Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Webová aplikácia na správu cenníkov

Bakalárska práca

2021
Adam Gonšenica

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Webová aplikácia na správu cenníkov

Bakalárska práca

Študijný program:	Aplikovaná informatika
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	RNDr. Marek Nagy, PhD
Konzultant:	Ing. Peter Kello, PhD

Bratislava, 2021
Adam Gonšenica

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Adam Gonšenica
Študijný program: aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Webová aplikácia na správu cenníkov
Web application for the pricelist management

Anotácia: Aplikácia spracúva ceny produktov, ktoré prichádzajú ako POST request v JSON formáte a zobrazuje ich v reálnom čase na strediskových obrazovkách. Spracovanie requestu nesmie spomaľovať systém a musí byť spracované do 100 ms. Ceny sa ukladajú do PostgrSQL databázy. Aby bola zaručená rýchlosť spracovania pri viacerých requestoch naraz, musí byť request spracovaný postupne. Po spracovaní requestu je nutné ceny premietnuť do obrazoviek pomocou WebSocket technológie.

Cieľ:
1. Vytvorenie vhodného návrhu a implementácia DB pre aplikáciu.
2. Vytvorenie vhodného mechanizmu na postupné spracovanie requestov a použitie dátových štruktúr pre čo najrýchlejšie spracovanie.
3. Nájdenie a zapracovanie technológie pre zobrazovanie nových cien v reálnom čase na viacerých miestach naraz.
4. Analýza a implementácia pre front-end aplikácie
5. Rozdelenie aplikácie podľa MVC alebo iného vhodného návrhového vzoru
6. Použite technológií Node.js a express.js pre API časť aplikácie.

Poznámka: Softvérové nástroje: Node.js, socket.io, express.js, react, vue, angular, pug,...
Aplikácia bude vyvíjaná pre potreby spoločnosti TMR, a.s.,
ktorej konzultantom bude Ing. Peter Kello, PhD.

Vedúci: RNDr. Marek Nagy, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Dátum zadania: 28.09.2021

Dátum schválenia: 30.09.2021 doc. RNDr. Damas Gruska, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie: Chcem sa poďakovať školiteľovi RNDr. Marekovi Nagyovi PhD. a konzultantovi Ing. Petrovi Kellovi PhD. za pomoc, cenné rady a čas, ktorý mi venovali počas tvorby mojej bakalárskej práce.

Abstrakt

Bakalársku prácu sme robili vo firme Tatry Moutain Resorts a.s. prevádzkovateľa turistických rezortov v regióne strednej a východnej Európy. Spoločnosť implementovala nástroj na manažovanie vyťaženosti stredísk, dynamické ceny lístkov – Flexi ceny. Prvá verzia dynamických cenníkov z roku 2019 mala problém s pomalou odozvou na dopyt po cenách a aj s pomalou odozvou na prijatie requestov s novým cenníkom. Preto sa ju spoločnosť rozhodla prerobiť. Cieľom tejto bakalárskej práce bolo vytvoriť novú verziu, ktorá zabezpečí čo najefektívnejšie spracovanie cien. Ceny je nutné spracovať s rýchlym časom odozvy na request a premietnuť tieto zmeny do obrazoviek s cenami lístkov v strediskách, ktoré sú súčasťou tejto aplikácie. V krajnom prípade, pri nahrávaní celej sezóny pre všetky strediská, je potrebné spracovať vyše 160 000 requestov. Pri spracovaní je treba myslieť na rýchlu dostupnosť cenníkov v špecifickom formáte pre prevádzkovateľa webových stránok stredísk. Výsledkom je otestované a implementované riešenie, ktoré efektívne zredukovalo čas odozvy, či už pri príchode cenníkov, ale hlavne pri dopyte po cenách z približne 500 až 2000 ms na 10 až 30 ms.

Kľúčové slová: dynamické ceny, spracovanie, algoritmus.

Abstract

We did the bachelor's thesis for the company Tatry Moutain Resorts as. operator of tourist resorts in the region of Central and Eastern Europe. The company has implemented a tool for managing the utilization of resorts, dynamic ticket prices - Flexi prices. The first version of dynamic price lists from 2019, had a problem with a slow response to price demand and also with a slow response to receiving requests with a new price list. It therefore had to be redone. The aim of this bachelor thesis was to create a new version that will ensure the most efficient price processing. Prices need to be processed with a fast response time to the request and reflected these changes on the ticket prices screens at the resorts that are part of this application. At extreme case, more than 160,000 requests need to be processed when reciving an entire season for all resorts. When processing, it is necessary to think about fast availability of price lists in a specific format for the operator of the resorts' websites. The result is a tested and implemented solution that effectively reduces the response time on price requests arrival, but especially on the demand for prices from approximately 500 to 2000 ms to 10 to 30 ms.

Keywords: dynamic prices, processing, algorithm.

Obsah

Úvod **1**

1 Východiská **3**

1.1 Serverový jazyk . . . 3

1.1.1 Node.js . . . 3

1.2 Frameworky a moduly . . . 4

1.2.1 Express.js . . . 4

1.2.2 EJS . . . 6

1.2.3 Nodemailer . . . 6

1.2.4 Axios . . . 6

1.2.5 Bcryptjs . . . 6

1.2.6 Socket.io . . . 6

1.2.7 Compression . . . 6

1.3 Web server . . . 7

1.3.1 Nginx . . . 7

1.4 Databáza . . . 7

1.4.1 Sequelize . . . 8

1.5 Dynamické programovanie . . . 9

1.6 Dátové štruktúry . . . 9

1.6.1 Binárna halda . . . 10

1.6.2 Modul Priorityqueue . . . 10

1.7 Predchádzajúce riešenia . . . 10

1.7.1 Pevné cenniky . . . 10

1.7.2 Flexi prices v1 . . . 11

1.8 Exitujúce podobné systémy . . . 11

2 Ciele práce **13**

2.1 Prijatie cien . . . 13

2.1.1 Krajný prípad . . . 13

2.1.2 Dôležitosť postupného spracovania a rýchlosti odozvy . . . 14

2.2 Spracovanie a uloženie cien . . . 15

2.3 Notifikácia o zmene cien 18
2.4 Obrazovky v rezortoch 18

3 Návrh **19**
3.1 Algoritmus spracovania requesov 19
3.2 Spracovanie a uloženie cenníka 21
3.3 Databázový model 23
3.3.1 Tabuľka ticket 23
3.3.2 Tabuľka product 24
3.3.3 Tabuľka resort 24
3.3.4 Tabuľka season 24
3.3.5 Tabuľka idleDays 24
3.4 Crud endpointy 24
3.4.1 Prijatie ceny 24
3.4.2 Dopyt po cenách v špecifickom formáte 25
3.4.3 Verejny endpoint na dopyt po cenníkoch 25
3.4.4 Verejny endpoint na dopyt po obrazovke s cenníkom 25

4 Implementácia **27**
4.1 Trieda PricesStack 27
4.1.1 Vytvorenie binárnej haldy 27
4.1.2 Postupné spracovávanie requestov 27
4.1.3 Rekurzívne spracovávanie stackObjectov 28
4.2 Trieda PricesWrapper 29
4.2.1 Ukladanie ceny lístka 29
4.2.2 Dynamicke ukladanie cenníku v špecifickom formáte 31
4.3 Premietnutie cien 32
4.4 Obrazovka s cenníkmi 32
4.5 Testovanie 32
4.6 Nasadenie do prevádzky 33

Záver **35**

Zoznam obrázkov

1.1 Porovnanie asynchrónneho kódu so synchrónnym kódom 4
1.2 Spracovanie requestu pomocou middleware funkcii 5
1.3 Príklad routing 5
1.4 Ngnix schéma 8
1.5 ORM Sequelize 9

2.1 Data flow diagram 15

3.1 Algoritmus spracovania requesov 20
3.2 Binárna halda zobrazená stromom 21
3.3 Reprezentácia binárnej haldy v pamäti 21
3.4 Spracovanie a uloženie cenníka 22
3.5 Entitno-relačný model databázy 23

4.1 Obrazovka s cenníkmi 32
4.2 Odozvy na prijatie POST requestu s cenami 33
4.3 Odozvy na dopyt po cenách v špecifickom formáte 33

Zoznam tabuliek

4.1 Čas spracovávania haldy 33

Úvod

Bakalársku prácu robíme vo firme Tatry Moutain Resorts a.s. Firma Tatry Moutain Resorts a.s. je prevádzkovateľom turistických rezortov. Do jej portfólia patria napríklad horské strediská, hotely, aquaparky, zábavné parky a iné turistické služby v regióne strednej a východnej Európy.

Lyžiarske strediská majú obmedzené kapacity, ich vyťaženie je počas týždňa nerovnomerné, či už vplyvom počasia, dňa v týždni, sviatkov alebo prázdnin. Preto spoločnosť Tatry Moutain Resorts potrebovala nástroj na manažovanie vyťažeností stredísk. Týmto nástrojom je dynamická cena lístka, respektíve flexibilné cenníky lístkov – Flexi ceny. Toto riešenie by malo zabezpečiť rovnomernejšie rozptýlenie lyžiarov v strediskách. Prvá verzia dynamických cenníkov z roku 2019 mala problém s pomalou odozvou na dopyt po cenách a aj s pomalou odozvou na prijatie requestov s novým cenníkom. Preto sa ju spoločnosť rozhodla prerobiť.

Úlohou aplikácie je zabezpečiť čo najefektívnejšie spracovanie cien, s rýchlym časom odozvy na request a premietnuť zmenu v cenách do obrazoviek pomocou WebSocket technológie. Výpočet cien má na starosti firma tretej strany AXASOFT a budú prichádzať ako POST request v Json formáte. Cenníky pre viacero stredísk na celú sezónu však môžu mať veľmi veľký objem dát a prísť v tisíckach requestov. Pri spracovaní bolo treba myslieť na rýchlu dostupnosť cenníkov v špecifickom formáte pre spoločnosť NETSUCCESS, prevádkovateľa webových stránok stredísk. Aplikácia by mala obsahovať obrazovky na zobrazenie aktuálnych cenníkov v lyžiarskom stredisku.

Hlavná časť tejto bakalárskej práce je rozdelená do štyroch kapitol. V prvej kapitole opíšeme použité technológie. V druhej kapitole opíšeme scenár spracovania cien a taktiež čo všetko systém robí. V tretej kapitole opíšeme architektúru, databázový model a algoritmické riešenie efektívneho spracovania cien. Štvrtá kapitola opisuje proces implementácie, testovania a nasadenia do prevádzky.

