovi a o účtoch, pre ktoré je potrebné vystaviť odporúčania. Po vygenerovaní odporúčaní mikroslužba produkuje správu, ktorú konzumuje služba pre návrh nastavení klienta, ktorá návrh uloží do dokumentovej databázy, z ktorej je následne možné získať dané odporúčania vo webovej aplikácii.

Kapitola 6

Implementácia systému pre odporúčanie

V tejto kapitole sú najdôležitejšie časti implementácia odporúčacieho systému medzi ostatné bankové komponenty. V prvej časti je popísaná implementácie mikroslužby pre odporúčanie a podporným procesom. Ďalej sú popísané ostatné mikroslužby navrhnuté v rámci architektonického návrhu, potrebné pre podporu vytvárania odporúčania. Záver kapitoly je venovaný vyhodnoteniu naimplementovaného riešenia.

6.1 Mikroslužba pre odporúčanie

Táto časť je venovaná implementácii mikroslužby pre odporúčanie bankových produktov klientom. Na základe častí 4.1 a 5.1 je známe, že táto služba konzumuje cenník produktov, aktuálne nastavenie klientských poplatkov, spracováva dáta z bankového analytického úložiska a spracováva klientske požiadavky na základe prijímania a odosielania správ pomocou sprostredkovateľa správ. Keďže je príprava dát výpočetne náročný proces, ktorý trvá niekoľko desiatok minút, je vhodné, aby táto príprava bola vykonávaná v čase najnižšieho zaťaženia ostatných systémov a aby bola vykonávaná v samostatnej inštancii pre efektívne využitie prostriedkov a správne nastavenie škálovania. Z toho dôvodu je funkcionalita rozdelená na skript slúžiaci na prípravu dát spúšťaný v rámci naplánovanej úlohy (angl. Cron Job) a na samotnú službu určenú na spracovanie používateľských žiadostí.

6.1.1 Naplánovaná úloha

Obsah skriptu slúžiaci na získanie, filtrovanie a spracovanie dát potrebných pre odporúčanie je identický s krokmi popísanými v časti 4.1 pri príprave údajov pre odporúčací algoritmus. Skript je napísaný v jazyku Python s použitím knižníc pre získavanie, manipuláciu a analýzu dát ako Pandas, a knižníc pre matematické výpočty ako NumPy a SciPy. Tento skript je spúšťaný periodicky každý deň v rámci Kubernetess úlohy typu CronJob. Ukážková konfigurácia je znázornená vo výpise kódu 6.1.

Výsledkom použitého skriptu je dátová štruktúra DataFrame, ktorú je možné použiť v následných výpočtoch. Výsledok je teda nutné sprostredkovať službe pre odporúčanie. Keďže knižnica Pandas poskytuje rozhranie pre uloženie dátovej štruktúry do súboru a keďže sa jedná o jedinú a celistvú štruktúru, v ktorej sa nachádzajú všetky potrebné údaje vygenerované skriptom, ako ideálna forma uloženia bol zvolený textový súbor vo formáte CSV. Tento súbor sa ukladá

do inštancie Kubernetess trvalého úložiska, ku ktorému môže následne pristupovať mikroslužba v čítacom režime.

