Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

Projektová dokumentácia k inžinierskemu dielu

(Tím 03 - SmartStore)

Akademický rok: 2016/2017
Predmet: Tímový projekt

Členovia tímu (študenti): Bc. Matej Červenka
Bc. Mária Dragúňová
Bc. Ondrej Kudláč
Bc. Matúš Salát
Bc. Martin Šidlo
Bc. Lukáš Šimek

Vedúci tímu: Ing. Peter Krátky
Vlastník produktu: Ing. Ján Sukeník

Obsah

1 Úvod

Kúpa rôznych produktov na trhu je každodennou súčasťou nášho života. Prirodzená vlastnosť zákazníkov je, že chcú, aby boli ich potreby uspokojené v čo najkratšom možnom čase. Pri kúpe produktov cez internet alebo priamo v predajni sa niekedy stretávame s aktuálnou nedostupnosťou produktov. Predajcovia si nemôžu dovoliť držať celý svoj sortiment na sklade, respektíve v neobmedzenom množstve. Pri veľmi malom sortimente je možné aby človek vedel sledovať predaje a ak má aj dostatočné skúsenosti, môže odhadovať koľko bude ktorých výrobkov treba. Tento prípad je avšak ojedinelý a vo väčšine prípadov je sortiment natoľko veľký, že nie je v ľudských silách ustrážiť všetky produkty a ich predaje. Preto tu vzniká potreba automatizovať toto vyhodnocovanie. Do úvahy treba brať viaceré faktory. Nie všetky predaje produktov sa môžu vyvíjať rovnako. Niektoré sa môžu správať lineárne a niektoré skokovo. Do úvahy treba brať aj také veci, ako sú ročné obdobia, respektíve rôzne druhy nákupných sezón, počas ktorých sa môžu zákazníci správať odlišne.

Jedným z ďalších problémov je samotná distribúcia, respektíve zabalenie nakúpených produktov pre zákazníka. Skladové priestory sú vo väčšine prípadov veľké. Pri vybavovaní objednávok a zbieraní produktov vzniká prechod po tomto sklade. Plánovanie takejto cesty je pre človeka náročná vec a preto sa môže stať, že si svoju cestu nenaplánuje optimálne. Tento fakt má za následok nižšiu produktivitu práce, ale čo je dôležitejšie, neskoršie doručenie produktov k zákazníkom. Optimálny výber cesty a s ním spojené urýchlenie výberu objednávky by však bolo realizované v sklade, v ktorom sú tovary rozmiestené neusporiadane, preto riešime aj usporiadanie tovaru.

V projekte ponúkame riešenie oboch problémov. Na základe nazbieraných dát z predajov predikujeme kedy a koľko kníh titulu objednať, čo má za následok skrátenie dodacej doby zákazníkom. K rýchlejšej distribúcii objednávok máme vytvorený modul na hľadanie optimálnej cesty v sklade *SmartCollect*, čo urýchli vybavovanie objednávky priamo v sklade. Súčasťou riešenia na urýchlenie na úrovni skladu je aj odporúčanie správnej pozície tovaru v sklade na základe predávanosti tovaru a analýzy nákupného košíka. Pevne veríme, že naše riešenie prispeje k väčšej spokojnosti zákazníkov.

2 Globálne ciele pre ZS

Predikcia, respektíve predpovedanie rôznych typov údajov, je momentálne aktuálna téma. V našom projekte sa zaoberáme konkrétne predikciou predajov produktov na základe ich doterajšieho vývoja. Tieto predaje, ktoré zahrňujú rôzne produkty, sa snažia obchodníci čo najskôr distribuovať. S týmto je spojená dostupnosť produktov na sklade. Nie všetky produkty je možné mať naskladnené, z čoho vyplýva potreba ich častého objednávania a zdržiavania vybavenia objednávok. Pre odstránenie tohto čakania je potrebné vopred vedieť aké a koľko produktov bude potrebné mať na sklade.

V rámci zimného semestra sme si s produktovým majiteľom vytýčili tieto ciele, ktoré by sme chceli stihnúť a to sú:

- spracovať a analyzovať dáta o predajoch,
- vytvoriť model predikcie počtu predajov produktov, dátumu vyčerpania zásob a dátumu objednania ďalších produktov a ich počet,
- automaticky zbierať dáta o predajoch,
- vytvoriť webové rozhranie pre zobrazenie predikcie pre pracovníkov.

Tieto ciele by mali byť naplnené do konca semestra. Distribúcia produktov k zákazníkom nie je spojená len s ich naskladnením, ale aj s vybavovaním jednotlivých objednávok v priestoroch skladu. Všetky položky z objednávky musia byť nájdené a následne zabelené v sklade. Pre zjednodušenie a urýchlenie zberu produktov na sklade je možné vyhľadať optimálnu cestu pre získanie všetkých produktov. Tento problém sa dá taktiež riešiť rôznymi algoritmami. V rámci zimného semestra sme si vytýčili ciele, ktoré sa týkajú aj tejto oblasti a to sú:

- analýza skladu a jeho organizácie,
- preskúmanie možností hľadania optimálnej cesty
- implementácia prototypu na nájdenie optimálnej cesty.

3 Globálne ciele pre LS

V zimnom semestri sa nám podarilo naplniť, ciele, ktoré sme si zadefinovali. Pre letný semester sme preto naplánovali ďalšie zlepšovanie na module *SmartCollect*, tentoraz aj na úrovni umiestnenia tovaru v sklade a sústredili sme sa aj na vyhodnotenie zlepšenia, ktoré implementácia tohto modulu priniesla.

Na zlepšenie modulu *SmartCollect* sme naplánovali tieto ciele:

- Vytvorenie rozhrania, v ktorom budú zobrazené tie produkty, ktoré sú aktuálne najnevhodnejšie umiestené.
- Dátová analýza nákupného košíka (zistenie produktov predávaných spolu)
- Návrh výpočtu popularity produktu (koľko kusov sa predá za poslednú dobu).
- Návrh rozdelenia skladu na sektory podľa popularity produktu.
- Návrh algoritmu na optimálne rozmiestnenie tovaru v sklade, ktorý berie do úvahy popularitu produktu aj produkty predávané s ním.
- Vytvorenie služby na získanie optimálnej pozície produktu (alebo optimálneho sektora v sklade).
- Vizualizácia cesty v sklade a vizualizácia optimálnej pozície produktu.