Kapitola 1

Východiská

Na efektívne riešenie zadania tejto bakalárskej práce sú potrebné rôzne nástroje. V tejto kapitole si opíšeme a vysvetlíme použité nástroje a knižnice, odôvodníme si konkrétny výber, prípadne porovnáme s alternatívnymi možnosťami. V závere východiskovej kapitoly je opísaný aktuálny stav a existujúce podobné systémy.

1.1 Serverový jazyk

Webové aplikácie rastú na popularite, pretože sú ľahšie na naprogramovanie, udržiavanie a zabezpečenie. Taktiež sú ľahko prístupné pomocou webového prehliadača, ktorý nájdeme na väčšine zariadení s prístupom na internet. Takže nepotrebujú nič inštalovať a ani vyvíjať rôzne verzie pre rôzne operačné systémy.

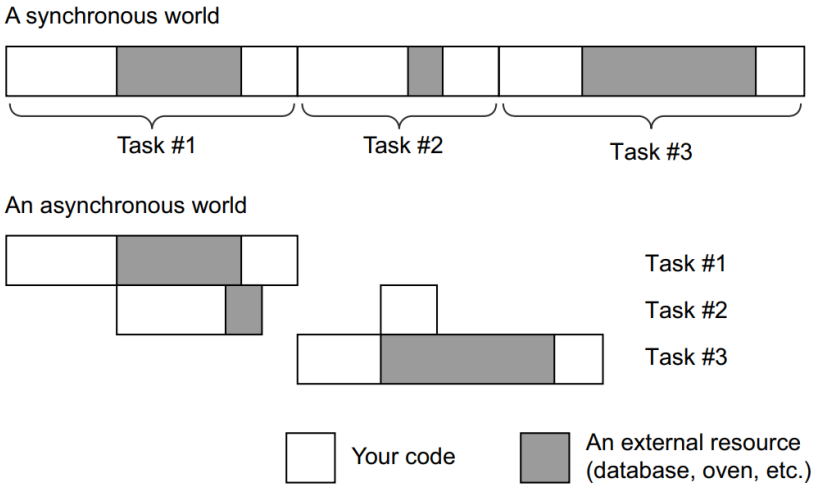
Medzi najdôležitejšie podmienky výberu serverového jazyka je schopnosť spracovať viacero používateľov efektívne, taktiež výkon a v neposlednom rade samotná ľahkosť implementácie. Rozhodovali sme sa medzi jazykmi PHP, Python a Node.js, čo je serverová implementácia JavaScriptu.

1.1.1 Node.js

Javascript nemusí operovať len vo webovom prehliadači, môžeme ho využiť aj ako každý serverový jazyk. Najrozšírenejšou takouto implementáciou je Node.js. Sekcia je spracovaná podla knihy [12]. Spĺňa teda Node.js naše podmienky na serverový jazyk?

Ako sme už vyššie spomenuli Node.js je serverová implementácia JavaScriptu. Je postavený na Google Chrome V8 JavaScript engine naprogramovanom v C++, ktorý je všeobecne známy svojim výkonom [14].

Ďalšou výhodou Node.js je jeho schopnosť spracovať viacero udalostí efektívne, pochádzajúca z asynchrónnosti Javascriptu. Javascript využíva takzvaný event loop teda cyklus udalostí, v praxi to znamená, že keď si prehliadač vyžiada request od nášho servera a počas toho nám príde ďalší request, pričom oba requesty potrebujú nejaké



Obr. 1.1: Porovnanie asynchrónneho kódu (napr Node.js) so synchrónnym kódom. Asynchrónny kód sa môže dokončiť oveľa rýchlejšie, hoci ho nikdy nevykonávame paralelne. Zdroj obrázku [12].

dáta z databázy, kým prvý request získava dáta, druhý sa môže začať tiež vybavovať. Možme to vidieť graficky znázornené na obrázku 1.1. Takéto možnosti nemajú iné riešenia defaultne implementované, častokrát na spracovanie viacero requestov naraz je potrebné nakúpiť dodatočný hardware.

V neposlednom rade zaujímavým modulom Node.js je framework Express.js, ktorý vie veľmi efektívne zjednodušiť programovanie backendu webovej aplikácie.

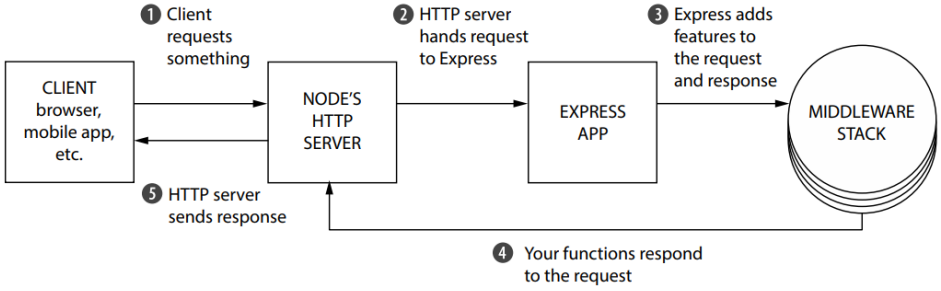
Node.js teda spĺňa všetky nami stanovené kritéria, preto je vhodný kandidát na využitie v tomto projekte.

1.2 Frameworky a moduly

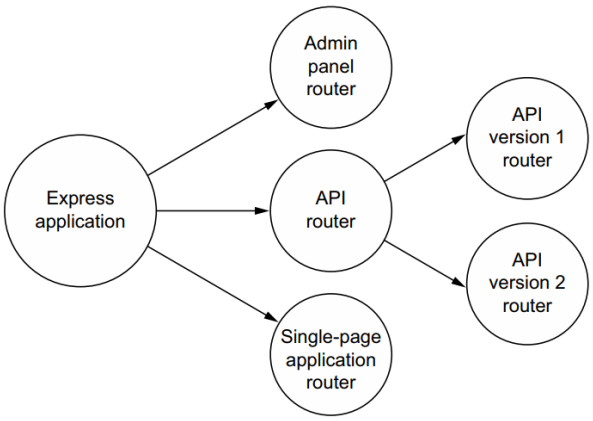
Node.js má k dispozícií veľké množstvo frameworkov a modulov tretích strán, ktoré sú často krát open source, s veľkou základňou aktívnych vývojárov. Taktiež sú vďaka častému využitiu dobre odladené a otestované. V tejto časti si opíšeme tie, ktoré sú vhodné na realizáciu našej aplikácie.

1.2.1 Express.js

Express je relatívne malý framework implementovaný v Node.js. Ako už má v názve pomáha s budovaním aplikácii v expresnom čase. Sekcia je spracovaná podla knihy [12]. Express.js pri budovaní API pridáva užitočnú funkcionalitu. Uľahčuje prácu na viacerých miestach a to hlavne organizáciu funkcionality aplikácii použitím takzvaných middleware funkcii, správu ciest API endpointov, takzvaný routing, pridáva užitočné funkcie na prácu s http protokolom a taktiež uľahčuje renderovanie dynamického



Obr. 1.2: Spracovanie requestu pomocou middleware funkcii. Zdroj obrázku [12].



Obr. 1.3: Diagram ukazujúci príklad routingu vo velkej aplikácii. Zdroj obrázku [12].

HTML. Ďalej bližšie špecifikujem túto funkcionalitu:

- Middleware – Node.js nám na spracovanie requestú dáva funkciu s ktorou môžeme ďalej pracovať. Request prichádza do tejto funkcie a response vraciame z tejto funkcie. Middleware je pojem, pod ktorým sa myslí to, že namiesto jednej veľkej request spracovávajúcej funkcie, máme viacero menších, z ktorých každá vykoná svoju menšiu časť roboty. Vzniká tak akási chain of responsibilities. Znázornené to možme vidieť na obrázku 1.2.

- Routing – Pri spracovaní requestov musíme brať do úvahy konkrétnu URL a http metódu na ktorú nám bol request poslaný. Routing v Express nám pomáha jednoducho nasmerovať URL cesty a http metódy k správnym funkciám. Na správne nasmerovanie používame takzvané routery po anglicky routers. Pri velkých aplikáciách je možné na seba nasmerovať viacero routerov. Vzniká tak prehľadná stromová štruktúra, ako to možme vidieť na obrázku 1.3.

Express samozrejme nefunguje samostatne. Súčasťou Node.js ekosystému sú rôzne ďalšie užitočné moduly tretej strany, ktoré s ním spolupracujú.

1.2.2 EJS

EJS, teda embedded JavaScript je jeden z najjednoduchších a najpopulárnejších zobrazovacích enginov. Dovoľuje nám generovať dynamicky vyplnené HTML pomocou Javascriptu. Má jednoduchú syntax pomocou EJS tagov ako napríklad <% %> pre vykonanie Javascriptu vo vnútry tagu, alebo <%= %> pre vloženie hodnoty v tagu do html. Taktiež podporuje vkladanie ďalších ejs súborov [5].

Funkcionalita je znázornená na príklade 1.1, kde sa dynamicky vyplní meno a vloží súbor bio.ejs.

```
1 var obj = {
2 name:"Adam"
3 }
4 <\% if(obj.name){ \%>
5   <h1> <\%= obj.name \%> </h1>
6 <\% } \%>  >
7 <\%- include("bio" ) \%>
```

Kód 1.1: Príklad EJS

1.2.3 Nodemailer

Nodemailer je modul pre Node.js aplikácie, ktorý umožňuje jednoduché posielanie mailov [8].

1.2.4 Axios

Axios je modul pre node.js, krorý umožňuje jednoduché http requesty [1].

1.2.5 Bcryptjs

Bcryptjs je modul ktorý umožňuje zašifrovať dáta, napríklad heslá. Používa šifrovaciu metódu bcrypt s použitím takzvanej soli, po anglicky salt. Je velmi odolný aj proti brute force útokom [2].

1.2.6 Socket.io

Socket.io je modul, ktorý dovoluje obojstrannu komunikáciu medzi serverom a prehliadačom v reálnom čase [11].

1.2.7 Compression

Compression je modul, ktorý dokáže zmenšiť velkosť tiel responsov. Používa sa ako middleware funkcia, a s jeho využitím vieme zmenšiť objem dát a zvýšiť rýchlosť spra-

covania requestov [3].