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: recommender-preprocessing-cronjob
spec:
  schedule: "0 0 * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: recommender-preprocessing
            image: recommender-preprocessing-docker-image:1.0
            volumeMounts:
            - name: recommendation-data
              mountPath: /recommendation-data
              readOnly: false
          volumes:
            - name: recommendation-data
              persistentVolumeClaim:
                claimName: recommendation-data-pvc
          restartPolicy: OnFailure
```

Výpis kódu 6.1 Konfigurácia Kubernetess CronJob pre spracovanie dát

6.1.2 Služba

Cieľom služby pre odporúčanie produktov je konzumovanie správ zo sprostredkovateľa, exekúcia odporúčacieho algoritmu a následné vyprodukovanie správy s vygenerovanými odporúčaniami. Ako sprostredkovateľ správ je použitý nástroj Kafka pre jeho prepoužitie, keďže je používaný v celom prostredí. Okrem toho musí služba byť schopná načítať súbor s pripravenými dátami v rámci naplánovanej úlohy, odosielať logy aplikácie pomocou platformy Kafka do analytického nástroja pre dohľadateľnosť chýb a možnosť ich opravy, a musí byť schopná poskytnúť stav aplikácie z dôvodu kontroly zdravia aplikácie platformou pomocou HTTP koncového bodu. Služba teda musí byť schopná efektívne zvládnuť veľké množstvo rozličných vstupno-výstupných operácií.

Jedným z možných riešení je použitie jedného z webových aplikačných rámcov pre jazyk Python a spustenie vlastného vlákna pri štarte aplikácie, ktoré by slúžilo pre konzumovanie Kafka správ a ich spracovanie počas celej doby behu aplikácie. Toto riešenie sa javí ako neefektívne kvôli vysokému počtu vstupno-výstupných operácií a prepínaniu kontextu vlákien medzi konzumovaním HTTP a Kafka správ. Okrem toho je očakávané, že v čase plného vyťaženia služby budú predstavovať HTTP požiadavky minimum záťaže. Z toho dôvodu je pre lepšiu škálovateľnosť a využitie zdrojov aplikácie vhodné využiť asynchrónne spracovanie operácií, kde pre dosiahnutie súbežnej exekúcie viacero úloh je možné využiť Python korutín a modelu event-loop. Tento prístup by mal byť obzvlášť efektívny pri využívaní vyššieho počtu vstupno-výstupných spojení.

V jazyku Python je pre asynchrónne operácie určená knižnica asyncio. Pre asynchrónnu implementáciu Kafka klienta je využitá knižnica aiokafka a pre asynchrónne logovanie je analogicky použitá knižnica aiologger. Ďalej je potrebné zvoliť vhodný webový aplikačný rámec pre tento prípad použitia. Ako ideálny aplikačný rámec je vhodný v tomto prípade FastAPI, ktorý pre svoj beh využíva ASGI[1] implementáciu webového servera, ktorá je navrhnutá na asynchrónnom princípe spracovania žiadostí. V tomto prípade bola zvolená Uvicorn ASGI implementácia webového

[1] Asynchronous Server Gateway Interface

servera. Okrem toho FastAPI implicitne spracuváva neošetrené chybové výstupy v JSON formáte namiesto HTML stránky, čo je v riešení architektúry mikroslužieb žiaduce. Jednou z nevýhod je to, že tento aplikačný rámec je relatívne nový a nemá takú vysokú komunitu ako ostatné nástroje, čo sa môže prejaviť vo vyššom výskyte chýb a nižšou podporou nástroja, čo sa prejavilo aj pri tejto implementácii.

Pre spustenie novej úlohy na pozadí je teda potrebné pridať novú korutinu do slučky udalostí pomocou zavolania metódy `create_task` knižnice asyncio. Vo výpise kódu 6.2 je zobrazené samotné konzumovanie Kafka správ a následné spracovanie. Inštancie tejto mikroslužby majú pridelenú vlastnú tzv. skupinu konzumentov, aby jedna správa nemohla byť konzumovaná viacero inštanciami zároveň. Ako vstup je očakávaná hodnoda v JSON formáte, ktorý obsahuje položku `user_id` pre identifikáciu klienta. Pre vygenerovanie odporúčaní je vytvorená abstraktná trieda `Algorithm` analogicky podľa modelu na obrázku 4.1, ktorá je pripravená na ďalšie rozšírenie a aplikovanie konkrétnej implementácie je riadené pomocou premenných prostredia pre čo najväčšiu konfigurovateľnosť a flexibilnosť. Z rovnakého dôvodu je možné nastaviť pomocou premenných prostredia atribúty jednotlivých odporúčacích algoritmov. Načítavanie súboru s dátami pripraveného v rámci naplánovanej úlohy prebieha synchrónne, keďže sa jedná o kritický súbor pre fungovanie služby a ostatné úkony bez tohto súboru nemajú opodstatnenie. Keďže sa ale tento súbor modifikuje jedenkrát denne, pre efektívne využitie prostriedkov je tento súbor v programe kešovaný. Výsledok odporúčania je odoslaný v tele žiadosti vo formáte JSON s ID produktami a ich odporúčaniami.