Na vyhodnotenie modulu *SmartCollect* sme naplánovali tieto ciele:

- Návrh algoritmu na vyhodnotenie počtu nachodených metrov
    - Pri objednávkach s rôznym počtom kusov
    - Pri použití rôznych rozložení skladu
    - Pri uvažovaní rôznych začínajúcich miest v sklade

Keďže súčasťou práce v letnom semestri je aj vhodné odprezentovanie našej práce na študentskej vedeckej konferencii IIT.src, naplánovali sme si aj úlohy súvisiace s úspešnou prezentáciou:

- Mobilná aplikácia - praktická ukážka použitia API modulu *SmartCollect*
- Kvalitný článok
- Pútavý plagát projektu

4 Celkový pohľad na systém

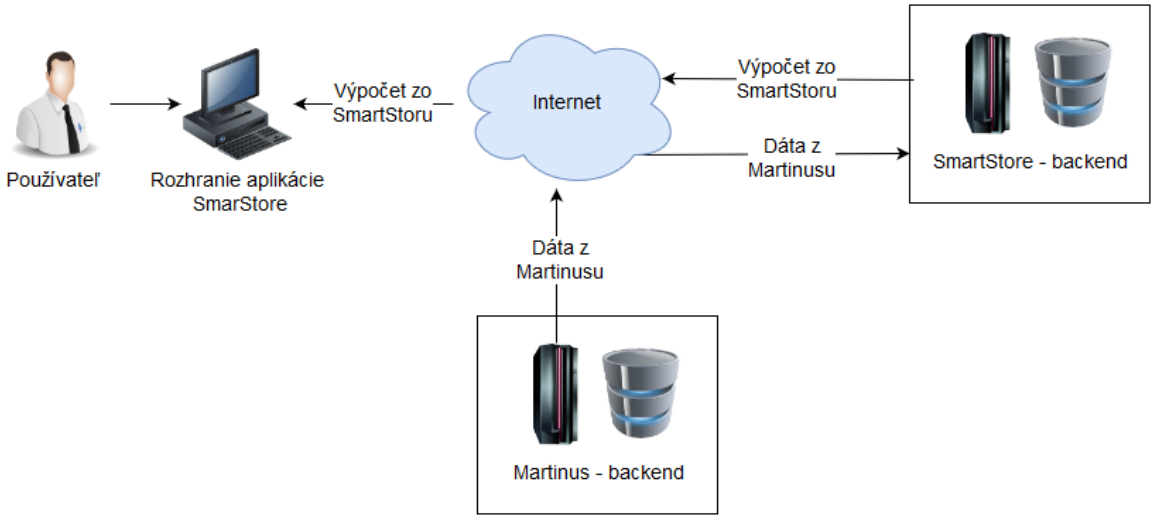
4.1 Architektúra SmartStore

Aplikácia SmartStore využíva architektúru klient-server. Klientská časť poskytuje používateľovi grafické rozhranie, ktoré je implementované ako tenký klient – jedná sa teda o webovú aplikáciu dostupnú pomocou webového prehliadača. Pomocou tohto rozhrania (*Rozhranie aplikácie SmartStore*) k aplikácii pristupuje *Používateľ*, ktorý má možnosť čítať zobrazovaný obsah alebo zasielať požiadavky na server.

Serverovú časť (*SmartStore - backend*) tvorí aplikačná logika s databázou, ktorej dátový model opisujeme v nasledujúcej sekcii 4.2. V rámci serverovej časti sú nasadené moduly:

- *SmartOrder* – modul na predikciu objednávok
- *SmartCollect* - modul na výpočet najefektívnejšej trasy pre pracovníka skladu a optimálne rozloženie skladu

Tieto moduly na výpočet používajú dáta zasielané z Matinusu (*Martinus - backend*).

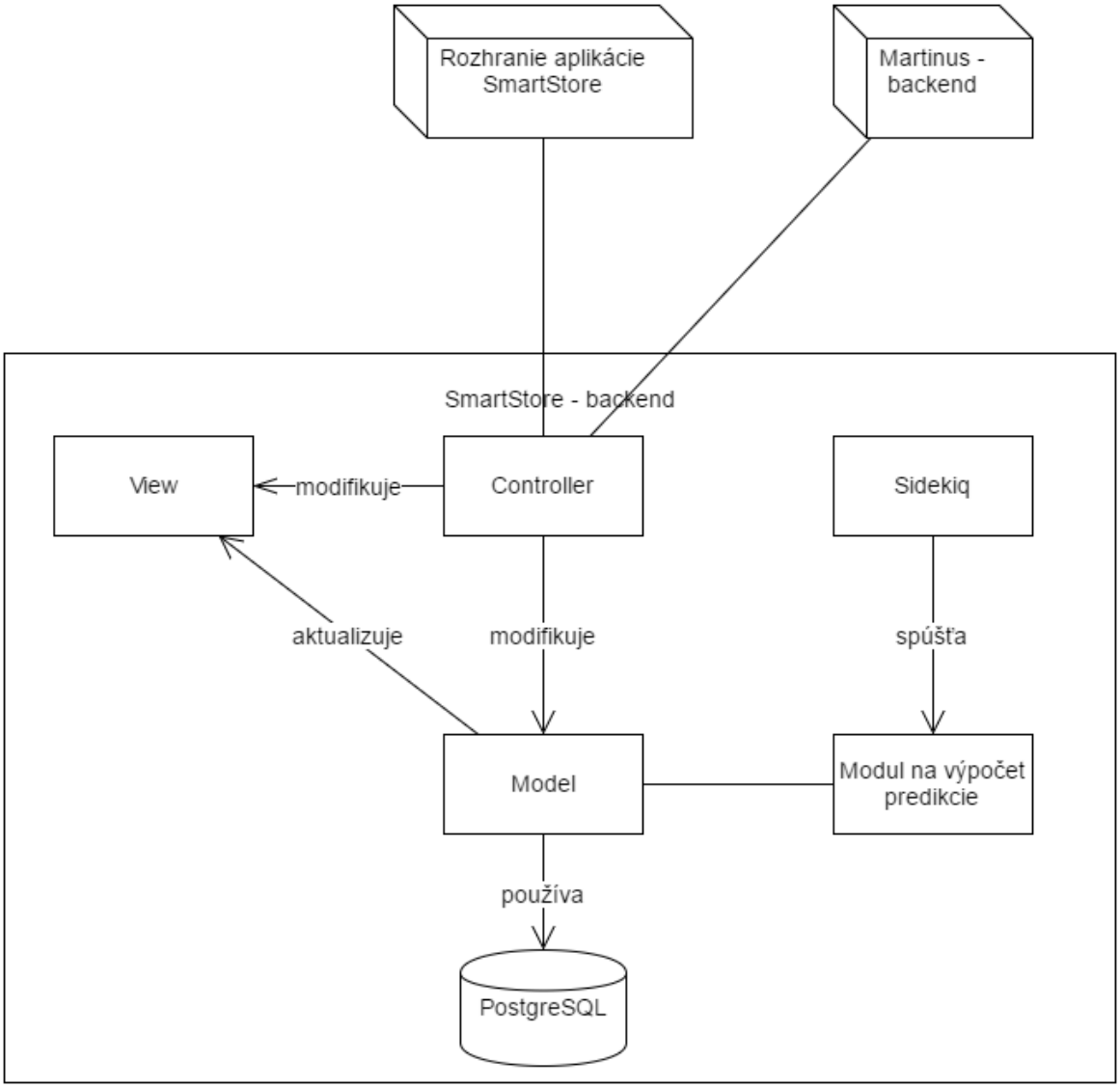


Obrázok č.1 Náčrt architektúry produktu SmartStore

4.1.1 SmartStore – backend

Serverová časť aplikácie je implementovaná s dodržaním konvencií architektúry MVC (*Model – View - Controller*), ktorú prirodzene podporuje použitý framework Ruby on Rails. Okrem aplikačnej logiky je dôležitou súčasťou backendu aj modul na predikciu objednávok. Serverová časť komunikuje s klientom pomocou REST API. Jednotlivé požiadavky sú priradené ku kontrolerom, ktoré vykonávajú operácie nad modelmi, ktoré pracujú s dátami. Na uchovanie dát je použitá databáza *PostgreSQL*.

Uskutočnené objednávky sú posielané z kníhkupectva každodenne. Ich pravidelné spracovanie je zabezpečené modulom *Sidekiq,* v rámci ktorého sú plánované procesy (napr. spracovanie dát, predikcia), ktoré sú vykonané na pozadí (*angl. backgroud processing*).