1.3 Web server

Server je počítač komunikujúci s ostatnými počítačmi, ktorým odosiela požadované informácie. Tieto počítače sa nazývajú klienti. Web server je server pripojený na internet, ktorý pomocou http protokolu prijíma requesty od klientov, ako napríklad internetový prehliadač, a odosiela http odpovede ako napríklad HTML stránku alebo dáta v JSON formáte ako napríklad pri volaní API. Táto sekcia je spracovaná podľa stránky [6].

Hoci Node.js aplikácia s využitím Express.js frameworku vie fungovať ako samostatný web server, s nasadením webovej aplikácie do produkcie sa oplatí mať osobitný web server, na ktorom našu Node.js webovú aplikáciu spustíme. V súčasnosti najpoužívanejšími riešeniami pre web servery sú Apache a Nginx. Z toho druhý menovaný je síce menej používaný, ale plynulo vytláča konkurenta a rastie na podiely na trhu.

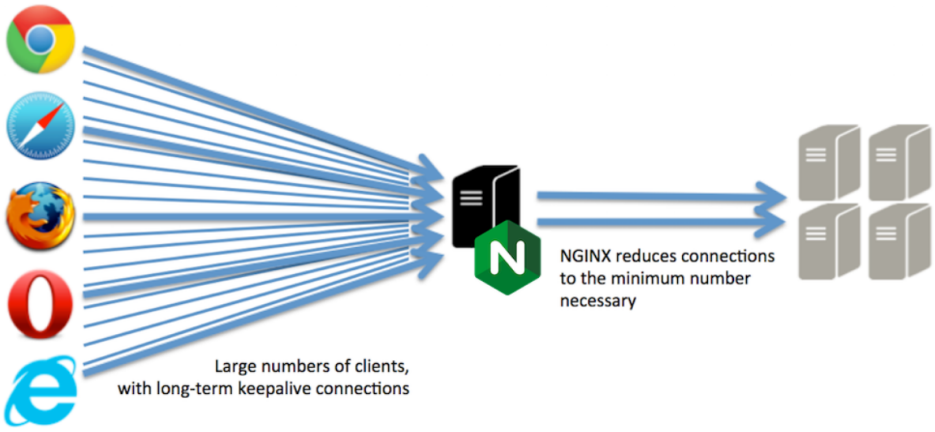
1.3.1 Nginx

Nginx je http a proxy server, ktorý sa využíva na viacero úloh a dokáže zvýšiť výkonnosť Node.js aplikácie, a to s využitím na:

- Reverzný proxy server – Keď sa návštevnosť webovej aplikácie zvyšuje, najlepším prístupom na zlepšenie výkonu je použitie NginX ako reverzného proxy servera pred serverom Node.js na vyrovnávanie prenosu medzi servermi, ako môžme vidieť na obrázku 1.4. Toto je hlavný prípad použitia NginX v aplikáciách Node.js.
- Vyvažovanie záťaže – zlepšuje výkon a zároveň znižuje záťaž na backendové služby odosielaním požiadaviek klientov, ktoré má splniť ktorýkoľvek server s prístupom k požadovanému súboru.
- Ukladanie statického obsahu do vyrovnávacej pamäte – Poskytovanie statického obsahu v aplikácii Node.js a používanie NginX ako reverzného proxy servera zdvojnásobuje výkon aplikácie.

1.4 Databáza

Každá aplikácia potrebuje spôsob na uloženie perzistentných údajov do pevnej pamäte a následne, keď ich treba, čo najefektívnejšie načítanie späť do operačnej pamäte. Za týmto účelom existujú databázové systémy. Sekcia je spracovaná podla knihy [12]. Na výber máme medzi relačnými databázami takzvané relational databases, alebo nerelačnými databázami takzvanými non-relational databases:



Obr. 1.4: Využitie Ngnix web servera na zvýšenie výkonnosti Node.js aplikácie. Zdroj obrázku [7].

- Relačné databázy sa podobajú tabuľkám. Dáta sú štruktúrované a každý záznam je v podstate riadok tabulky. Väčšina relačných databáz sa dopytujú nejakov formou SQL jazyka čo je skratka pre structured query language, teda štruktúrovaný dopytovací jazyk. V Node.js je najrozsiahlejší a najpoužívanejší modul na prácu s relačnými databázami Sequelize.
- Nerelačné databázy sú odlišné od relačných, nie sú štruktúrované ako tabuľky. Sú menej robustné, nemajú stĺpce ani tabuľky. Majú kolekcie, čo je niečo ako súbor dokumentov respektíve zoznam. Záznamy sú potom dokumenty v kolekcii. Dokumenty nemajú nejakú špecifickú štruktúru ako napríklad stĺpce v tabuľke. V Node.js je najrozsiahlejší a najpoužívanejší modul na prácu s relačnými databázami Mongoose.

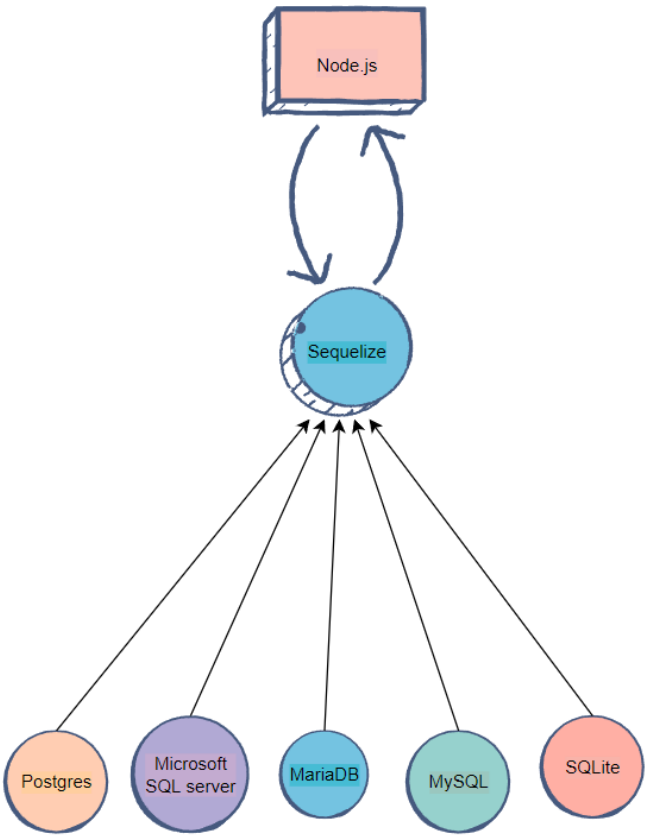
Z týchto možností sme pre našu aplikáciu vybrali Relačnú databázu, z dôvodu našich predchádzajúcich skúsenosti s SQL jazykmi najmä s PostgreSQL, preto ako SQL jazyk sme si vybrali pravé tento.

1.4.1 Sequelize

Sequelize je ORM(Object–relational mapping) pre Node.js umožňujúci komunikáciu s viacerými SQL jazykmi, ako môžeme vidieť na obrázku 1.5. Podporuje transakcie, relácie, migrácie, lazy loading aj eager loading [9].

Modelovanie databázy pomocou Sequelize je jednoduché, model v sequelize reprezentuje vlastne tabuľku v databáze. Pri definovaní modelu môžeme zadať rôzne parametre ako typ, dovolenie null hodnoty, primárny kľuč, unikátnosť hodnoty. Samotný model následne za nás vygeneruje samotné SQL.

Podobne aj pri dopytoch do databázy stačí z referencie na model zavolať jednu z funkcii ako napríklad create, find, findAll, update. Taktiež môžeme pridať parametre



Obr. 1.5: Komunikácia Sequelize s podporovanými SQL jazykmi. Zdroj obrázku [10].

ako atribúty, agregácie, zgrupovanie, where klauzulu a podobne. Následne modul za nás vygeneruje SQL a vykoná dopyt do databázy.

1.5 Dynamické programovanie

Na spracovanie veľkého množstva cien, potrebujeme optimálne riešenie, použijeme teda metódu dynamického programovania:

Kľúčom k zníženiu množstva práce, ktorú robíme, je zapamätať si niektoré z minulých výsledkov, aby sme sa vyhli prepočítavaniu výsledkov, ktoré už poznáme. Jednoduchým riešením je uložiť výsledky do tabuľky, keď ich nájdeme. Potom predtým, ako vypočítame nové výsledky, najprv skontrolujeme tabuľku, či už nie je známy výsledok. Ak už v tabuľke existuje výsledok, použijeme namiesto prepočítania hodnotu z tabuľky [13].

1.6 Dátové štruktúry

Pri príchode cenníkov ich potrebujeme spracovať postupne podľa priority. Na toto spracovanie potrebujeme implementáciu prioritného frontu. Sekcia je spracovaná podla

knihy [13].

1.6.1 Binárna halda

Binárna halda je implementácia prioritného frontu, ktorá využíva dátovú štruktúru halda. Halda je binárny strom, ktorý musí spĺňať tieto dve podmienky:

- Každý vrchol (okrem koreňa) má hodnotu kľúča väčšiu alebo rovnú ako hodnota kľúča v jeho predkovi, takže v koreni je minimálna hodnota kľúča z celého stromu, teda prvok s najväčšou prioritou.

- Je to skoro úplný binárny strom, a to preto, že okrem najnižšej úrovne sú všetky úrovne úplné a v najnižšej úrovni sú všetky vrcholy umiestňované len zľava.

Pri spracovaní cenníkov podla priority budeme potrebovať tieto metódy:

- Pridanie prvku - prvok sa pridá na koniec haldy. Následne sa povymieňa v strome smerom nahor a to až kým nenatrafi na menšieho predka alebo sa dostane až do koreňa stromu.

- Odobranie prvku s najväčšou prioritou - to je prvok v koreni haldy. Následne sa koreň nahradí posledným prvkom v halde a povymieňa sa v strome smerom nadol a to až kým vrchol už nemá ani jedného syna, alebo obaja synovia už nemajú menšiu hodnotu ako prvok.

1.6.2 Modul Priorityqueue

Modul priorityqueue implementuje binárnu haldu v JavaScripte ako BinaryHeap(comparator) - comparator definuje porovnanie, podľa ktorého sa prvky v binárnej halde budú usporadúvať. Modul má implementované obe potrebné metódy a to push() na pridanie prvku a pop() na odobranie prvku s najväčšou prioritou.

1.7 Predchádzajúce riešenia

1.7.1 Pevné cenniky

V minulosti sa v lyžiarskych strediskách spoločnosti Tatry Moutain Resort používaju pevné cenniky. Sezóna je väčšinov rozdelena na tri časti a v nich je pevne nastavena cena listkov, v zavislosti na ocakavanu vyťaženosť strediska v danej časti.