```
try:
    await self.recommendation_request_consumer.start()
    await self.producer.start()
except Exception as e:
    await self.logger.critical(...)
    self.health_status.set_unhealth_status()
    return
try:
    async for message in self.recommendation_request_consumer:
        ...
        try:
            recommendations = await (self.recommender
                .generate_recommendations(user_id))
        except Exception as e:
            await self.logger.error(...)
            continue
        output = {
            "user_id": user_id,
            "recommendations": {
                recommendation.fee_id: bool(recommendation.recommended)
                    for recommendation in recommendations.recommendations
            }
        }
        ...
        await self.producer.send(self.output_topic_name, output)
...
finally:
    await self.recommendation_request_consumer.stop()
    await self.producer.stop()
```

Výpis kódu 6.2 Ukážka asynchrónneho konzumovania Kafka správ, následného spracovania odporúčania a odoslania odpovede

Funkčnosti boli rozdelené do jednotlivých tried podľa ich zodpovednosti a pre získanie závislostí tried bol použitý vzor vkladania závislostí pomocou knižnice dependency-injector. Pre testy

funkcionalít boli použité knižnice pytest a pytest-asyncio. Po otestovaní a overení všetkých funkčností je mikroslužba plne funkčná a pripravená na produkčné využitie.

Summary

9 tests took 92 ms.

(Un)check the boxes to filter the results.

0 Failed, 9 Passed, 0 Skipped, 0 Expected failures, 0 Unexpected passes, 0 Errors, 0 Reruns Show all details / Hide all details

Result	Test	Duration	Links
Passed	src/tests/data_loader_test.py::test_load_recommender_data_dataset	19 ms	
Passed	src/tests/data_loader_test.py::test_load_recommender_data_from_cache	6 ms	
Passed	src/tests/data_loader_test.py::test_load_recommender_data_cache_expired	8 ms	
Passed	src/tests/health_status_test.py::test_initial_health_status	1 ms	
Passed	src/tests/health_status_test.py::test_set_unhealth_status	1 ms	
Passed	src/tests/kafka_logging_handler_test.py::test_emit_start_producer_once	11 ms	
Passed	src/tests/kafka_recommendations_handler_test.py::test_start	21 ms	
Passed	src/tests/recommender_test.py::test_generate_recommendations_success	12 ms	
Passed	src/tests/recommender_test.py::test_generate_recommendations_user_not_found	12 ms	

Obr. 6.1 Report výsledkov testovania pomocou nástroja `pytest-report`

6.2 Ostatné mikroslužby

Táto časť je venovaná implementácii ostatných mikroslužieb potrebných pre správny chod systému a samotnej služby pre odporúčanie. Implementácia týchto mikroslužieb je riešená pomocou aplikačného rámca Spring Boot s použitím jazyka Kotlin pre zabezpečenie dlhodobej podpory a silnej statickej typovej kontroly s použitím moderných konštruktov. Okrem toho sú tieto technológie výrazne používané v bankovom prostredí, čo umožňuje prepoužitie spoločných funkčností. Pre automatizáciu zostavenia aplikácií je použitý nástroj Maven a pre kontinuálnu integráciu a nasadenie je použitý nástroj Jenkins.

Služba pre získanie cenníka Táto služba je zodpovedná za správu a sprostredkovanie cenníka klientom a ostatným aplikáciám. Vzhľadom k tomu, že je táto služba verejne prístupná, je potrebné zabezpečiť autorizáciu používateľov a sprístupniť kontext a funkcionality na základe používateľských rolí, čo bolo implementované pomocou knižnice Spring Security a jej anotácie `@Secured`. Pre ukladanie cenníka bola použitá relačná databáza PostgreSQL spolu s knižnicou Flyway pre databázové migrácie. Pre kešovanie spracovaných cenníkov v rámci aplikačnej vrstvy bola použitá databáza Redis. Pre zabezpečenie najnovších dát a rýchlosti načítavania sú údaje v databáze každodenne aktualizované pomocou napánovaných úloh. Služba poskytuje nasledujúce koncové body:

```
GET   /pricelist?effectiveDate={effectiveData}&forced={forced}
PUT   /pricelist?validFrom={validFrom}
```

Tieto koncové body slúžia na získanie alebo pridanie cenníka s možnosťou nastavenia dátumu platnosti dát. Na základe funkčných požiadaviek je možné prepísať verziu cenníka pre konkrétny dátum, kde je stále platná najnovšie nahraná verzia. Z toho dôvodu je pre zachovanie idempotencie použitá HTTP metóda PUT namiesto typicky používanej metódy POST pre podobné prípady.