Obrázok č.2 Diagram znázorňujúci serverovú časť aplikácie SmartStore

4.2 Dátový model

Ústrednú časť dátového modelu sme vytvorili na základe informácií od vlastníka produktu podľa existujúceho dátového modelu používaného kníhkupectvom. Táto ústredná časť je tvorená triedami *Distributor, Product, Sale* a *Stock.* Každý produkt má špecifikovaný aktuálny stav skladu, ktorý je uchovaný v samostatnej triede sklad – *Stock* a má referenciu na distribútora. Predaj *Sale* sa viaže k určitému produktu a tiež uchováva stav počtu kusov, ktoré boli na sklade v čase predaja, aby bola uchovaná história stavu skladu.

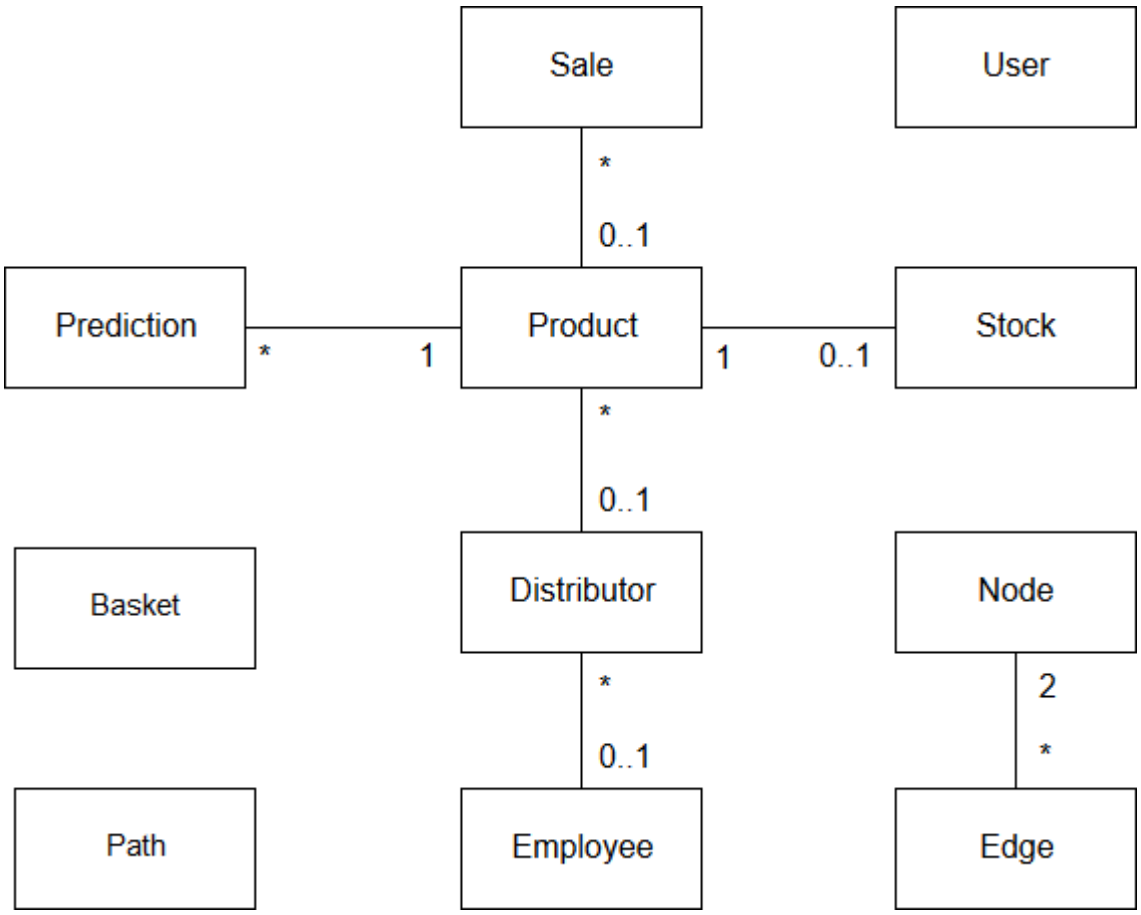
Jednotliví zamestnanci *Employees* skladu majú na starosti určitý počet distribútorov, od ktorých objednávajú produkty. Používateľ *User* nie je viazaný k zamestnancovi skladu, ale je ním akákoľvek osoba používajúca aplikáciu SmartStore, ktorá chce zobraziť predikcie *Predictions* k jednotlivým produktom.

Pre modul *SmartCollect* je potrebné ukladať grafovú reprezentáciu skladu, na čo slúžia modely *Node* a *Edge,* na uloženie spolupredávaných produktov slúži model *Basket.* Optimálne rozloženie skladu je riešené ukladaním potrebných atribútov s modeli *Product.*

Všetky triedy dedia funkcionalitu na vytváranie a upravovanie biznis objektov od triedy *ActiveRecord* poskytovanej frameworkom Rails.

- *Basket* – Nákupný košík; určuje, ktoré produkty sa predávajú spolu
  - *id1* – identifikátor prvého produktu
  - *id2* – identifikátor produktu, ktorý sa predáva s produktom *id1*
  - *position* – pozícia populárnejšieho z produktov *id1* alebo *id2*
- *Path* – Cesta; určuje prepočítanú hodnotu najkratšej cesty medzi dvomi bodmi, vrátane bodov cesty
  - *path*
  - *cost*
  - *path_points*
- *Node* – Uzol z grafovej reprezentácii skladu
  - *title* – názov uzla
- *Edge* – Hrana v grafovej reprezentácii skladu
  - *path_cost* – Ohodnotenie hrany podľa dĺžky trasy
  - *from_node_id* – Identifikačné číslo uzla, z ktorého hrana vychádza
  - *to_node_id* – Identifikačné číslo uzla, do ktorého hrana vchádza
- *Distributor* – distribútor produktov;
  - *delivery_days* – počet dní, za ktoré distribútor doručí produkt
  - *name* – meno distribútora
  - *employee_id* – identifikačné číslo zamestnanca skladu, ktorý má distribútora na starosti
- *Employee* – zamestnanec skladu;
  - *name* – meno zamestnanca skladu

- *Prediction* – predpoveď, predikcia; Výsledok algoritmu na predikovanie vypredania zásob, resp. objednanie tovaru
    - *product_id* – identifikačné číslo produktu, ku ktorému patrí predikcia
    - *out_of_stock* - predikovaný dátum, kedy sa zásoby úplne vypredajú
    - *books_count* - predikovaný počet kníh, ktoré objednať v (*need_to order*)
    - *need_to_order* - predikovaný dátum, kedy treba objednať nový tovar
- *Product* – produkt;
    - *distributor_id* - identifikačné číslo distribútora produktu
    - *name* – názov produktu (knihy)
    - *authors* – autori produktu (knihy)
    - *position* – reálna pozícia produktu v sklade
    - *popularity* – vypočítaná popularita produktu
    - *real_distance* – reálna vzdialenosť produktu od stredu skladu
    - *optimal_distance* – optimálna vzdialenosť produktu od stredu skladu
    - *optimal_position* – optimálna pozícia produktu
- *Sale* – predaj; Vykonaná objednávka, resp. predaj určitého tovaru
    - *date_time* – dátuma čas, kedy bol uskutočnený predaj
    - *product_id* - identifikačné číslo produktu, ktorý bol objektom predaja
    - *stock_count* – aktuálny stav skladu v čase uskutočnenia predaja
- *Stock* –sklad; záznam o stave skladu určitého produktu
    - *count* – počet kusov, ktoré sú aktuálne na sklade
    - *product_id* - identifikačné číslo produktu, ku ktorému sa viaže stav skladu
- *User* – používateľ aplikácie SmartStore
    - *user_name* – používateľské meno používateľa aplikácie
    - *encrypted_password* – heslo používateľa uložené v zašifrovanej podobe
    - *remember_created_at* – dátum a čas kedy používateľ zvolil možnosť zapamätať na počítači
    - *sign_in_count* – počítadlo prihlásení
    - *current_sign_in_at* – časová pečiatka aktuálneho prihlásenia
    - *last_sign_in_at* – časová pečiatka posledného prihlásenia
    - *current_sign_in_ip* – IP adresa, z ktorej je aktuálne používateľ prihlásený
    - *last_sign_in_ip* - IP adresa, z ktorej bol používateľ naposledy prihlásený