1.7.2 Flexi prices v1

Prvá verzia dynamických cenníkov z roku 2019. Táto verzia mala problém s pomalou odozvou na dopyt po cenách a aj s pomalou odozvou na prijatie requestu s novým cenníkom. Bolo nutné ju preto prerobiť.

1.8 Exitujúce podobné systémy

Dynamická tvorba cien je cenová stratégia, pri ktorej podniky stanovujú flexibilné ceny produktov alebo služieb na základe aktuálnych požiadaviek trhu. Podniky sú schopné meniť ceny na základe algoritmov, ktoré zohľadňujú ceny konkurentov, ponuku a dopyt a ďalšie vonkajšie faktory na trhu. Dynamická tvorba cien je bežnou praxou vo viacerých odvetviach, ako je pohostinstvo, cestovný ruch, zábava, maloobchod, elektrina a verejná doprava. Každé odvetvie pristupuje k dynamickej tvorbe cien trochu inak, na základe svojich individuálnych potrieb a dopytu po produkte. Tu je niekoľko spoločností, ktoré majú podobný systém implementovaný: [4]

- Ticketmaster - Dynamické ceny za koncerty a iné podujatia sú často založené na dopyte trhu a je to stratégia, ktorú veľké spoločnosti ako Ticketmaster používajú už roky.

- Tímy NFL - Keďže 16 tímov NFL prešlo z pevných cien, je zrejmé, že dynamické stanovovanie cien v športe je trendom v celom odvetví. Dnes sa univerzitné a profesionálne športové tímy spoliehajú na rôzne modely pri obsadzovaní miest v závislosti od počasia, dátumu alebo počtu zľavnených miest. Niektoré tímy tiež pracujú na maximalizácii výnosov znížením počtu zľavnených a bezplatných vstupeniek. Toto rozhodnutie im pomohlo pomaly zvyšovať základňu držiteľov sezónnych vstupeniek a prilákať verných fanúšikov, ktorí sú ochotní zaplatiť viac.

Kapitola 2

Ciele práce

V tejto kapitole sa opisuje scenár spracovania cien a taktiež čo všetko systém robí. Aplikácia má na starosti digital signage – teda zobrazovanie na obrazovkách v strediskách. Aplikácia sa integruje do dlhodobo bežiacich systémov, a preto sa musí im prispôsobiť, či už na vstupných, ale aj výstupných údajoch.

2.1 Prijatie cien

Ceny prichádzajú ako POST request v Json formáte od spoločnosti AXASOFT. Cenníky však môžu mať veľmi veľký objem dát.

2.1.1 Krajný prípad

Spoločnosť Tatry Moutain Resorts prevádzkuje momentálne 9 stredísk a v budúcnosti budú určite ďalšie pribúdať. Každá cena má 4 typy:

1. Online – cena za ktorú si lístok môžeme nakúpiť online cez web `https://www.gopass.travel/`.
2. AOM – cena za ktorú si lístok môžeme nakúpiť v samoobslužnom kiosku priamo v stredisku.
3. Pokladňa – cena za ktorú si lístok môžeme nakúpiť na pokladni priamo v stredisku.
4. B2B – business to business cena, za túto cenu môžu lístky kúpiť business partneri, ako napríklad hotely.

Každá cena prichádza pod 5 kategóriami:

1. Dospelý

2. Junior
3. Senior
4. Dieťa
5. Študent

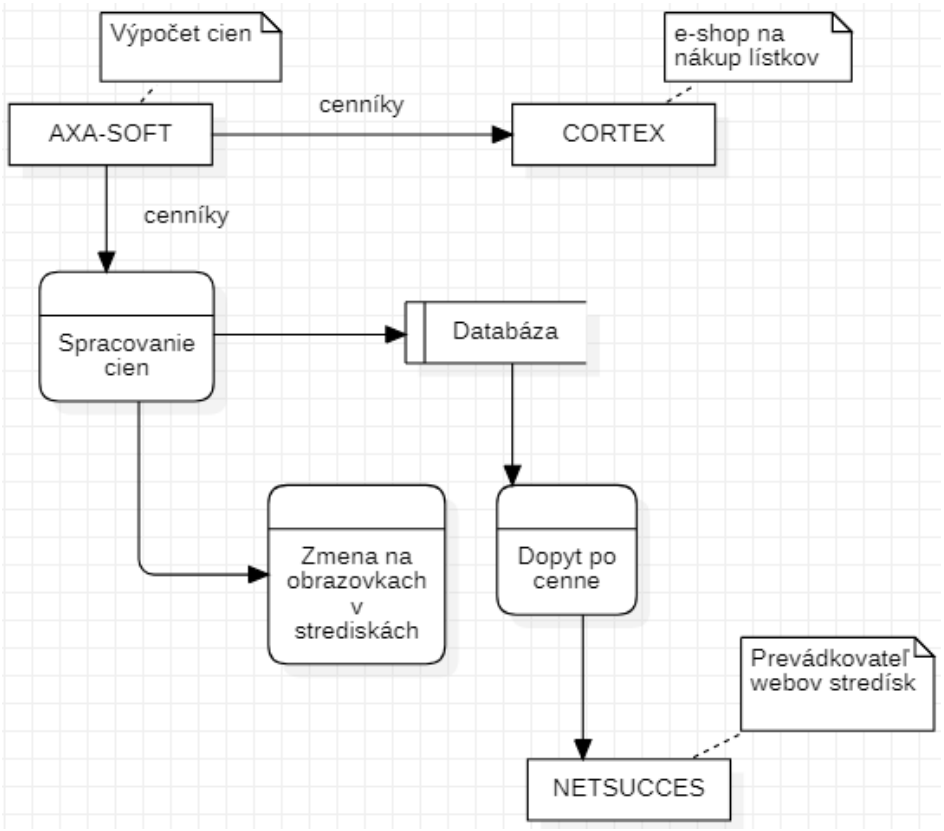
Každá cena prichádza pre 1 až 6 dňový lístok.

Krajný prípad nastáva vždy pred začatím novéj sezóny a je ním nahratie celej sezóny, čo je v priemere 150 dní. Podľa výpočtu 2.1 vychádza, že okrajový prípad je spracovať 162 000 cien pri aktuálnom počte stredísk. Dĺžka sezón je pre rôzne strediská rôzna. V stredisku s ľadovcom, kde sa lyžuje celoročne, to môže byť až 365 dní.

$$150 * 9 * 4 * 6 * 5 = 162000 \tag{2.1}$$

2.1.2 Dôležitosť postupného spracovania a rýchlosti odozvy

AXASOFT okrem našej aplikácie, ktorá má na starosti digital signage – teda zobrazovanie na obrazovkách v strediskách, posiela ceny aj spoločnosti CORTEX, ktorá má na starosti vyššie spomínaný eshop https://www.gopass.travel/, na ktorom sa nakupujú lístky online. Nesmie sa stať, že by zákazník videl inú cenu na našich obrazovkách, webových stránkach stredísk prevádzkovaných spoločnosťou NETSUCCESS, alebo v online eshope prevádzkovanom spoločnosťou CORTEX. Spoločnosť NETSUCCESS taktiež potrebuje dáta v konkrétnom formáte. Preto je rýchlosť odozvy na jej dopyt mimoriadne dôležitá. Vzťah medzi spoločnosťami je zobrazený na data flow diagrame 2.1, kde v obdĺžnikoch sú zobrazené externé entity, v oválach procesy v našej aplikácii a obdĺžnik s čiarou je úložisko dát.



Obr. 2.1: Data flow diagram, zobrazujúci externé spoločnosti AXASOFT a NETSUCCES, procesy v aplikácii a databázu

2.2 Spracovanie a uloženie cien

Najdôležitejším cieľom aplikácie je spracovanie prichádzajúcich cien. Ceny prichádzajú ako telo requestu so štruktúrov zobrazenom na príklade 2.1, na ktorom sú z bezpečnostných dôvodov niektoré dáta nahradené podtržníkom. Každý request teda má kľúč day, ktorý hovorí o tom, na ktorý deň cena je. Potom má kľúč products ktorého hodnota je zoznam produktov. Väčšinou ich je 6, čo reprezentuje 1 až 6 dňový lístok. Každý produkt má svoje ProductId a obsahuje zoznam s cenami pre každú vyššie spomínanú kategóriu. Online cena je pod kľúčom price, a cena na pokladni je pod kľúčom offlinePrice. V každom requeste teda príde cenník na jeden deň, pre jedno stredisko a pre jeden sales channel teda typ ceny.

```
1 {
2       day: '_______',
3       products: [
4         { ProductId: ___, categories: [Array] },
5         {
6           ccProductId: ___,
7           categories: [
8             {
```

```
9                categoryId: '__',
10               price: 4,
11               defaultPrice: 145,
12               offlinePrice: 2599,
13             },
14             {
15               categoryId: '__',
16               price: 95,
17               defaultPrice: 116,
18               offlinePrice: 1647,
19             },
20             {
21               categoryId: '__',
22               price: 50,
23               defaultPrice: 116,
24               offlinePrice: 8979,
25             },
26             {
27               categoryId: '__',
28               price: 93,
29               defaultPrice: 102,
30               offlinePrice: 1894,
31             },
32             {
33               categoryId: '__',
34               price: 88,
35               defaultPrice: 116,
36               offlinePrice: 5803,
37             },
38           ],
39         },
40         { ccProductId: ___, categories: [Array] },
41         { ccProductId: ___, categories: [Array] },
42         { ccProductId: ___, categories: [Array] },
43         { ccProductId: ___, categories: [Array] },
44       ],
45       received: '2021-11-09T08:03:27.300Z',
46 }
```

Kód 2.1: Príklad jedného tela requestu s cenami

Predchádzajúce riešenie spracovania ukladalo cenník v podobnej štruktúre ako prišla, čo však pri skladaní cien do konkrétneho formátu 2.2 na weby spoločnosti NETSUCCESS spôsobovalo dlhý čas odozvy. Treba preto navrhnúť a implementovať vhodný system na uloženie cien do aplikácie.