```
@Scheduled(cron = "00␣00␣00␣*␣*␣*")
@Cacheable(value = [PRICE_LIST_CACHE], sync = true)
fun getActualPriceList(): PriceList {
    return getPriceList(LocalDate.now())
}
```

Výpis kódu 6.3 Ukážka aktualizovania kešovaných dát pomocou naplánovanej úlohy

Služba pre získanie poplatkov klienta Táto služba podobne ako služba pre získanie cenníka využíva relačnú databázu PostgreSQL pre ukladanie klientských poplatkov. Tieto údaje nemôžu byť dlhodobo kešované pre potrebu získania aktuálnych dát, a preto je nutné využitie paralelného získavania údajov z interných systémov pre rýchle spracovanie pomocou Kotlin korutín. Hlavné koncové body tejto služby sú:

```
GET    /charges/{clientId}
POST   /charges/{clientId}
```

Na rozdiel od ukladania novej verzie cenníka vytvára v tomto prípade HTTP volanie nový záznam o poplatkovaní klienta, a teda táto operácia nie je idempotentná.

Služba pre navrhovanie nastavení klienta V tejto službe bola použitá dokumentová databáza pomocou nástroja ElasticSearch slúžiaceho okrem iného na distribuované textové vyhľadávanie a analýzu dát. Služba musí primárne poskytovať spoľahlivé a rýchle úložisko žiadostí ukladané vo formáte JSON, k čomu slúžia nasledujúce koncové body:

```
GET      /workflows/{workflowId}/designs
POST     /workflows/{workflowId}/designs
GET      /designs/{designId}
PUT      /designs/{designId}
PATCH    /designs/{designId}?accountNumber={accountNumber}&feeId={feeId}
DELETE   /designs/{designId}
```

V rámci jednej žiadosti môže existovať niekoľko návrhov, ktorých identifikácie vrátane aktuálneho alebo historického návrhu v rámci konktétnej etapy žiadosti je možné získať pomocou prvého definovaného koncového bodu, a to za pomoci princípu HATEOAS, vďaka ktorému získa klient odkazy na konkrétne návrhy. Keďže táto služba ako jediná poskytuje rozhranie, na základe ktorého je možné modifikovať údaje, bol kladený dôrad na to, aby konkrétny návrh mohol modifikovať len privilégovaný používateľ. Pomocou metódy PATCH je možné modifikovať špecifickú časť žiadosti, a preto je nutné uviesť cestu k časti dokumentu, ktorý je žiaduce modifikovať, a to pomocou URI parametra pre číslo účtu alebo parametra pre modifikáciu konkrétneho poplatku. Táto mikroslužba taktiež automaticky odosiela a prijíma Kafka správy pre komunikáciu so službou pre odporúčanie produktov. V prípade, že služba prijme správu s odporúčaním, toto odporúčanie automaticky vloží do prináležitého dokumentu žiadosti.

```
val content = objectMapper.writeValueAsString(design)
val request = Request("PUT", "/${designIndex.lowercase()}/_doc/$id")
request.entity = NStringEntity(content, ContentType.APPLICATION_JSON)
elasticRestClient.performRequest(request)
```

Výpis kódu 6.4 Ukážka aktualizovania dokumentu návrhu pomocou knižnice ElasticSearch Java REST Client

Služba pre generovanie reportu nastavení klienta Cieľom tejto služby je generovanie súborov vo formáte XLSL. K tomuto účelu bola použitá známa Apache knižnica POI poskytujúca API pre prístup k súborom Microsoft formátov. Generovanie takéhoto súboru môže byť

časovo náročné, a preto bola použitá implementácia jednoduchého asynchrónneho API:

```
POST   /reports
GET    /reports/{jobId}
```

Klient získa po zavolaní metódy POST identifikáciu založeného procesu, pomocou krorej sa môže dopytovať na stav generovania. V prípade, že súbor nebol vygenerovaný, bude vrátený HTTP status `202 Accepted`. V prípade, že bol súbor vygenerovaný, bude vrátená HTTP odpoveď s adekvátným nastaveným typom obsahu tela správy. K tomu, aby prehliadač bol schopný správne nazvať stirhnutý súbor, bola priložená HTTP hlavička `Content-Disposition` so zakódovaným názvom súboru.