Obrázok č.3 Dátový model aplikácie SmartStore

4.3 Moduly

Jednotlivé moduly systému popisujeme tejto kapitole len stručne, podrobný opis modulov uvádzame v Prílohe A: *Moduly systému*.

4.3.1 Modul robotníci na pozadí

Tento modul slúži na spracovanie prác (*angl. jobs*) na pozadí aplikácie. Na pozadí spracúvame napríklad výpočty predikcií.

4.3.2 Modul predikcie

V module predikcie prebieha výpočet predikcie objednávok na základe predajov zaslaných z kníhkupectva. Tento modul je implementovaný v jazyku R. Na vstupe pre tento modul je súbor s exportom z databázy SmartStore a na výstupe súbor obsahujúci predikcie pre jednotlivé tovary.

4.3.3 Modul grafického rozhrania

Modul grafického rozhrania slúži na zobrazovanie výsledkov produktu *SmartOrder* – teda pomocou tohto modulu prehľadne zobrazujeme predikcie objednávok pre jednotlivé produkty.

4.3.4 Modul autorizácie používateľa

Tento modul slúži na autorizáciu používateľa kvôli chráneniu aplikácie pred nepovolanými osobami, ktoré nemajú právo používať aplikáciu a nemajú prihlasovacie údaje. Modul tiež zahŕňa autorizáciu API, aby sa na ňu mohli prihlasovať len používatelia, ktorí majú prihlasovacie tokeny.

4.3.5 Modul na hľadanie cesty v sklade

Modul patrí k produktu *SmartCollect*, cieľom ktorého je vyhľadať (vypočítať) najefektívnejšiu trasu pre pracovníka skladu pri vybavovaní objednávky, keďže objednávka môže zahŕňať viacero produktov, ktoré sa v sklade nachádzajú na rôznych miestach.

4.3.6 Modul na analýzu nákupného košíka

Modul patrí k produktu *SmartCollect*, kde slúži na pravidelný výpočet produktov, ktoré sa predávajú spolu. Na základe toho je potom možné zahrnúť informáciu o spolupredávaných produktoch do výpočtu optimálneho rozloženia skladu.

4.3.7 Modul na výpočet optimálneho rozloženia skladu

Modul patrí k produktu *SmartCollect,* kde slúži na každodenný výpočet optimálneho rozloženia skladu podľa aktuálnej popularity jednotlivých produktov a podľa produktov

predávaných spolu. Výsledkom je potom zobrazenie najnevhodnejšie umiestnených produktov.

4.4 Dátová analýza

Dátová analýza je realizovaná na informáciách o predajoch, ktoré nadobúdame od Martinusu. Je vykonávaná v programovacom jazyku R, určenom na štatistické analýzy. Vstupnou hodnotou do skriptu v tomto jazyku je CSV súbor, obsahujúci všetky potrebné informácie k predikcii.

Dáta, ktoré sa v súbore nachádzajú sú v prvej fázy prefiltrované, a pracuje sa len z relevantnými údajmi. Následne sú pomocou nami vytvorených funkcií a metód predikované dátumy o vypredaní, o objednaní a počte potrebných kusov. Výstupným artefaktom z dátovej analýzy je nový CSV súbor, ktorý obsahuje všetky spomenuté informácie ku každému produktu. V tejto časti dátová analýza končí a všetky nové údaje sú uložené do databázy.

4.4.1 Metóda predikcie dátumu

Metóda predikcie dátumu je založená na historických dátach. Konkrétne ide o predaje staré 90 dní. Každý predaj obsahuje informácie o presnom dátume a čase, kedy bola objednávka vytvorená, aktuálny stav skladu pre daný produkt, počet dní, kedy je dodávateľ schopný priviesť nový tovar a samotné identifikačné číslo dodávateľa.

V prvom prototype sa samotné predikovanie začína výpočtom rozdielu medzi aktuálnym dňom a prvým záznamom o danom produkte, kedy dostaneme dĺžku obdobia, za ktoré sa produkt predával. Za tento istý časový úsek spočítame počet predajov. Vďaka týmto dvom hodnotám sa vieme dostať k číslu, ktoré hovorí, ako rýchlo sa produkt predáva. Teda za aké časové obdobie sa predá jeden kus. Keďže poznáme rýchlosť predaja a aktuálny stav skladu, pre daný produkt, vieme približne určiť dátum, kedy sa produkt vypredá a taktiež dátum, kedy je potrebné spraviť objednanie nového tovaru

4.4.2 Metóda predikcie počtu kusov

Podobne ako predošlá metóda, aj táto je založená na historických dátach. Využíva rovnaké dáta ako predošlá metóda. Navyše je jej vstupom informácia o rýchlosti predaja produktu.

Pri predikovaný správneho počtu kusov, ktorý je treba objednať sa zameriavame na to, aby bol na sklade dostatočný počet kusov, ktorým budeme schopný pokryť obdobie medzi dvoma dodávkami tovaru od dodávateľa. Predikcia sa opiera najmä o rýchlosť predaja a počet dní, za ktoré príde nový tovar. Na základe týchto dvoch údajov je možné vyrátať počet kusov, ktorý by mal stačiť na pokrytie spomenutého obdobia.

4.4.3 Metóda analýzy nákupného košíka

Bez historických dát sa neviete zaobísť ani pri tejto metóda. Taktiež využíva informácie staré 90 dní. Nakoľko analyzujeme príbuznosť jednotlivých produktov, respektíve ich naviazanie v rovnakých objednávkach, zameriavame sa predovšetkým na presné časové údaje. Z dostupných dát tak vybrať produkty, ktoré patrili do rovnakej objednávky.

Zameriavame sa hlavne na zistenie vzťahov medzi produktmi. Snažíme sa určiť, ktoré produkty sa objavujú častejšie v rovnakých objednávkach. Proces začína rozdelením produktov do konkrétnych údajov na základe dostupných informácií o čase objednania. Pre každý produkt je následne vytvorený histogram, ktorý hovorí o počte predaných kusov iných produktov, ktoré sa s ním vyskytovali v objednávkach. Výsledkom metódy je tabuľku obsahujúca zoradené produkty podľa intenzity predávanie, spolu s najčastejšími produktami vyskytujúcimi sa v rovnakých objednávkach. Takto vytvorené záznamy sú uložené a poslané na ďalšie spracovanie.