Pri importe viacerých dní sa teda stane to, že príde viacero requestov s novými

cenami, kým ešte tie predchádzajúce neboli spracované. Treba ich preto spracovať podla priority. Najvyššiu prioritu pri spracovaní majú ceny na súčasný deň. Priorita spracovania klesá smerom do budúcnosti. Bez priority sú dni v minulosti alebo dni kedy má stredisko zavreté, takzvané idle days.

```
{
      "resort":  '__',
      "JSON": [
      {
         "day":  '______',
         "products": [
           {
             "ccProductId":  '__',
             "categories": [
               {
                 "categoryId": "__",
                 "price": 23
               },
               {
                 "categoryId": "__",
                 "price": 42
               },
               {
                 "categoryId": "__",
                 "price": 40
               },
               {
                 "categoryId": "__",
                 "price": 40
               },
               {
                 "categoryId": "__",
                 "price": 40
               }
             ]
           },
           { "ccProductId":  '__', "categories": [Array] },
           ...
           { "ccProductId":  '__', "categories": [Array] }
         ]
       },
      { "day":  '______', "products": [Array] },
       ...
      { "day":  '______', "products": [Array] },
    ]
}
```

Kód 2.2: Konkrétny formát, v ktorom potrebujú dáta weby spoločnosti NETSUCCESS. A to zoznam dní, kde každý má zoznam produktov(typov lístkov), kde každý má zoznam cien pre každú z 5 kategórii.

2.3 Notifikácia o zmene cien

Po spracovaní cien treba zobraziť nové ceny v reálnom čase na strediskových obrazovkách pomocou WebSocket technológie, aby nemohla nastať situácia, že si zákazník kúpi lístok za inú cenu, ako je zobrazená na strediskových obrazovkách.

2.4 Obrazovky v rezortoch

Aplikácia obsahuje obrazovky do stredísk zobrazujúce aktuálne ceny lístkov pre zvolený rezort a deň. Zobrazené ceny sú pre 1 až 6 dňový lístok a 3 kategórie: dospelý, junior & senior a dieťa.

Kapitola 3

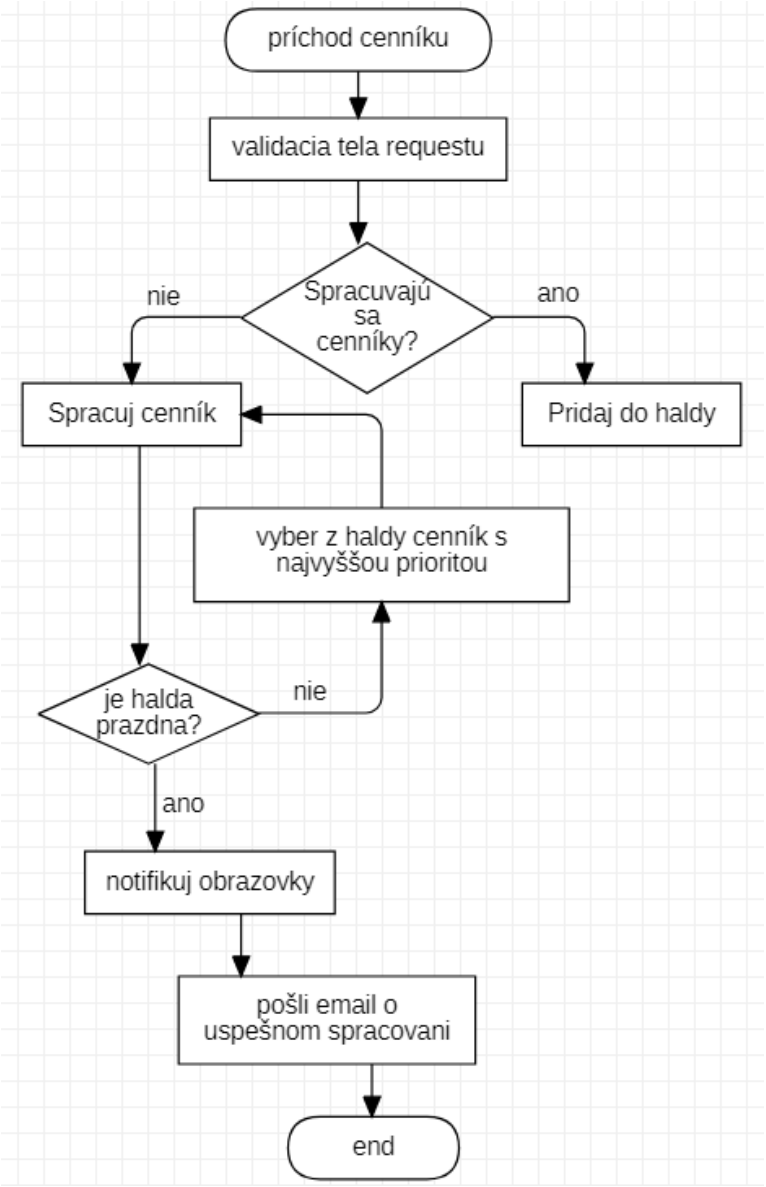
Návrh

V tejto kapitole si opíšeme architektúru, databázový model, algoritmické riešenie efektívneho spracovania cien.

3.1 Algoritmus spracovania requesov

Ako sme už v kapitole o cieľoch práce spomínali requesty s cenníkmi spracúvame podľa priority. Najvyššiu prioritu pri spracovaní majú ceny na súčasný deň. Priorita spracovania klesá smerom do budúcnosti. Bez priority sú dni v minulosti alebo dni kedy má stredisko zavreté, takzvané idle days. Využijeme preto implementáciu prioritného frontu a to binárnu haldu. Algoritmus spracovania requestu s cenami je nasledovný:

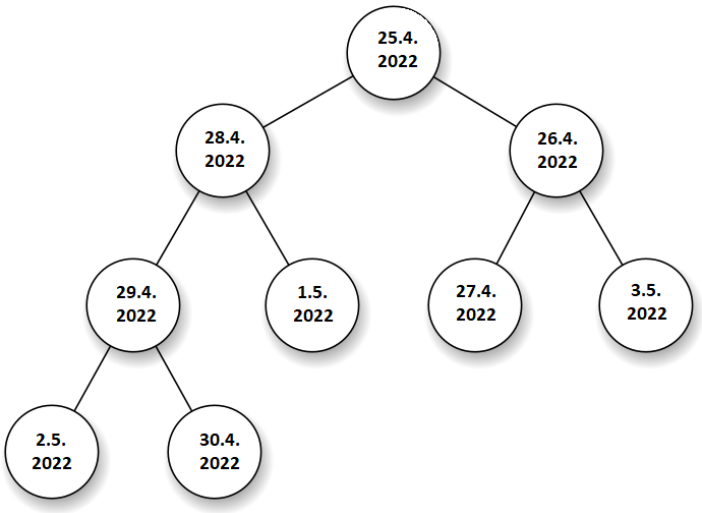
1. Príchod requestu s cenníkom.
2. Validácia správnosti requestu.
3. Ak sa ceny spracovávajú, to znamená, že halda nie je prázdna alebo sa nejaký cenník ešte spracúva, pridá sa cenník do haldy.
4. Ak sa ceny nespracovávajú, tak sa cenník môže spracovať, samotné spracovanie si vysvetlíme v ďalšej sekcii.
5. Po spracovaní sa pozriem do haldy, či je prázdna.
6. Ak halda nie je prázdna, vyberieme z haldy cenník v koreni, to je ten s najvyššou prioritou. A vraciame sa do kroku 4. v ktorom vybratý cenník spracujeme. Vzniká tu teda rekurzia.
7. Ak halda je prázdna skončili sme so spracovávaním. Je to teda triviálny prípad a rekurzia končí.
8. Následne treba zobraziť nové ceny v reálnom čase na strediskových obrazovkách pomocou WebSocket technológie.



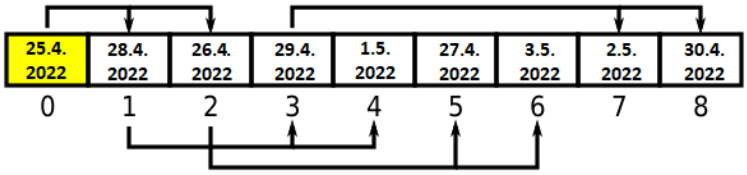
Obr. 3.1: Algoritmus postupného spracovania requesov podľa priority.

9. Nakoniec sa notifikuje mailom na určené adresy, že import prebehol úspešne.

Algoritmus je zobrazený na diagrame 3.1. Binárna halda je zobrazená stromom na obrázku 3.2, za predpokladu, že dnešný deň je 25.4.2022. V koreni je prvok s najvyššou prioritou. Synovia každého prvku majú prioritu nižšiu, teda dátum cenníka je neskôr. Binárna halda je v pamäti reprezentovaná ako zoznam. Vizualizácia je na obrázku 3.3.



Obr. 3.2: Binárna halda zobrazená stromom, za predpokladu, že dnes je 25.4.2022.



Obr. 3.3: Reprezentácia binárnej haldy v pamäti.