```
val encodedFilename =
    URLEncoder.encode(filename, "UTF-8").replace("+", "%20")
val headers = HttpHeaders()
headers["Content-Disposition"] =
    "attachment;filename*=utf-8''$encodedFilename"
headers.contentType = MediaType(
    type = "application",
    subtype = "vnd.openxmlformats-officedocument.spreadsheetml.sheet")
return ResponseEntity(file, headers, HttpStatus.OK)
```

Výpis kódu 6.5 Konfigurácia HTTP hlavičiek pre vrátenie vygenerovaného XLSL súboru. Názov bol zakódovaný pomocou URLEncoder, ale keďže je medzera v ceste URI zakódovaná pomocou symbolu `+`, musí byť tento symbol manuálne nahradený.

6.3 Vyhodnotenie

Po tom, čo bol implementovaný popísaný návrh, uplynulo isté časové obdobie, na základe čoho je možné vyhodnotiť fungovanie systému v produkčnom prostredí. Hoci má služba pre získanie cenníka vysokú previazanosť s ostatnými komponentami, jej efektívnosť a spoľahlivosť dosahuje významnú úroveň, a to vďaka jej funkčnosti iba na čítanie a vhodného použitia keše. Na druhej strane, táto služba má problémy so zdrojmi a výkonom v momente, kedy je potrebné nahrať novú verziu cenníka. Táto operácia je výkonnostne náročná a typicky počas nahrávania je vytvorená nová inštancia v rámci automatického horizontálneho škálovania. To je spôsobené tým, že sa zdroje inštancie výrazne vychýlia od bežného priemeru spotreby. Riešením tohto problému by bolo napríklad použitie Kubernetess Job, kde sa pre nahranie novej verzie použije vlastný zdroj, ktorý sa po úspešnom nahraní do databázy automaticky odstráni. Pre mikroslužbu slúžiacu na získanie poplatkov klienta je dôležitá vlastnosť škálovania, keďže výkyvy v počte súbežných požiadaviek v čase sú zjavné. Systém je výrazne stabilnejší vďaka oddeleniu čítacích a zapisovacích operácií do dvoch mikroslužieb (t.j. získanie poplatkov a ich navrhovanie). V prípade, že služba pre navrhovanie klientských nastavení je zaťažená zvýšeným počtom zapisovacích operácii alebo je výrazná latencia pri komunikácii so službou obsluhujúcou procesy, zvyšné funkcionality systému to neovplyvní. Pre službu pre generovanie klientských reportov platí, že vzhľadom na zvýšenú spotrebu operačnej pamäte a sieťových prvkov je dôležité efektívne využitie týchto zdrojov. K tomu, aby nedošlo k zahlteniu a pádu inštancie je potrebné správne určenie veľkosti toku dát a konfigurácia JVM parametrov. Keďže je typicky očakávaná zväčšujúca sa spotreba prostriedkov počas toho, ako sa generuje obsiahly report, je potrebné včas vytvoriť novú inštanciu, aby mala pôvodná inštancia dostatok prostriedkov na dokončenie operácie.

Táto architektúra nepopierateľne zvyšuje čitateľnosť, oddelenosť a rozšíriteľosť zdrojového kódu a lepšie organizovanie výpočetných prostriedkov. Znižuje počiatočnú krivku učenia a je možné lepšie odprezentovať a vysvetliť fungovanie systému ostatným členom tímu. Na druhej strane, údržba kódu, komunikácia medzi mikroslužbami, ošetrovanie chybových stavov a efektívne využitie zdrojov, rovnako ako správne oddelenie zodpovednosti medzi mikroslužbami predstavuje

výzvu aj pre skúsenejších odborníkov a nesprávny návrh môže predstavovať výrazné problémy pre stabilitu systému a efektívne vynaloženie zdrojov, a refaktoring takejto architektúry môže byť náročný a nákladný proces. Ďalej stojí za zmienku porovnanie celkových použitých zdrojov na server oproti monolitickej aplikácii. Väčšina funkčností predstavených v rámci tejto domény slúži pre bankérov v rámci interných procesov, čo nepredstavuje vysoký počet používateľov v porovnaní s inými aplikáciami, ako je napríklad Internet Banking. Okrem toho spustenie jednej mikroslužby, ktorá je navrhnutá v technológiách JVM a Spring Boot typicky vyžaduje vysoké množstvo operačnej pamäte pre beh aplikácie a štart týchto systémov je v porovnaní s ostatnými technológiami relatívne pomalý. Dá sa teda predpokladať, že zdroje použité pre chod jednej monolitickej aplikácie by v tomto prípade mohli byť nižšie oproti desiatke inštancií mikroslužieb.