5 Prílohy

- Príloha A: Moduly systému
- Príloha B: Inštalačná príručka
- Príloha C: Technická dokumentácia

Príloha A Moduly systému

V prílohe sa nachádza popis nasledujúcich modulov systému:

1 Modul - robotníci na pozadí

1.1 Analýza

1.1.1 Požiadavky

V rámci projektu je potrebné vykonávať zložité operácie na pozadí, ktoré by mali byť manažované zvlášť. Identifikované operácie vieme kategorizovať nasledovne:

- pravidelný import dopytovaných dát z Martinusu,
- pravidelné obnovenie predpovedí na základe dát,
- pravidelné spracovanie získaných dát z Martinusu.

Jednotlivé operácie by sa mali vykonávať v čase, kedy nebude so systémom pracovať nikto, aby sa nezhoršila funkcionalita systému.

Operácie musia byť schopné sa zotaviť v prípade neočakávaných chýb. Chyby, ktoré vzniknú by mali byť zaznamenané a odoslané na analýzu zodpovednej osobe.

Operácie musia byť v čo najväčšej miere automatizované, zároveň však nemôžu zaťažiť server vo vysokej miere.

Administrátorovi aplikácie a vlastníkovi produktu by mala byť prístupná konzola pre správu a sledovanie úspešnosti vykonávania operácii.

Modul by mal využívať softvérové riešenia, ktoré daný problém riešia a sú overené.

1.1.2 Existujúce riešenia

V súčasnosti sa na správu operácií v pozadí webových aplikácií využívajú asynchrónny robotníci (*angl. workers*), ktorý vykonávajú operácie definované ich telom. Pre technológiu Ruby on Rails sú robotníci zahrnutí priamo v architektúre v podobe prác (*angl. jobov*). Webový vývojár si pre ich aplikáciu musí vybrať softvérové riešenie na správu radov takýchto robotníkov. Medzi najznámejšie patria:

- *Sidekiq,*
- *Resque,*
- *Delayed Job.*

Riešenia potrebujú na svoj chod aj technológiu *Redis*, ktorú používajú na dynamické ukladanie informácií o poradí robotníkov v rámci radov.

1.2 Návrh

V našom projekte navrhujeme použiť jednu z analyzovaných technológií na správu robotníkov na pozadí. Niektorý robotníci budú v závislosti od dohody automatizovaný tak, aby sa púšťali samostatne. Na správu robotníkov na pozadí sme navrhli použiť technológiu *Sidekiq* na základe týchto požiadaviek:

- požiadavka na rýchlosť vykonania prác, v ktorej je *Sidekiq* najlepšie riešenie zo všetkých identifikovaných riešení,
- požiadavka na jednoduché rozšírenie o periodických robotníkov, ktorý sa budú spúšťať automaticky – priamo v *Sidekiq* technológii,
- požiadavka na jednoduché manažovanie súbežne vykonávaných procesov a riadenie vykonávania procesov na základe výkonnosti servera,
- jednoduché rozšírenie rozhrania o prehľadovú konzolu spracovania prác v čase.

Robotníci budú vykonávať zložité operácie spojené s importnom dát a predikciou, ktorá nie je vykonávaná priamo v aplikácií, ale vyžaduje si samostatné spustenie a sledovanie stavu vykonania.

1.3 Implementácia

1.3.1 Zvolená technológia

Nami implementované riešenie pozostáva z využitia technológie *Sidekiq* v rámci prác (*angl. jobov*) architektúry frameworku Rails. Technológia *Sidekiq* vyžaduje aj funkčnú inštaláciu noSQL databázy *Redis*.

1.3.2 Implementácia jobov

V súčasnosti sú implementovaní nasledujúci robotníci:

1. *CreateCsvForProcessingJob* (periodický robotník - každý deň v noci)
    - *Vstup*: -
    - *Činnosť*: vytvorenie CSV s potrebnými informáciami pre komponent na dátovú analýzu, spustenie a kontrola skriptu na dátovú analýzu, uloženie výsledku do databázy
    - *Výstup*: predikcia dňa vypredania zásob a počtu kníh potrebných na objednanie,
2. *ImportJsonJob*
    - *Vstup*: cesta k prijatému súboru s dátami z Martinusu
    - *Činnosť*: spracovanie nových dát a aktualizácia starých prijatých z Martinusu
    - *Výstup*: aktuálne informácie v databáze potrebné na dátovú analýzu
3. *ImportProductsJob* (periodický robotník – každý deň v noci)
    - *Vstup*: -

- *Činnosť*: stiahnutie informácií o nových produktov (a k nim pridruženým modeloch), aktualizácia starých informácií
- *Výstup*: aktuálne informácie v databáze o produktoch

1.4 Testovanie

Každý z robotníkov na pozadí je testovaný zvlášť pomocou automatizovaných testov v testovacom frameworku Rspec implementovanom priamo v Ruby on Rails.

Testy pre jednotlivých robotníkov:

CreateCsvForProcessingJob (periodický robotník - každý deň v noci)

- Kontrola vytvorenia CSV súboru s predajmi
- Kontrola vytvorenia CSV súboru s predikciami
- Kontrola naplnenia súboru s predajmi
- Kontrola naplnenia súboru s predikciami
- Kontrola naplnenia databázy predikciami
- Kontrola či nedošlo k zmene názvu fronty pracovníka

ImportJsonJob

- Kontrola naplnenia tabuľky s predajmi v databáze
- Kontrola naplnenia tabuľky so skladovými zásobami v databáze
- Kontrola naplnenia tabuľky s produktmi v databáze
- Kontrola korektného behu pri validnom súbore s 0 predajmi
- Kontrola správnej úpravy počtu tovaru na sklade v databáze
- Kontrola vytvorenia ešte neexistujúceho tovaru do databáze
- Kontrola či nedošlo k zmene názvu fronty pracovníka produktov na sklade

ImportProductsJob

- Kontrola či sa po importe dát z CSV súboru nachádzajú v tabuľke *Product*
- Kontrola či sa po importe dát z CSV súboru nachádzajú v tabuľke *Distributor*
- Kontrola zmeny hodnoty v tabuľke *Product* u produktu kde došlo k aktualizácii hodnôt
- Kontrola zmeny hodnoty v tabuľke *Distributor* u distribútorov kde došlo k aktualizácii hodnôt.
- Kontrola či nedošlo k zmene názvu fronty pracovníka

2 Modul predikcie

2.1 Analýza

Pri prvotnej analýze problému sme uvažovali nad rôznymi riešeniami. Do úvahy nám pripadali dva programovacie jazyky, ktoré bolo možné použiť. Prvým z nich je Python, druhý jazyk R. Pri podrobnejšom skúmaní problematiky sme sa v projekte rozhodli využiť práve jazyk R. Hlavným dôvodom bola jednoduchá manipulácia s dátami a predovšetkým dostupnosť rôznych knižníc, ktoré nebolo potrebné inštalovať. Python bol tiež dobrou alternatívou, avšak nepotrebovali sme programátorský prístup, nakoľko aplikácia je programovaná v inom prostredí, a preto sme ho nepoužili.

2.2 Návrh

V projekte budeme používať nami navrhnuté metódy na predikciu. Hlavným cieľom predikcie je stanoviť dve hlavné výsledné hodnoty, dátum, kedy treba tovar objednať a počet kníh, ktorý treba objednať. Preto budeme navrhovať dve primárne metódy, ktoré tieto výsledky zabezpečia.