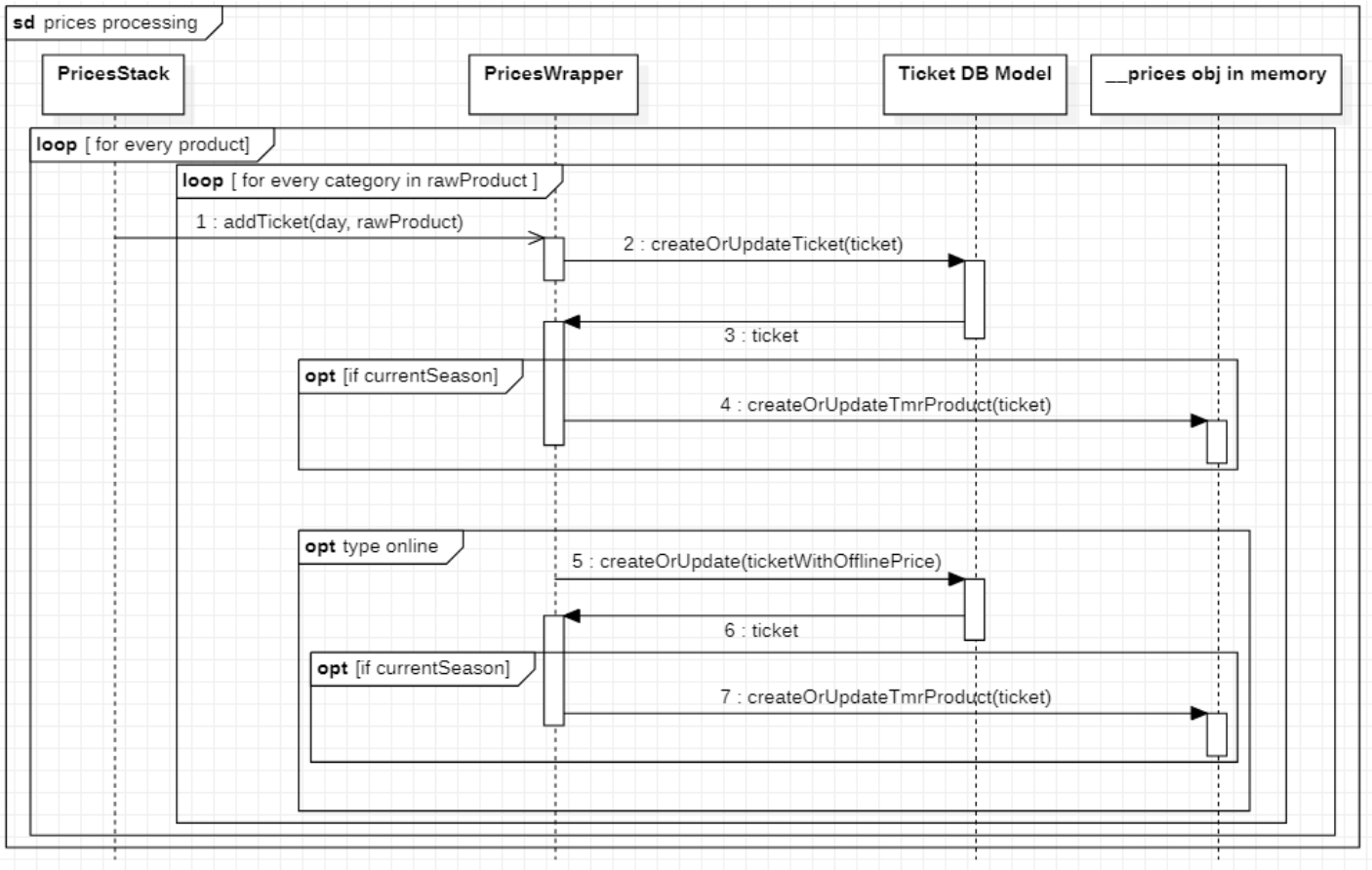
3.2 Spracovanie a uloženie cenníka

Pri spracovaní ceny si uložíme ceny do databázy. Online cena a offline, teda pokladničná cena, sú od seba závislé a preto chodia spolu. Musíme to teda v prípade príchodu ceny typu online brať do úvahy. Okrem toho však musíme vyriešiť problém predchádzajúcej verzie, a to dlhý čas odpovede na dopyt po cenníku od stránok stredísk prevádzkovaných spoločnosťou NETSUCCESS v ich špecifickom formáte. Riešením je dynamické programovanie. Aplikácia si bude pomocou triedy PricesWrapper v pamäti udržiavať __prices objekt, do ktorého sa budú dynamicky ukladať všetky prijate ceny v formáte, ktorý očakáva pri dopyte spoločnosť NETSUCCESS. Takto sa pri dopyte stačí pozrieť do __prices objektu a okamžite môžeme vrátiť aktuálny cenník. Toto spracovanie je zobrazené na sekvenčnom diagrame 3.4 s vysvetlením krokov:

1. Z haldy pri spracovaní cenníka v cykle príde do triedy PricesWrapper rawProduct, čo je cenník na jeden deň, pre jedno stredisko, pre jeden druh (1 - 6 dňový).

2. V cykle sa pre každú kategóriu (dieťa, dospelý...) buď vytvorí alebo prepíše cena lístka v databáze.

3. Databáza vráti lístok.



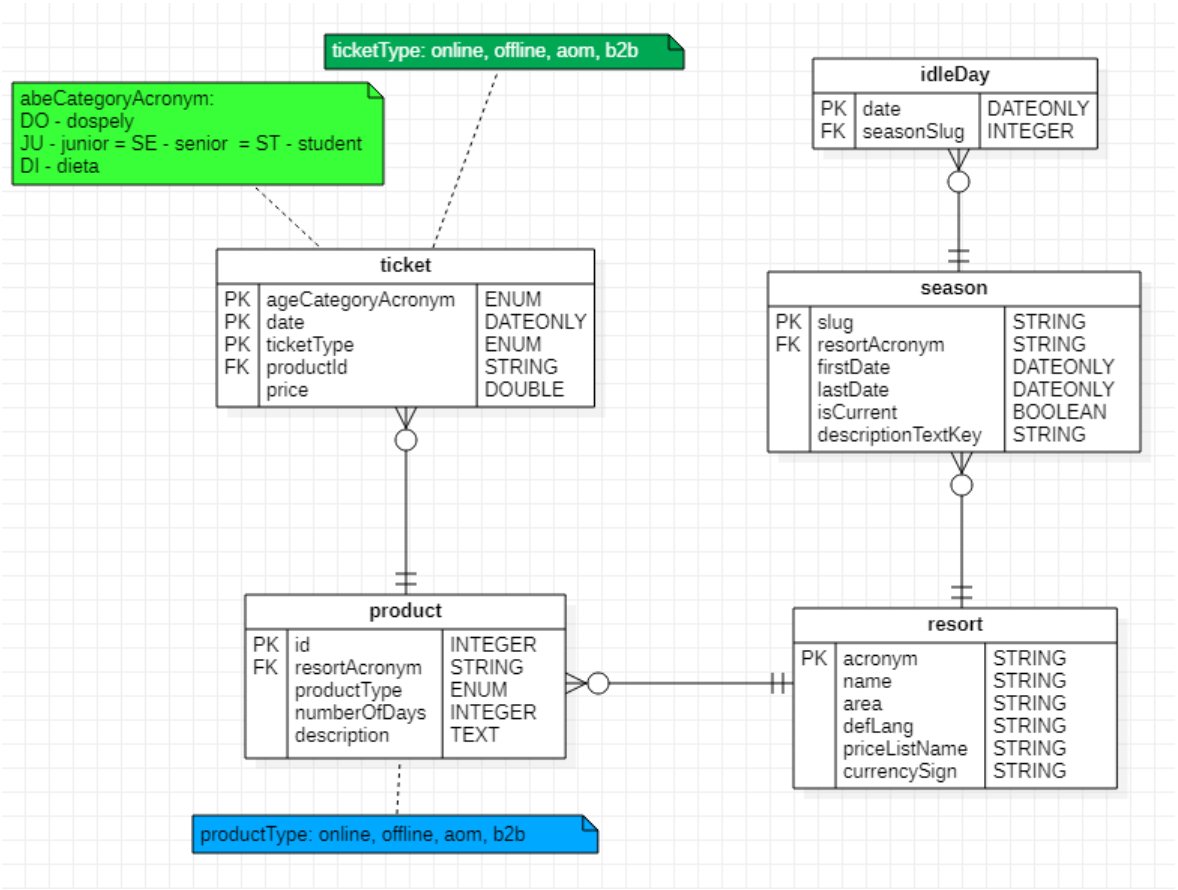
Obr. 3.4: Sequenčný diagram spracovania a uloženia cenníka.

4. Ak je lístok z aktuálnej sezóny, lístok sa uloží do __prices objektu, vo špecifickom formáte pre NETSUCCESS.

5. Ak je lístok typu online, zoberie sa nie cena ale offline cena a taktiež sa buď vytvorí alebo prepíše cena v databáze.

6. Databáza vráti lístok s offline cenou.

7. Znova ak je lístok z aktuálnej sezóny, lístok s offline cenou sa uloží do __prices objektu, vo špecifickom formáte pre NETSUCCESS.



Obr. 3.5: Entitno-relačný model databázy.

3.3 Databázový model

Ceny treba po spracovaní uložiť. Musíme preto navrhnúť a vytvoriť databázu a v nej uchovať potrebné dáta. Entitno-relačný model databázy si môžeme pozrieť na obrázku 3.5.

3.3.1 Tabuľka ticket

V tabuľke ticket ukladáme jednotlivé ceny. Primárny kľúč je zložený z troch stĺpcov:

1. ageCategoryAcronym – veková kategória lístka. Je typu ENUM a teda može mať hodnoty DO, JU, SE, ST.DI,
2. date – deň, na ktorý cena je. Je typu DATEONLY.
3. ticketType – typ lístka. Je typu ENUM a teda može mať hodnoty onnline, offline, aom, b2b.
4. Ak sa ceny nespracovávajú, tak sa cenník môže spracovať, samotné spracovanie si vysvetlíme v ďalšej sekcii.

Ďalej má cudzí kľúč productID, ktorým sa cena odkazuje na svoj produkt. Jeden produkt má viacero cien. A nakoniec stĺpec price so samotnou cenou.

3.3.2 Tabuľka product

V tabuľke product ukladáme produkty, na ktoré sa pri prijatí cenníkov odkazuje kľuč ccProductId. Primárny kľúč je práve toto id. Ďalej má cudzí kľuč resortAcronym ktorým sa produkt odkazuje na svoj rezort. Jeden rezort má viacero produktov. Zvyšné stĺpce sú:

1. productType – typ produktu. Je typu ENUM, a teda môže mať hodnoty online, aom, b2b. Sú to podobné typy ako v tabuľke ticket, avšak bez typu offline, je to preto, lebo keď ceny prichádzajú pri každom z týchto ccProductId je aj online aj offline cena.
2. numberOfDays – počet na koľko dní je lístok. Je typu INTEGER.
3. description – slovný popis lístku. Je typu TEXT.

3.3.3 Tabuľka resort

V tabuľke resort sú uložené rezorty. Primárnym kľúčom je stĺpec acronym typu STRING. Ďalej stĺpce name, area, defLang, priceListName, currencySign, všetky typu STRING.

3.3.4 Tabuľka season

V tabuľke season sú uložené sezóny. Primárnym kľúčom je stĺpec slug typu STRING. Ďalej má cudzí kľuč resortAcronym ktorým sa sezóna odkazuje na svoj rezort. Jeden rezort má viacero sezón. Ďalej firstDate a lastDate typu DATEONLY, isCurrent typu BOOLEAN a descriptionTextKey typu STRING.

3.3.5 Tabuľka idleDays

Tabuľka idleDays obsahuje dni, kedy je stredisko zatvorené. Primárny kľúč je stĺpec date typu DATEONLY. Cudzí kľúč seasonSlug sa odkazuje na tabuľku season. Jedna sezóna môže mať viacero idleDays.