Kapitola 7
Záver

Cieľom tejto práce bol návrh a implementácia odporúčacieho systému zameraného na firemných bankových klientov a jeho začlenenie do bankového prostredia medzi ostatné systémy. Pre dosiahnutie tohto cieľa bola analyzovaná banková doména a s tým súvisiace dáta, procesy a boli zanalyzované požiadavky na aplikáciu pre navrhovanie a odporúčanie nastavení. Na základe týchto poznatkov bola vykonaná rešerš nad problematikou odporúčacích systémov a boli vybrané algoritmy vhodné pre riešenie daného problému. Okrem rešerše nad odporúčacími systémamy bola taktiež naštudovaná problematika architektúry mikroslužieb pre kvalitné návrhnutie a začlenenie systému do prostredia.

V praktickej časti boli vykonané experimenty nad vybranými algoritmami, kde bola testovaná konfigurácia algoritmov a ich celková úspešnosť naprieč všetkými vykonanými experimentami. Spomedzi všetkých meraní dosahoval najlepšie výsledky hybridný model na základe zhlukovania použitého na analytické dáta klientov a používateľsky založeného kolaboratívneho filtrovania aplikovaného na poplatky. Ďalej bola navrhnutá architektúra mikroslužieb, ktorá bola úspešne zakomponovaná do bankového prostredia medzi ostatné systémy. V rámci návrhu bolo vytvorených päť nových mikroslužieb na základe biznisových potrieb a ich funkčností. Na záver bol naimplementovaný systém pre odporúčanie na základe asynchrónneho princípu, sprostredkovateľa správ a webového aplikačného rámca pre jazyk Python.

V celom návrhu a implemetácii bol kladený dôraz na budúcu rozšíriteľnosť riešenia. Pred vykonaním experimentov bol vytvorený model s očakávaným jednotným výstupom a rozhranie pre exekúciu a testovanie. Mikroslužby boli navrhnuté s očakávaním budúceho rozšírenia a prepojenia s ďalšími bankovými systémami. Implementácia odporúčacieho systému je navrhnutá s možnosťou pridania alebo úpravy ďalších odporúčacích techník bez nutnosti zmeny ostatných častí kódu a pre dosiahnutie vyššej flexibilnosti je možné meniť konfiguráciu algoritmov pomocou premenných prostredia.

Bibliografia

1. ČESKÁ SPOŘITELNA, A. S. (ed.). *Ceník pro korporátní klientelu* [online]. [B.r.]. [cit. 2023-10-21]. Dostupné z : `https://www.csas.cz/static_internet/cs/Obchodni_informace-Produkty/Sazebnik/Komercni_klientela/Prilohy/cenik_korporatni_klientela_cz-od-192023_podepsany.pdf`.
2. RAIFFEISENBANK A. S. (ed.). *Ceník produktů a služeb pro firmy a korporace* [online]. [B.r.]. [cit. 2023-10-21]. Dostupné z : `https://www.rb.cz/attachments/ceniky/cenik-corp-1.pdf`.
3. ČESKOSLOVENSKÁ OBCHODNÍ BANKA, A. S. (ed.). *Sazebník ČSOB pro právnické osoby a pro fyzické osoby – podnikatele* [online]. [B.r.]. [cit. 2023-10-21]. Dostupné z : `https://www.csob.cz/documents/10710/423623/sazebnik-sme-cz.pdf`.
4. *What is a Data Warehouse?* [online]. Google, [b.r.] [cit. 2023-10-21]. Dostupné z : `https://cloud.google.com/learn/what-is-a-data-warehouse`.
5. *Banking DWH Model* [online]. Data Warehouse Models, 2023 [cit. 2023-10-21]. Dostupné z : `https://f.hubspotusercontent20.net/hubfs/8981679/Banking%20DWH%20Model%20e-book.pdf`.
6. ČESKÁ SPOŘITELNA, A. S. (ed.). *Zjistěte, v jaké kondici je vaše podnikání* [online]. [B.r.]. [cit. 2023-10-21]. Dostupné z : `https://www.csas.cz/cs/firmy/page/poradenstvi-pro-firmy`.
7. KRAJÍČEK, JAN. *Marketing v peněžnictví*. Masarykova Univerzita v Brně, 2005. ISBN 80-210-3659-1.
8. BRICHOVÁ, Jana. *Segmentace klientely vybrané banky a metody jejího získávání a udržení*. 2011. Dostupné tiež z: `https://is.ambis.cz/th/xt28k/`. Diplomová práca. AMBIS vysoká škola, a.s. Vedúci práce Denis BIEDERMANN.