Základom pre obe navrhované hodnoty, budú štatistické údaje. Pomocou dátovej analýzy ich preskúmame a použitím navrhovaných metód získame potrebné výsledky, teda dátum a počet.

2.3 Implementácia

2.3.1 Technológia

Keďže pôjde primárne o skúmanie štatistických a historických záznamov o predajoch, na predikciu budeme využívať programovací jazyk R.

2.3.2 Implementácia metód na predikciu

V súčasnosti máme implementované dve základné metódy:

1. Metóda na predikciu dátumu objednania
    - *Vstup*: CSV súbor s históriou predaja
    - *Vykonanie*: na základe dátumu objednávky, aktuálneho stavu skladu a rýchlosti dodania nového tovar, stanovíme predpokladaný dátum vypredania zásob, z ktorého následne určíme dátum, kedy je potrebné produkty objednať
    - *Výstup*: CSV súbor s predikovanými hodnotami
2. Metóda na predikciu počtu kníh na objednanie
    - *Vstup*: CSV súbor s históriou predaja

- *Vykonanie*: na základe priemernej rýchlosti objednania a času dodania nového tovaru, stanovíme predpokladaný počet kníh, ktoré je treba objednať

2.4 Testovanie

Pre porovnávanie jednotlivých metód predikcie a ich úspešnosti používame nami implementovaný nástroj, ktorý dokáže porovnať naraz viacero metód predikcie aj s úspešnosťou zamestnancov Martinusu. Pre vyhodnotenie úspešnosti dokáže nástroj porovnať predpoveď s reálnymi predajmi.

3 Modul grafického rozhrania

3.1 Analýza

Keďže náš projekt je riešený ako internetová aplikácia, voľba technológií bola zrejmá. V projekte sa používa pre tvorbu rozhrania HTML, Javascript, CSS, respektíve SCSS. Pre zobrazovanie grafov existuje viacero možností. Na tvorbu sme si vybrali pre porovnanie dve technológie používané knižnice a to:

- *Chartkick,*
- *Charts.js*.

3.2 Návrh

V našom projekte sme návrh používateľského rozhrania konzultovali s vlastníkom produktu, ktorý nám zadal požiadavky čo by malo rozhranie používateľovi umožňovať. Hlavnou požiadavkou bolo požívateľovi vypísať predikcie predaja kníh, ktoré sme vypočítali na základe prijatých dát z Martinusu o predajoch. Ďalšia požiadavka sa týkala možnosti filtrovania týchto predikcií podľa prideleného zamestnanca, ktorý daný produkt má na starosti a podľa distribútora produktu.

Ďalšia časť rozhrania, ktorá bola vyžadovaná, predstavovala zobrazovanie grafov predajov pre jednotlivé výrobky. Obidve knižnice spomenuté v analýze ponúkajú bohatú palety funkcií. Po zvážení sme sa rozhodli v našom projekte použiť *Charts.js*, keďže ponúka podporu aj starších prehliadačov a negeneruje grafy v podobe svg, ako druhá spomínaná technológia.

Produktový vlastník zároveň poznamenal, že toto grafické prostredie nie je konečné a môže sa v budúcnosti meniť podľa požiadaviek ľudí, ktorý systém budú používať.

3.3 Implementácia

Implementovali sme veľmi jednoduché používateľské rozhranie aplikácie použitím bežného HTML spolu s *Embedded RuBy* (*ERB*). Využitím frontend-ového framework-u *Bootsrap* sme vytvorili responzívny dizajn stránky, aby sa rozhranie prehľadne zobrazovalo na všetkých zariadeniach bez ohľadu na veľkosť displeja. Pre implementáciu grafov bola použitá knižnica *Charts.js*, ktorá je taktiež responzívna.

Keďže v rozhraní zobrazujeme predikcie pre obrovské množstvo produktov bolo potrebné zabezpečiť stránkovanie týchto predikcií, aby sme zachovali prehľadnosť. Na stránkovanie sme využili gem *will_paginate*.

Aby sme umožnili používateľovi filtrovať predikcie podľa distribútora a prideleného zamestnanca, implementovali sme filter vo forme elementu „výberu zo zoznamu“, ktorý je umiestnený nad zobrazenými predikciami.

Ako poslednú vec sme do rozhrania implementovali zobrazenie počtu predajov v jednotlivé týždne v roku, v podobe grafu, ktorý je možne zobraziť pre každú predikciu zvlášť prejdením kurzora myši nad ikonou grafu.

3.4 Testovanie

Tento modul si vzhľadom na jeho malú zložitosť testuje vývojár manuálne sám po implementácii každej zmeny.

4 Modul autorizácie používateľa

4.1 Analýza

Vzhľadom na to, že aplikácia projektu je voľne prístupná na internete je nutné ju zabezpečiť a teda povoliť prístup k nej iba oprávneným osobám aby sme predišli jej neoprávnenému použitiu.

Modul by mal využívať softvérové riešenia, ktoré daný problém rieši, sú overené a zároveň jednoduché pre použitie našich služieb reálnymi používateľmi.

4.1.1 Existujúce riešenia

Autentifikácia používateľov je pomerne častý problém, ktorý sa rieši v internetových aplikáciách existuje mnoho riešení. Ruby on Rails umožňuje jednoduchú integráciu riešení autentifikácie používateľov, ktoré už niekto implementoval a je možné ich voľne používať. Existujúce riešenia, ktoré sme analyzovali sú:

- *Devise*
- *OmniAuth*
- *Authlogic*

Tieto riešenia sú zároveň najsťahovanejšie spomedzi všetkých riešení autentifikácií.

4.2 Návrh

Z analyzovaných existujúcich riešení pre autentifikáciu grafického rozhranie navrhujeme použiť práve Devise. Tento nástroj patrí medzi tie najrozsiahlejšie autentifikačné riešenia pre Ruby on Rails a aj napriek tomu, že využiť jeho plný potenciál v našej aplikácii nie je možné, rozhodli sme sa ho použiť práve z týchto dôvodov:

- jednoducho rozšíriteľný pri nových požiadavkách na autentifikáciu,
- ľahko modifikovateľný na náročnejšie požiadavky.

Devise nám zabezpečí, že budeme vedieť jednoducho doimplementovať takmer akúkoľvek funkčnosť do aplikácie podľa požiadaviek vlastníka produktu z pohľadu rôznych druhov používateľov.

Ďalšou časťou systému, ktorú treba ochrániť je samotný prístup na naše API služby, ktoré poskytujeme. Pre túto ochranu sme sa rozhodli využiť jednoduchý a overený prístup cez autorizačný reťazec znakov. Pre použitie týchto služieb musí požiadavka na server obsahovať unikátny autentifikačný kľúč. V inom prípade nebude obslúžená.

4.3 Implementácia

V súčasnosti je pre prototyp aplikácie implementovaná jednoduchšia verzia autentifikácie. Tá zahŕňa prístup iba k jednému používateľovi. Prístup k tomuto používateľovi budú mať všetci zamestnanci.

Do budúcnosti očakávame doimplementovanie používateľského modelu vytvorením bežného používateľa aplikácie a administrátora.

Implementácia ochrany API služieb je veľmi jednoduchá, keďže je potrebné skontrolovať, či sa daný reťazec nachádza v žiadosti na server pod daným kľúčom.

4.4 Testovanie

Vytvorené riešenie je otestované jednoduchými testami zahrňujúcimi skúšku prihlásenia so správnymi a taktiež aj s nesprávnymi prístupovými údajmi. API rozhranie je otestované identicky.