3.4 Crud endpointy

3.4.1 Prijatie ceny

Prijatie ceny z bezpečnostných dôvodov nezverejňujeme keďže je určený len pre spoločnosť AXA-SOFT.

3.4.2 Dopyt po cenách v špecifickom formáte

Dopyt po cenách v špecifickom formáte pre spoločnosť NETSUCCESS z bezpečnostných dôvodov nezverejňujeme, keďže je určený len pre spoločnosť NETSUCCESS.

3.4.3 Verejny endpoint na dopyt po cenníkoch

Verejný endpoint na dopyt po cenníkoch je GET request na adresu s cestou "/api/v2/best-prices/:resortAcronym". Route parameter resortAcronym určuje, pre ktorý resort ceny chceme. Query parametre sú nasledovné:

- firstDate - dátum, od ktorého chceme ceny vo formáte YYYY-MM-DD. Defaultne sa nastaví na zajtrajší deň.
- countPrices - počet cien ako číslo. Defaultne sa nastaví na 3.
- countDays - počet dní, od firstDate, na ktoré chceme ceny. Defaultne sa nastaví na 14.
- priceCategory - veková kategória lístkov ako jedno z [DO, JU, SE, ST, DI]. Defaultne sa nastaví na DO.
- ticketType - typ lístku, podľa počtu dní od 1 do 6. Defaultne sa nastaví na 1.
- salesChannel - typ predajného kanálu lístku ako jedno z [online, aom, offline, b2b]. Defaultne sa nastaví na online.

3.4.4 Verejny endpoint na dopyt po obrazovke s cenníkom

Verejný endpoint na dopyt po obrazovke s cenníkom je GET rquest na adresu s cestou "/api/v2/best-prices/:resortAcronym". Route parameter resortAcronym určuje, pre ktorý resort ceny chceme. Query parametr je iba date - dátum, na ktorý chceme zobraziť ceny na obrazovky, vo formáte YYYY-MM-DD. Defaultne sa nastaví na dnešný deň.

Kapitola 4

Implementácia

V tejto kapitole si opíšeme zaujímavé časti kódu, testovanie aplikácie a nasadenie do prevádzky.

4.1 Trieda PricesStack

Trieda PricesStack má na starosti algoritmus postupného spracovania prichádzajúcich requestov. Ktorého dôležité útržky kódu vysvetlíme v nasledujúcich sekciách. V triede sa taktiež ukladajú štatistické dáta typu ako dlho sa ceny spracovávali, aká hlboká halda vznikla a podobne. Ktoré sa neskôr využívajú v štatistických reportoch.

4.1.1 Vytvorenie binárnej haldy

V konštruktore triedy vytvoríme binárnu haldu z modulu priorityqueue. Na porovnávanie priority jej posunieme comparator, ktorý najskôr porovnáva či je cena na dnešný deň. Následne porovnáva podľa toho ktorá cena je v čase neskôr. Vytvorenie tejto binárnej haldy si možte pozrieť na časti kódu 4.1.

```
1 this.requestStack = new BinaryHeap({
2         comparator: (a, b) => {
3           if (a.stackPriority.isSame(moment(), 'day')) return 1;
4           return a.stackPriority.isAfter(b.stackPriority) ? -1 : 1;
5         },
6       });
```

Kód 4.1: Vytvorenie binárnej haldy

4.1.2 Postupné spracovávanie requestov

V aplikácii pomocou Express modulu naroutujeme najskôr cez validačnú middleware do správneho controllera, v ktorom sa zavolá metóda processRequest(body, reqInfo). A

hneď sa posiela úspešný response, čo zabezpečuje rýchly čas odpovede na prijatie ceny. Metóda processRequest(body, reqInfo), však musí ceny spracovať, a to je podľa algoritmu z návrhu implementované nasledovne. Z requestu sa vytvorí takzvaný stackObject pomocou metódy createStackObj(body, reqInfo). Tu nastáva rozhodnutie, ak sa žiadne iné requesty nespracovávajú, posunie sa stackObject metóde processProductsFromStackObj(stackObj). V opačnom prípade sa stackObject pridá do haldy, ktorá ho neskôr podľa priority spracuje. Postupné spracovávanie requestov si možte pozrieť na časti kódu 4.2.

```
async processRequest(body, reqInfo) {
    const stackObj = this.createStackObj(body, reqInfo);
    if (this.isStackInUse || __pricesWrapper.pricesUpdating) {
      if (this.scheduleJobStackFinish)
        this.scheduleJobStackFinish.cancelNext();
      this.requestStack.push(stackObj);
      return;
    }

    // stack is in not in use branch
    this.isStackInUse = true;
    this.processProductsFromStackObj(stackObj);
  }
```

Kód 4.2: Postupné spracovávanie requestov

4.1.3 Rekurzívne spracovávanie stackObjectov

Metóda processProductsFromStackObj(stackObj) na každý produkt v stackObjecte zavolá metódu triedy PricesWrapper addTicket. týmto sa ukončilo spracovanie requestu a môže sa rekurzívne zavolať metóda processStack(). V nej sa skontroluje triviálny prípad rekurzie, a to či je halda prázdna. Ak je, znamená to, že sa spracovávanie cien dokončilo, ak nie, z haldy sa vyberie pomocou funkcie pop currentStackObj s najvyššou prioritou a rekurzívne sa zavolá metóda processProductsFromStackObj(currentStackObj) práve s týmto stackObjectom. Rekurzívne spracovávanie stackObjectov si môžete pozrieť na časti kódu 4.3.

```
async processProductsFromStackObj(stackObj) {
    const { products, day } = stackObj;
    const { ccProductId } = products[0];
    const resortAcronym =
      __pricesWrapper.getResortAcronymByProductId(ccProductId);
    for (const product of products) {
      await __pricesWrapper.addTicket(day, product);
    }
    this.processStack();
```

```
10    }
11  processStack() {
12      if (this.requestStack.length === 0) {
13        // Stack is empty branch
14        this.isStackInUse = false;
15        this.stackLastFinish = moment();
16        // send data to net-success
17        __eventEmitter.emit(
18          'axaRequestProcessingFinish',
19          this.resortsChanged
20        );
21        return;
22      }
23
24      const currentStackObj = this.requestStack.pop();
25      this.processProductsFromStackObj(currentStackObj);
26    }
```

Kód 4.3: Rekurzívne spracovávanie stackObjectov

4.2 Trieda PricesWrapper

Trieda PricesStack má na starosti dynamické ukladanie cenníku v špecifickom formáte, ako aj uloženie cien do databázy. Pri inicializácii sa vytvorí __prices objekt, v ktorom budú pripravené dáta v špecifickom formáte potrebnom pre rýchlu odozvu na dopyt od spoločnosti NETSUCCESS.

4.2.1 Ukladanie ceny lístka

Metóda addTicket(day, rawProduct) najskôr v databáze podľa ccProductId nájde produkt, teda typ lístka, pre ktorý prišla cena. Potom v cykle pre všetky kategórie pomocou metódy createOrUpdateTicket(params) databázového modelu Ticket vytvorí, alebo ak lístok už existuje updatne záznam s cenou v databáze. Ak je dátum lístka z aktuálnej sezóny zavolá sa metóda createOrUpdateTmrProduct(ticket, tmrPricesDay, resortAcronym), ktorá má na starosti dynamické ukladanie cenníku v špecifickom formáte pre firmu NETSUCCESS. Ak je lístok typu online zoberie sa aj offline cena a znova sa zavolá metóda createOrUpdateTmrProduct(ticket, tmrPricesDay, resortAcronym) s typom lístku offline a offline cenou. Po každom zavolaní tejto metódy sa ceny v __prices objekte preusporiadajú pomocou metódy sortPrices(). Nakoniec, ak je cena na dnešný deň emitne sa event todayPriceArrival, ktorý dá vedieť triede PricesStack, že po skončení práce haldy má premietnuť nové ceny na obrazovkách s cenníkmi v

strediskách. Opísanú časť kódu metódy addTicket(day, rawProduct) si možte pozrieť na časti kódu 4.4.

```
async addTicket(day, rawProduct) {
  const { ccProductId, categories } = rawProduct;
  const product = this.getProductById(ccProductId);
  if (!product) return;

  const { productType, resortAcronym } = product;

  const tmrPricesDay = moment(day).format('DD/MM/YYYY');

  for (const category of categories) {
    const { categoryId, offlinePrice, price } = category;
    const params = {
      ageCategoryAcronym: categoryId,
      date: day,
      ticketType: productType,
      productId: ccProductId,
      price,
    };

    let ticket = await this.ticketModel.createOrUpdateTicket(params);
    if (this.isTicketDateFromCurrentSeason(day, resortAcronym))
      this.createOrUpdateTmrProduct(ticket, tmrPricesDay,
  resortAcronym);
    this.sortPrices();

    if (productType !== 'online' || resortAcronym === 'mg') continue;

     params.ticketType = 'offline';
     params.price = offlinePrice;
     ticket = await this.ticketModel.createOrUpdateTicket(params);
     if (!this.isTicketDateFromCurrentSeason(day, resortAcronym))
       continue;
     this.createOrUpdateTmrProduct(ticket, tmrPricesDay,
  resortAcronym);
     this.sortPrices();

     const today = moment().format('YYYY-MM-DD');
     if (day === today)
       __eventEmitter.emit('todayPriceArrival', rawProduct);
   }
 }
```

Kód 4.4: Ukladanie ceny lístka

4.2.2 Dynamicke ukladanie cenníku v špecifickom formáte

Metóda triedy createOrUpdateTmrProduct(ticket, tmrPricesDay, resortAcronym) najskôr poskladá ccProductId v špecifickom formáte pomocou metódy getCcProductId(productId, ticketType). Potom sa poskladajú kategórie pomocou metódy createTmrProductCategories(ageCategoryAcronym, price). A následne už môžeme v metóde findOrCreateDayInResortJson(resortAcronym, day, ccProductId, categories) dynamicky uložiť cenník v špecifickom formáte. Je tu ešte jedno špecifikum a to rezorty Tatranská Lomnica a Štrbské Pleso, ktoré sa ukladajú aj pod akronymom VT ako Vysoké Tatry. Opísanú časť kódu metódy createOrUpdateTmrProduct(ticket, day, resortAcronym) si možte pozrieť na časti kódu 4.5.

```
1  createOrUpdateTmrProduct(ticket, day, resortAcronym) {
2    const { ageCategoryAcronym, ticketType, price, productId } =
     ticket;
3
4    const ccProductId = this.getCcProductId(productId, ticketType);
5
6    const categories = this.createTmrProductCategories(
7      ageCategoryAcronym,
8      price
9    );
10
11   this.findOrCreateDayInResortJson(
12     resortAcronym,
13     day,
14     ccProductId,
15     categories
16   );
17   if (resortAcronym === 'tl' || resortAcronym === 'sp')
18     this.findOrCreateDayInResortJson('vt', day, ccProductId,
     categories);
19 }
```

Kód 4.5: Ukladanie ceny lístka

Metóda findOrCreateDayInResortJson(resortAcronym, day, ccProductId, categories) hľadá v __prices objekte podľa nasledujúcich parametrov a vo vymenovanom poradí: deň, produkt, kategória. Ak v objekte dáta ešte nie sú, vytvorí ich. Problémom samotného špecifického formátu je to že dni, produkty aj kategórie sú uložené v poliach teda ich treba prehľadávať so zložitosťou O(dĺžka poľa). Jednoduchšie a rýchlejšie by bolo hašovať kľuč na hodnotu, ale my špecifický formát nevieme ovplyvniť, musíme dáta poslať tak, ako ich chcú v NETSUCCESS.