9. ČESKÁ SPOŘITELNA, A. S. *George*. 2013. 23.34.17-google. Dostupné tiež z: `https://www.csas.cz/cs/internetove-bankovnictvi/george`. Dostupné po prihlásení.
10. RAIFFEISENBANK. *Raiffeisenbank*. 2023. 231003144-5.6.0. Dostupné tiež z: `https://online.rb.cz/`. Dostupné po prihlásení.
11. RICCI, Francesco; ROKACH, Lior; SHAPIRA, Bracha. Introduction to recommender systems handbook. In: *Recommender systems handbook*. Springer, 2010, s. 1–35. ISBN 978-0-387-85819-7.
12. SAMMUT, Claude; WEBB, G. *Encyclopedia of Machine Learning.(2010)*. Springer-Verlag New York Incorporated, [b.r.]. ISBN 978-0-387-30768-8.
13. RESNICK, Paul; VARIAN, Hal R. Recommender systems. *Communications of the ACM*. 1997, roč. 40, č. 3, s. 56–58.

14. PATEL, Bansari; DESAI, Palak; PANCHAL, Urvi. Methods of recommender system: A review. In: *2017 international conference on innovations in information, embedded and communication systems (ICIIECS)*. IEEE, 2017, s. 1–4.

15. ISINKAYE, Folasade Olubusola; FOLAJIMI, Yetunde O; OJOKOH, Bolande Adefowoke. Recommendation systems: Principles, methods and evaluation. *Egyptian informatics journal*. 2015, roč. 16, č. 3, s. 261–273.

16. BREESE, John S; HECKERMAN, David; KADIE, Carl. Empirical analysis of predictive algorithms for collaborative filtering. *arXiv preprint arXiv:1301.7363*. 2013.

17. ADOMAVICIUS, Gediminas; ZHANG, Jingjing. Impact of data characteristics on recommender systems performance. *ACM Transactions on Management Information Systems (TMIS)*. 2012, roč. 3, č. 1, s. 1–17.

18. FKIH, Fethi. Similarity measures for Collaborative Filtering-based Recommender Systems: Review and experimental comparison. *Journal of King Saud University-Computer and Information Sciences*. 2022, roč. 34, č. 9, s. 7645–7669.

19. GEDIKLI, Fatih. *Recommender systems and the social web: Leveraging tagging data for recommender systems*. Springer Science & Business Media, 2013.

20. SALUJA, Chhavi. *Collaborative filtering based recommendation systems exemplified.*. Towards Data Science, 2018. Dostupné tiež z: `https://towardsdatascience.com/collaborative-filtering-based-recommendation-systems-exemplified-ecbffe1c20b1`.

21. ISINKAYE, F.O.; FOLAJIMI, Y.O.; OJOKOH, B.A. Recommendation systems: Principles, methods and evaluation. *Egyptian Informatics Journal*. 2015, roč. 16, č. 3, s. 261–273. ISSN 1110-8665. Dostupné z DOI: `https://doi.org/10.1016/j.eij.2015.06.005`.

22. BISHOP, Christopher M; NASRABADI, Nasser M. *Pattern recognition and machine learning*. Zv. 4. Springer, 2006. Č. 4.

23. SCHÜTZE, Hinrich; MANNING, Christopher D; RAGHAVAN, Prabhakar. *Introduction to information retrieval*. Zv. 39. Cambridge University Press Cambridge, 2008.

24. EKSTRAND, Michael D; RIEDL, John T; KONSTAN, Joseph A et al. Collaborative filtering recommender systems. *Foundations and Trends® in Human–Computer Interaction*. 2011, roč. 4, č. 2, s. 81–173.

25. JANNACH, Dietmar; ZANKER, Markus; FELFERNIG, Alexander; FRIEDRICH, Gerhard. *Recommender systems: an introduction*. Cambridge University Press, 2010.

26. CROFT, W Bruce; METZLER, Donald; STROHMAN, Trevor. *Search engines: Information retrieval in practice*. Zv. 520. Addison-Wesley Reading, 2010.

27. CHIRE. *Cluster analysis with k-Means on a gaussian-distribution-based data set*. 2011. Dostupné tiež z: `https://commons.wikimedia.org/wiki/File:KMeans-Gaussian-data.svg`.

28. SAKKAF, Yaser. *Decision trees for classification: Id3 Algorithm explained*. Towards Data Science, 2020. Dostupné tiež z: `https://towardsdatascience.com/decision-trees-for-classification-id3-algorithm-explained-89df76e72df1`.

29. NILASHI, Mehrbakhsh; BAGHERIFARD, Karamollah; IBRAHIM, Assoc Prof. Dr. Othman; ALIZADEH, Hamid; LASISI, Ayodele; ROOZEGAR, Nazanin. Collaborative Filtering Recommender Systems. *Research Journal of Applied Sciences, Engineering and Technology*. 2013, roč. 5, s. 4168–4182. Dostupné z DOI: `10.19026/rjaset.5.4644`.