5 Modul na hľadanie cesty v sklade

5.1 Analýza

Vyhľadávanie optimálnej cesty v sklade je tzv. problém obchodného cestujúceho, kde je veľmi náročné vytvoriť efektívny algoritmus. V našom prípade, je nutné prejsť každým daným bodom, pričom sa cesta začína aj končí v rovnakom bode. Pri analýze modulu na hľadanie cesty v sklade sme uvažovali dve alternatívy na vyhľadávanie cesty v sklade:

- Evolučný algoritmus (problém obchodného cestujúceho) – použitie vhodné v prípade, že by sme v sklade hľadali optimálnu cestu medzi veľkým počtom políc,
- Hľadanie cesty A star algoritmom – použitie vhodné v prípade, že by sme v sklade hľadali optimálnu cestu medzi menším počtom ulíc.

Pri analýze sme zohľadňovali reálne požiadavky na vyhľadávanie v sklade. Cesta v sklade je generovaná priemerne pre tri 3 +- 2 body (police). Pri použití A star algoritmu by sme tak v najhoršom prípade (5 bodov) museli hľadať najlepšiu cestu medzi 5! možnými cestami. Naopak pri implementácii evolučného algoritmu by zložitosť výpočtu klesla, ale optimálnosť cesty by bola ovplyvnená náhodou. Pri uvažovaní algoritmu sme prihliadli na potenciálnu rýchlosť generovania cesty a správnosť nášho riešenia.

5.2 Návrh

Z dvoch analyzovaných možností navrhujeme použiť A star algoritmus s heuristikou využívajúcou euklidovu vzdialenosť bodov. Pre reálne požiadavky skladu je toto riešenie vhodnejšie, pretože bude poskytovať najlepšiu možnú cestu a priemerný počet bodov (políc) si nevyžaduje implementáciu evolučným algoritmom, respektíve daný počet bodov nedosiahne natoľko vysokého počtu, aby bolo vyhľadanie cesty výpočtovo veľmi náročné.

Na základe zoznamu bodov (políc), ktoré musí pracovník skladu navštíviť navrhujeme vytvoriť permutácie bodov a pre každú z nich zistiť cenu cesty na základe reálnych dát skladu. Po identifikovaní optimálnej cesty v odpovedi server pošle poradie bodov (políc), ktoré má pracovník skladu navštíviť.

5.3 Implementácia

Implementácia modulu na hľadanie cesty v sklade spočíva v konštrukcii reprezentácie skladu v aplikácii a konštrukcii algoritmu na hľadanie cesty. V súčasnosti je sklad uložený v databáze a v aplikácii sa využíva jeho jediná inštancia. Algoritmus vyhľadáva optimálnu cestu v rámci permutácií na základe počtu vyhľadávaných bodov.

Pre potreby A star algoritmu používame heuristiku, kde sa body zo skladu premietnu do dvojrozmerného priestoru. Následne sa vypočítava heuristická vzdialenosť ako euklidova

vzdialenosť dvoch bodov. Pre čo najlepšie usmernenie vyberania prvkov je táto funkcia prispôsobená reálnym podmienkam skladu. Keďže sklad má obdĺžnikový charakter, dva body v dvoch za sebou idúcich radoch majú malú vzdialenosť, ale reálne je potrebné obísť celý rad. Pre zlepšenie výberu sme sa teda pokúsili zväčšiť túto vzdialenosť a tým pádom lepšie usmerniť vyberanie prvkov v algoritme.

Aplikácia poskytuje API, ktorá ako parametre prijíma body (police), ktoré musí pracovník navštíviť. Po obdržaní žiadosti na vyhľadanie optimálnej cesty sa najskôr vytvoria všetky možné permutácie. Následne sa začne vyhľadávať najkratšia cesta pre danú permutáciu pomocou A star algoritmu s vyššie popísanou heuristikou. Po vyhľadávaní najkratšej cesty medzi dvoma bodmi, sa táto cesta uloží do globálnej hash premennej triedy. Pri ďalšej permutácií alebo úplne inom prehľadávaní už nebude potrebné túto cestu znova hľadať, ale sa s konštantným časom vyberie jej cena. Toto pamätanie nám výrazne urýchli vyhľadávanie a tým pádom čas potrebný na nájdenie cesty. Po ohodnotení všetkých permutácií sa následne vyberie najkratšia možnosť. V odpovedi sa posielajú body zoradené na základe výstupu algoritmu - optimálneho navštívenia políc skladu.

5.4 Testovanie

5.4.1 Akceptačné testovanie

V rámci výsledného testovania a vyhodnotenia modulu sme vytvorili niekoľko testov na porovnanie výsledkov algoritmu a súčasného riešenia.

1. Testovanie optimálneho počtu štartovných a koncových bodov pri prechádzaní skladov. Testovanie prejdenej vzdialenosti pri odlišných štartoch:
    a. 1 štart: 112 316 km
    b. 2 štarty: 121 070 km
    c. 3 štarty: 108 587 km
    d. 4 štarty: 116 744 km
    e. 5 štartov: 105 887 km

    Po testovaní sme identifikovali optimálne množstvo štartov v sklade – 5 štartov.

2. Testovanie prejdenej vzdialenosti pri použití najlepšej cesty v porovnaní s prejdenou cestou s abecedným usporiadaním – po testovaní sme identifikovali zlepšenie prejdenej vzdialenosti o 4,3 až 10,2 percenta.

Spoločný štart aj cieľ

	Typ cesty: abecedné usporiadanie (m)	Typ cesty: najlepšia cesta (m)	Zlepšenie (%)
5 štartov (2-10 produktov v objednávke)	108621	102922	5,2
5 štartov (6-10 produktov v objednávke)	45025	41313	8,2
4 štarty (2-10 produktov v objednávke)	131446	125406	4.6
4 štarty (6-10 produktov v objednávke)	50834	46909	7,7
3 štarty (2-10 produktov v objednávke)	109344	103136	5,6
3 štarty (6-10 produktov v objednávke)	45450	41430	8,8
2 štarty (2-10 produktov v objednávke)	131328	125395	4,5
2 štarty (6-10 produktov v objednávke)	51055	47198	7,5
1 štart (2-10 produktov v objednávke)	112316	106010	5,6
1 štart (6-10 produktov v objednávke)	46479	42414	8,7

Odlišný štart aj cieľ

	Typ cesty: abecedné usporiadanie (m)	Typ cesty: najlepšia cesta (m)	Zlepšenie (%)
5 štartov (2-8 produktov v objednávke)	105887	95062	10,2
5 štartov (5-8 produktov v objednávke)	66831	60009	10,2
4 štarty (2-8 produktov v objednávke)	116744	110182	5,6
4 štarty (5-8 produktov v objednávke)	71829	67361	6,2
3 štarty (2-8 produktov v objednávke)	108587	98103	9,6
3 štarty (5-8 produktov v objednávke)	68472	61726	9,8
2 štarty (2-8 produktov v objednávke)	121070	116000	4,1
2 štarty (5-8 produktov v objednávke)	74246	71000	4,3

3. Testovanie prejdenej vzdialenosti pri použití najlepšej cesty v porovnaní s mediánom všetkých kombinácií prejdených ciest – po testovaní sme podobnými experimentami ako v kroku 2 namerali zlepšenie až 16,4 – 30% prejdenej cesty.