CENNÍK SKIPASOV
SKIPASS PRICELIST

JASNÁ
NÍZKE TATRY

	GOPASS.sk online e-shop			GOPASS Tickets platba len kartou | card payment only			POKLADŇA platba v hotovosti alebo kartou | cash or card payment		
	Dospelý Adult	Junior & Senior	Dieťa Child	Dospelý Adult	Junior & Senior	Dieťa Child	Dospelý Adult	Junior & Senior	Dieťa Child
1 DEŇ | DAY	53 €	43 €	38 €	53 €	43 €	38 €	59 €	48 €	42 €
2 DNI | DAYS	96 €	77 €	67 €	103 €	83 €	72 €	115 €	92 €	81 €
3 DNI | DAYS	135 €	108 €	95 €	155 €	124 €	108 €	172 €	138 €	121 €
4 DNI | DAYS	168 €	134 €	118 €	200 €	160 €	140 €	222 €	178 €	156 €
5 DNÍ | DAYS	200 €	160 €	140 €	244 €	196 €	171 €	272 €	218 €	190 €
6 DNÍ | DAYS	229 €	183 €	161 €	287 €	229 €	201 €	319 €	255 €	224 €

Obr. 4.1: Obrazovka s cenníkmi.

4.3 Premietnutie cien

V triede PricesStack je na event axaRequestProcessingFinish, ktorý sa vyvolá po dokončení spracovávanie requestov haldou, nastavená funkcia, ktorá pomocou technológie Socket IO vyvolá opätovné načítanie obrazoviek s cenami. Taktiež sa po úspešnom spracovaní cien čaká 10 sekúnd, či ešte ďalši request nepríde, následne sa posiela get request do NETSUCCESS pomocou modulu Axios a taktiež mail na prednastavené mailové adresy o úspešnom spracovaní cenníkov. Na posielanie mailov je použitý modul Nodemailer.

4.4 Obrazovka s cenníkmi

Trida PricesWrapper má metódu getResortPricesViewObj(resort, date), ktorá pre rezort a deň v parametroch funkcie poskladá objekt resortPricesObj z __prices objektu. Objekt resortPricesObj obsahuje dáta potrebné na obrazovky s cenníkmi do stredísk. Tie sú dynamicky vyplnené pomocou modulu EJS. Snímky obrazoviek môžete vidieť na obrázku 4.1.

4.5 Testovanie

Na testovanie aplikácie sme si vytvorili server, ktorý posielal POST requesty s cenami. A taktiež server ktorý bol maketa serveru v NETSUCCESS a prijímal request po úspešnom prijatí cien.

```
POST 200 0.519 ms - 173
POST 200 0.593 ms - 173
POST 200 0.649 ms - 173
POST 200 0.787 ms - 173
POST 200 0.954 ms - 173
POST 200 0.914 ms - 173
POST 200 0.940 ms - 173
POST 200 0.636 ms - 173
POST 200 0.845 ms - 173
POST 200 0.521 ms - 173
POST 200 0.611 ms - 173
POST 200 0.589 ms - 173
```

Obr. 4.2: Výpis odozvy na prijatie POST requestu s cenami.

```
GET sm 200 20.440 ms
GET jt 200 20.168 ms
GET sp 200 8.294 ms
GET nt 200 13.474 ms
GET sz 200 23.318 ms
GET tl 200 20.794 ms
GET vt 200 18.510 ms
```

Obr. 4.3: Výpis odozvy na dopyt po cenách v špecifickom formáte pre rôzne strediská.

4.6 Nasadenie do prevádzky

Do prevádzky bolo riešenie nasadené koncom roka 2021 na servery bežiace na softwari Nginx. Riešenie úspešne zrýchlilo časy odozvy, či už na prijatie requestu s cenou tak aj na dopyt po cenách v špecifickom formáte. Prijatie POST requestu s cenou má odozvu okolo 1 ms. Názorný výpis môžete vidieť na obrázku 4.2. Dopyt po cenách sa zrýchlil z približne 500 až 2000 ms na odozvu okolo 20 ms. Názorný výpis môžete vidieť na obrázku 4.3. Čas spracovávania haldy možete vidieť v tabulke 4.1. Pri krajnom prípade, teda importe celých sezón pre 8 stredísk, je to približne štvrť hodina.

Tabuľka 4.1: Čas spracovania haldy v závyslosti na počte importovaných dní a rezortov.

Počet dní	Počet rezortov	Čas (min : s)
150	8	14:02
150	1	2:25
30	8	2:44
30	1	0:28
1	8	0:06
1	1	<0:01

Záver

Bakalársku prácu sme robili vo firme Tatry Moutain Resorts, a.s. prevádzkovateľa turistických rezortov v regióne strednej a východnej Európy. Spoločnosť implementovala nástroj na manažovanie vyťaženosti stredísk, dynamické ceny lístkov – Flexi ceny. Prvá verzia dynamických cenníkov z roku 2019 mala problém s pomalou odozvou na dopyt po cenách, a aj s pomalou odozvou na prijatie requestov s novým cenníkom . Preto sa ju spoločnosť rozhodla prerobiť.

Cieľom tejto bakalárskej práce bolo vytvoriť novú verziu, ktoré zabezpečí čo najefektívnejšie spracovanie cien. Ceny, od spoločnosti tretej strany AXASOFT, bolo treba spracovať s rýchlym časom odozvy na POST request a premietnuť tieto zmeny do obrazoviek s cenami lístkov v strediskách. V krajnom prípade, pri nahrávaní celej sezóny pre všetky strediská, je treba spracovať vyše 160 000 POST requestov. Pri spracovaní bolo treba myslieť na rýchlu dostupnosť cenníkov v špecifickom formáte pre spoločnosť NETSUCCESS, prevádkovateľa webových stŕanok stredísk. Aplikácia mala obsahovať obrazovky na zobrazenie aktuálnych cenníkov v lyžiarskom stredisku.

Pri návrhu sme sa museli prispôsobiť formátom už zabehnutých riešení. Preto na splnenie cieľov bolo treba vymyslieť adekvátny databázový model, algoritmus spracovania veľkého množstva prichádzajúcich requestov s cenami podľa priority a využiť dynamické programovanie na zrýchlenie času odozvy po dopyte po cenách v špecifickom formáte. Premietnutie zmien do obrazoviek s cenníkmi bolo zabezpečené pomocou WebSocket technológie.

Všetky požiadavky boli úspešne implementované a podarilo sa nám efektívne zredukovať čas odozvy, či už pri príchode cenníkov, ale hlavne pri dopyte na ceny v špecifickom formáte. Riešenie bolo úspešne otestované pomocou makiet serverov vyššie spomínaných spoločností a nasadené do produkcie koncom roka 2021, z dôvodu začatia lyžiarskej sezóny. Obrazovky s cenníkmi možno vidieť v strediskách s vždy aktuálnymi údajmi.

Táto bakalárska práca je súčasťou väčšieho projektu, v ktorom plánujeme, alebo už je implementované požívateľské a admin rozhranie, reporty spracovávania cien, ďalšie obrazovky zobrazujúce podmienky v strediskách a iná potrebná funkcionalita pre spoločnosť Tatry Moutain Resort a.s. Zároveň sa kontinuálne pracuje na novej verzii pre zimnú sezónu 2022/23. Kompletný zdrojový kód je majetkom firmy Tatry Moutain

Resort a.s. a je súčasťou komerčného projektu, ktorý je nasadený v ostrej prevádzke. Obsahuje citlivé dáta a tak z bezpečnostných dôvodov, po dohode s firmou, v ktorej bol vyvíjaný, nie je zverejnený.

Literatúra

[1] Axios. [Citované 2022-1-15] Dostupné z https://www.npmjs.com/package/axios.

[2] Bcryptjs. [Citované 2022-1-15] Dostupné z https://www.npmjs.com/package/bcryptjs.

[3] Compression. [Citované 2022-1-15] Dostupné z https://www.npmjs.com/package/compression.

[4] Dynamicpricing. [Citované 2022-1-15] Dostupné z https://softjourn.com/insights/dynamic-pricing-in-ticketing.

[5] Ejs. [Citované 2022-1-15] Dostupné z https://www.npmjs.com/package/ejs.

[6] Nginx. [Citované 2022-1-15] Dostupné z https://blog.logrocket.com/how-to-run-a-node-js-server-with-nginx/.

[7] Nginx load balancing. [Citované 2022-1-15] Dostupné z https://www.nginx.com/blog/load-balancing-with-nginx-plus-part-2/.

[8] Nodemailer. [Citované 2022-1-15] Dostupné z https://www.npmjs.com/package/nodemailer.

[9] Sequelize. [Citované 2022-1-15] Dostupné z https://sequelize.org/v7/.

[10] Sequelize educative.io. [Citované 2022-1-15] Dostupné z https://www.educative.io/edpresso/what-is-sequelizejs.

[11] Socket.io. [Citované 2022-1-15] Dostupné z https://www.npmjs.com/package/socket.io.

[12] Evan Hahn. *Express in Action: Writing, building, and testing Node. js applications.* Simon and Schuster, 2016.

[13] Bradley N Miller and David L Ranum. *Problem solving with algorithms and data structures using python Second Edition.* Franklin, Beedle & Associates Inc., 2011.

[14] Hezbullah Shah. Node. js challenges in implementation. *Global Journal of Computer Science and Technology*, 2017.