30. *PR curve with optimal fscore*. 2023. Dostupné tiež z: `https://commons.wikimedia.org/wiki/File:PR_curve_with_optimal_fscore.png`.

31. FAYYAD, Usama; PIATETSKY-SHAPIRO, Gregory; SMYTH, Padhraic. The KDD Process for Extracting Useful Knowledge from Volumes of Data. *Commun. ACM*. 1996, roč. 39, č. 11, s. 27–34. ISSN 0001-0782. Dostupné z DOI: `10.1145/240455.240464`.

32. LÜ, Linyuan; MEDO, Matúš; YEUNG, Chi Ho; ZHANG, Yi-Cheng; ZHANG, Zi-Ke; ZHOU, Tao. Recommender systems. *Physics reports*. 2012, roč. 519, č. 1, s. 1–49.
33. GIGLI, Andrea; LILLO, Fabrizio; REGOLI, Daniele. Recommender Systems for Banking and Financial Services. In: *RecSys Posters*. 2017.
34. SHARAF, Marwa; HEMDAN, Ezz El-Din; EL-SAYED, Ayman; EL-BAHNASAWY, Nirmeen A. A survey on recommendation systems for financial services. *Multimedia Tools and Applications*. 2022, roč. 81, č. 12, s. 16761–16781.
35. SUN, Yunchuan; FANG, Mengting; WANG, Xinyu. A novel stock recommendation system using Guba sentiment analysis. *Personal and Ubiquitous Computing*. 2018, roč. 22, s. 575–587.
36. OYEBODE, Oladapo; ORJI, Rita. A hybrid recommender system for product sales in a banking environment. *Journal of Banking and Financial Technology*. 2020, roč. 4, s. 15–25.
37. PAHL, Claus; JAMSHIDI, Pooyan. Microservices: A Systematic Mapping Study. *CLOSER (1)*. 2016, s. 137–146.
38. AL-DEBAGY, Omar; MARTINEK, Peter. A Comparative Review of Microservices and Monolithic Architectures. In: *2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI)*. 2018, s. 000149–000154. Dostupné z DOI: 10.1109/CINTI.2018.8928192.
39. FOWLER, Martin. *Microservices*. 2014. Dostupné tiež z: https://martinfowler.com/articles/microservices.html.
40. PINKAS, Jiří. *Microservices + Event Driven Architecture v tech. firmě*. 2023. JavaDays.
41. BOLBOACA, Adrian. *What is evolutionary design?* 2018. Dostupné tiež z: https://mozaicworks.com/blog/what-is-evolutionary-design.
42. RICHARDSON, Chris. *Microservices pattern: Sagas*. 2023. Dostupné tiež z: https://microservices.io/patterns/data/saga.html.
43. RICHARDSON, Chris. *Microservices pattern: Event sourcing*. 2023. Dostupné tiež z: https://microservices.io/patterns/data/event-sourcing.html.
44. RICHARDSON, Chris. *Microservices pattern: CQRS*. 2023. Dostupné tiež z: https://microservices.io/patterns/data/cqrs.html.
45. RICHARDSON, Chris. *Microservices pattern: Circuit Breaker*. 2023. Dostupné tiež z: https://microservices.io/patterns/reliability/circuit-breaker.html.
46. BUCCHIARONE, Antonio; DRAGONI, Nicola; DUSTDAR, Schahram; LAGO, Patricia; MAZZARA, Manuel; RIVERA, Victor; SADOVYKH, Andrey. Microservices. *Science and Engineering. Springer*. 2020.

Zoznam skratiek

ACID	Atomicity, Consistency, Isolation, Durability
AML	Anti-Money Laundering
API	Application Programming Interface
ASGI	Asynchronous Server Gateway Interface
BFF	Backend For Frontend
BPMN	Business Process Model and Notation
CQRS	Command Query Responsibility Segregation
CRM	Customer Relationship Management
DWH	Data Warehouse
FSM	Finite State Machine
HATEOAS	Hypermedia as the Engine of Application State
HTTP	Hypertext Transfer Protocol
JVM	Java Virtual Machine
KDD	Knowledge Discovery in Databases
MAE	Mean Absolute Error
NMAE	Normalized Mean Absolute Error
REST	Representational State Transfer

Obsah príloh

- `readme.txt` ..... stručný popis obsahu média
- `src`
  - `experiments` ..... vykonané anonymizované merania v platforme Jupyter Notebook
  - `implementation` ..... zdrojové kódy implementácie mikroslužby pre odporúčanie
  - `thesis` ..... zdrojová forma práce vo formáte LaTeX
- `text`
  - `thesis.pdf` ..... text práce ve formáte PDF