6 Modul na analýzu nákupného košíka

6.1 Analýza

Keďže za objednávku sa zvyčajne platí poštovné, je bežné, že zákazníci častokrát nakupujú viacero produktov spolu. Príkladom je nakúpenie viacerých častí detektívneho románu, či učebníc pre žiaka základnej školy. Na týchto príkladoch je zrejmé, že sú predávané spolu, stáva sa však, že spolu sú predávané aj produkty, ktoré zdanlivo na prvý pohľad nemajú veľa spoločné a dôvod, prečo ich zákazníci kupujú spolu, je pre bežnú dátovú analýzu skrytý. Analýza nákupného košíka (*angl. basket analysis*) je však bežný spôsob, ako je možné identifikovať spolu predávané produkty.

6.2 Návrh

Pre potreby analýzy nákupného košíka využijeme historické dáta z obdobia posledných 90 dní. Záznamy o predajoch je potrebné preskúmať a jednotlivé predaja rozdeliť do príslušných objednávok. Až následne takto zatriedené dáta je možné podrobne analyzovať a dosiahnuť požadovaného cieľa v podobe spolu predávaných produktov. Keďže poskytnuté dáta sú v podobe surových dát, na vyriešenie daného problému využijeme programovací jazyk R, ktorý poskytuje a spĺňa všetky predpoklady pre daný problém.

6.3 Implementácia

Proces analýzy je postavený na nami vytvorenej metóde. Analýza začína vybraním všetkých unikátnych produktov zo skúmaných dát. Takto získané produkty prechádzame, a pre každý produkt vyberieme jeho všetky predaje. V tomto bode sa dostávame do pozície, kedy máme extrahované predaje o jednom konkrétnom produkte spolu s presnými časmi objednaní, ktoré zohrávajú dôležitú úlohu v procese analýzy. Ďalším krokom je na základe spomenutých časových záznamov vybrať zo všetkých produktov len tie, ktoré spĺňajú rovnaké časové záznamy. Tieto vytiahnuté predaje predstavujú produkty, ktoré sa predali spolu s analyzovaným unikátnym produktom. Zo spolu predávaných produktov je vytvorený histogram, ktorý je zoradený na základe intenzity predajov a následne použitý na ďalšie spracovanie

7 Modul na optimálne rozloženie skladu

7.1 Analýza

Rozloženie produktov v sklade je aktuálne riešené len pracovníkmi, na základe ich skúseností. Toto spôsobuje, že produkty sú v sklade častokrát rozmiestnené nevhodne, neberúc do úvahy početnosť ich predajov, či to, či sa predávajú spolu. Príkladom je, že pracovník musí ísť vybrať do objednávky fiktívny produkt *Čítanka 1* umiestnený na severnej strane skladu a k nemu produkt *Čítanka 2,* umiestnený na južnej strane skladu. Práve tomuto scenáru by sme sa chceli vyhnúť, keďže zbytočne predlžuje cestu skladom a tým aj čas vybavenia objednávky.

7.2 Návrh

Modul na optimálne rozloženie skladu bol navrhnutý na úzku spoluprácu s modulom na vyhľadanie najkratšej cesty. Cieľom vytvorenia tohto modulu je ešte viac skrátiť cestu tým, že produkty, ktoré sa často predávajú, budú blízko miestu, odkiaľ vychádzajú pracovníci skladu pri vybavovaní objednávok (stred skladu). V rámci tohto modulu bude pracovníkom poskytnuté používateľské rozhranie, obsahujúce všetky produkty, ktoré sú aktuálne na sklade, zoradené od tých, ktorých pozícia v sklade je na základe parametrov najmenej vhodná.

7.3 Implementácia

Modul optimálneho rozloženia skladu ukladá postupne produkty do políc v sklade, pričom uvažuje, že sklad je prázdny. Algoritmus je závislý na *Module na analýzu nákupného košíka 6,* ktorý mu poskytuje údaje o spolu predávaných produktoch.

Najskôr pridelí všetkým produktom, ktoré sa aktuálne nachádzajú v sklade popularitu, ktorá reprezentuje počet predajov za posledných 14 dní. Ku každému produktu sa tiež zistí aktuálna vzdialenosť v metroch od stredu skladu aby nám bolo umožnené porovnávať ich súčasnú vzdialenosť s odporúčanou. Následne algoritmus začne priradzovať k najbližším policiam od stredu produkty, od najvyššej predajnosti za posledných 14 dní (tj. najvyššia popularita). Po priradení všetkých produktov, ktoré sú na sklade, priradzovanie končí.

Na umiestenie spolu predávaných produktov vedľa seba, využijeme výstup z *Modulu na analýzu nákupného košíka* a pri priradzovaní vhodných políc k produktom zakaždým kontrolujeme, či existuje k danému produktu nejaký iný, ktorý sa s nim dobre predáva. Ak áno, spolu predávanú knižku umiestnime do rovnakej, prípadne do police hneď vedľa.

7.4 Testovanie

7.4.1 Akceptačné testovanie

Pri akceptačnom testovaní sme porovnávali vplyv optimálneho rozmiestnenia produktov v sklade a použitia najlepšej cesty na prejdenú vzdialenosť priamo v sklade. Testovanie spočívalo vo vytvorení offline experimentu. Testovanie bolo charakteristické nasledujúcimi črtami:

- Testovanie spočívalo v aplikácií metód (hľadanie najkratšej cesty, optimálne rozloženie skladu) na objednávky jedného dňa.
- Všetky položky skladu boli reorganizované na základe výsledkov modulu.
- Prejdená vzdialenosť bola porovnávaná so vzdialenosťou, ktorú prešli pracovníci skladu bez použitia akejkoľvek metódy.

Experiment na jednom dni ukázal zlepšenie prejdenej cesty o 44,1%. Aplikovanie modulu na reorganizovanie skladu samo o sebe zlepšilo prejdenú vzdialenosť o cca 40%. Použitie optimálnej cesty pri reorganizovanom sklade neprinieslo veľké zlepšenie, pretože rozmiestnenie skladu už bolo najvhodnejšie vzhľadom k použitým metódam a cesta ku produktom tak už bola takmer optimálna.

Príloha B Inštalačná príručka

Inštalačná príručka a závislosti:

- Potrebná Ruby verzia: 2.3.1
- Potrebná Rails verzia: 5.0.0.1
- Systémové závislosti: Linux OS, PostgreSQL
- Inštalácia knižníc (v priečinku aplikácie): bundle install
- Meno, heslo databázového používateľa (vývoj): postgres/postgres
- Databázová konfigurácia: V **/etc/postgresql/9.5/main/pg_hba.conf** zmeniť všetky **METHODS** z *peer* na *trust*
- Vytvorenie databáz: rails db:create
- Inicializácia tabuliek: rails db:migrate
- Zaplnenie tabuliek: rails db:seed
- Štart servera: rails s
- Domovská stránka vývoja: http://localhost:3000/smartstore
- Prihlásenie na stránku -> prihlásenia: **"martinus"**, heslo: **"********"**
- R verzia: 3.3.1 a potrebné knižnice a nastavenia:
    - splitstackshape
    - plyr
    - pastecs
    - environmentálna premenná:
        - .bashrc -> export R_DATA_PATH="..../Smart-Store/prediction/"

Príloha C Technická dokumentácia

Technická dokumentácia bola vygenerovaná zo zdrojového kódu vo formáte HTML. Je dostupná v elektronickej forme na elektronickom médiu priloženom k tlačenej dokumentácii.